**Εθνικό Μετσόβιο Πολυτεχνείο**

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τομέας Πληροφορικής και Τεχνολογίας Υπολογιστών

# Μοντελοποίηση εφαρμογών και τελεστών μεγάλων δεδομένων σε περιβάλλοντα υπολογιστικών νεφών

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

**Ιωάννης Κ. Γιαννακόπουλος**

Αθήνα, Νοέμβριος 2018

Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τομέας Πληροφορικής και Τεχνολογίας Υπολογιστών

# Μοντελοποίηση εφαρμογών και τελεστών μεγάλων δεδομένων σε περιβάλλοντα υπολογιστικών νεφών

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

**Ιωάννης Κ. Γιαννακόπουλος**

**Συμβουλευτική Επιτροπή:** Νεκτάριος Κοζύρης
Δημήτριος Τσουμάκος
Παναγιώτης Τσανάκας

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 26η Νοεμβρίου 2018.

| | | |
|---|---|---|
| . . . . . . . . . . . . | . . . . . . . . . . . . | . . . . . . . . . . . . |
| Νεκτάριος Κοζύρης | Δημήτριος Τσουμάκος | Παναγιώτης Τσανάκας |
| Καθηγητής ΕΜΠ | Αναπληρωτής Καθηγητής | Καθηγητής ΕΜΠ |
| | Ιόνιο Πανεπιστήμιο | |
| . . . . . . . . . . . . | . . . . . . . . . . . . | . . . . . . . . . . . . |
| Γεώργιος Πάλλης | Ιωάννης Κωτίδης | Δήμητρα Ρουσοπούλου |
| Επίκουρος Καθηγητής | Αναπληρωτής Καθηγητής | Αναπληρώτρια Καθηγήτρια |
| Πανεπιστήμιο Κύπρου | Οικον. Πανεπιστήμιο Αθηνών | ΕΚΠΑ |
| | . . . . . . . . . . . . | |
| | Ευαγγελία Πιτουρά | |
| | Καθηγήτρια | |
| | Πανεπιστήμιο Ιωαννίνων | |

Αθήνα, Νοέμβριος 2018

. . . . . . . . . . . .

**Ιωάννης Κ. Γιαννακόπουλος**

Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών

Εθνικό Μετσόβιο Πολυτεχνείο (2018)

# Contents

**Ευχαριστίες** **xv**

**Abstract** **xvii**

**Γλωσσάριο Τεχνικών Όρων** **xix**

**Εκτεταμένη Περίληψη** **1**

0.1 Εισαγωγή . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

0.1.1 Κίνητρο . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

0.1.2 Συνεισφορές . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

0.2 Προσαρμοστική Μοντελοποίηση Απόδοσης Τελεστή Μεγάλων Δεδομένων . . 4

0.2.1 Θεμελίωση του προβλήματος . . . . . . . . . . . . . . . . . . . . . 4

0.2.2 Μεθοδολογία . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

0.2.3 Παρατηρήσεις . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

0.3 Βασισμένη στο Περιεχόμενο Μοντελοποίηση Τελεστών Ανάλυσης Δεδομένων . 9

0.3.1 Θεμελίωση του προβλήματος . . . . . . . . . . . . . . . . . . . . . 9

0.3.2 Μεθοδολογία . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

0.3.3 Παρατηρήσεις . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

0.4 Συμπεράσματα . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

**1 Introduction** **17**

1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

1.2 Contribution . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

1.3 Document Outline . . . . . . . . . . 24

**2 Adaptive Performance Modeling of Big Data Applications and Operators 27**

2.1 Overview . . . . . . . . . . 28
2.2 Preliminaries . . . . . . . . . . 30
2.2.1 Problem formulation . . . . . . . . . . 30
2.2.2 Decision Trees . . . . . . . . . . 31
2.3 Profiling Methodology . . . . . . . . . . 32
2.3.1 Method overview . . . . . . . . . . 32
2.3.2 Decision Tree Expansion . . . . . . . . . . 34
2.3.3 Adaptive Sampling . . . . . . . . . . 37
2.3.4 Modeling . . . . . . . . . . 41
2.3.5 Complexity analysis . . . . . . . . . . 42
2.3.6 An end-to-end example . . . . . . . . . . 43
2.4 Experimental Evaluation . . . . . . . . . . 44
2.4.1 Comparison of end-to-end profiling methods for varying Sampling Rate 47
2.4.2 Impact of performance function complexity and dimensionality . . . . . 49
2.4.3 Impact of per-iteration number of deployments . . . . . . . . . . 50
2.4.4 Impact of oblique boundaries . . . . . . . . . . 52
2.4.5 Cost-aware profiling . . . . . . . . . . 54
2.5 Discussion . . . . . . . . . . 55

**3 A Content-Based Approach for Modeling Analytics Operators 57**

3.1 Overview . . . . . . . . . . 58
3.2 Preliminaries . . . . . . . . . . 60
3.2.1 Problem Description . . . . . . . . . . 60
3.2.2 Operators and Dataset Properties . . . . . . . . . . 60
3.3 Methodology . . . . . . . . . . 62
3.3.1 Methodology Overview . . . . . . . . . . 62
3.3.2 Similarity Estimation . . . . . . . . . . 63
3.3.3 Dataset Space Projection . . . . . . . . . . 67
3.3.4 Modeling . . . . . . . . . . 69
3.3.5 Optimizations . . . . . . . . . . 69
Approximate Similarity Matrices . . . . . . . . . . 69
Online Indexing . . . . . . . . . . 71
3.4 Accommodating graph datasets . . . . . . . . . . 72
3.4.1 Operators and Graph properties . . . . . . . . . . 72

3.4.2 Similarity metric estimation . . . 73
3.5 Experimental Evaluation . . . 75
3.5.1 Dataset Space Construction . . . 77
3.5.2 Operator Modeling . . . 81
3.5.3 Combining Similarity Metrics . . . 83
3.5.4 Distribution Similarity Granularity . . . 84
3.5.5 Approximate Similarity Matrices . . . 85
3.5.6 Online Indexing . . . 86
3.5.7 Extension to graph data . . . 88
3.6 Discussion . . . 95

**4 Related Work** **97**
4.1 Performance Modeling of Big Data Applications . . . 97
4.2 Data-driven Modeling of Analytics Operators . . . 103

**5 Conclusions and Future Directions** **109**

**Publications** **113**

**Bibliography** **117**

**A Cloud Application Deployment with Transient Failure Recovery** **131**
A.1 Introduction . . . 131
A.2 Related Work . . . 134
A.3 Application Deployment . . . 137
A.3.1 Architecture . . . 138
A.3.2 Deployment Model . . . 139
A.3.3 Error Recovery . . . 141
A.3.4 Idempotency . . . 145
A.4 Implementation Aspects . . . 146
A.4.1 Optimizations . . . 147
A.4.2 Portability and Reusability . . . 148
A.5 Experimental Evaluation . . . 150
A.5.1 Deployment Model Scalability . . . 153
A.5.2 Transient Error Frequency . . . 155
A.5.3 Snapshot Implementation Overhead . . . 157
A.5.4 End-to-End Performance Comparison . . . 159
A.6 Conclusions and Future Work . . . 161

# List of Figures

1 Επισκόπηση μεθόδου προσαρμοστικής μοντελοποίησης . . . . . . . . . . . . . 5
2 Επισκόπηση μεθοδολογίας μοντελοποίησης της εξόδου ενός τελεστή . . . . . . 11

1.1 The 4 V's of Big Data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
1.2 Two dimensions of modeling operator behavior . . . . . . . . . . . . . . . . . . 21

2.1 Modeling errors of different profiling schemes . . . . . . . . . . . . . . . . . . . 28
2.1a Random sampling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
2.1b Uncertainty sampling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
2.2 Example of an Oblique Decision Tree . . . . . . . . . . . . . . . . . . . . . . . . 32
2.2a Oblique Decision Tree structure . . . . . . . . . . . . . . . . . . . . . . . . . 32
2.2b Space partitioning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
2.3 Method overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
2.4 An example of Decision Tree Expansion for $f(x, y) = (x + y)\exp(x + y)$ . . . 36
2.4a 3d projection of the performance function . . . . . . . . . . . . . . . . . . . 36
2.4b Top view of the performance function . . . . . . . . . . . . . . . . . . . . . 36
2.4c Top view with samples . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
2.4d Candidate split lines . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
2.5 Distribution of samples for different weights . . . . . . . . . . . . . . . . . . . . 39
2.5a Original function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
2.5b $w_{error} = 1.0, w_{size} = 0.0$ . . . . . . . . . . . . . . . . . . . . . . . . . 39
2.5c $w_{error} = 1.0, w_{size} = 0.5$ . . . . . . . . . . . . . . . . . . . . . . . . . 39
2.5d $w_{error} = 0.0, w_{size} = 1.0$ . . . . . . . . . . . . . . . . . . . . . . . . . 39

2.6 Algorithm execution for Wordcount . . . . . 43
2.6a First iteration . . . . . 43
2.6b Second iteration . . . . . 43
2.7 Accuracy vs sampling rate (MSE) . . . . . 47
2.7a k-means . . . . . 47
2.7b Bayes . . . . . 47
2.7c Wordcount . . . . . 47
2.7d MongoDB . . . . . 47
2.8 Accuracy vs sampling rate (MAE) . . . . . 48
2.8a k-means . . . . . 48
2.8b Bayes . . . . . 48
2.8c Wordcount . . . . . 48
2.8d MongoDB . . . . . 48
2.9 Per-iteration number of deployments evaluation . . . . . 51
2.9a Accuracy . . . . . 51
2.9b Execution time . . . . . 51
2.10 Accuracy for flat and oblique cuts . . . . . 53
2.10a Accuracy for k-means . . . . . 53
2.10b Accuracy for EXPABS - entire space . . . . . 53
2.10c Accuracy for EXPABS - ABN space . . . . . 53
2.10d Relative error for different test sets . . . . . 53

3.1 Relationship between Data Distribution and Operator behavior . . . . . 58
3.1a Statistical distribution of 6 datasets . . . . . 58
3.1b Output of operators for the 6 datasets . . . . . 58
3.2 Methodology workflow . . . . . 63
3.3 Distribution Similarity Example . . . . . 65
3.4 Example of MDS and Sammon mapping . . . . . 69
3.5 Dataset distances . . . . . 70
3.6 Node degree estimation in different levels . . . . . 74
3.6a Graph overview . . . . . 74
3.6b Adjacency list and node degrees . . . . . 74
3.7 Two example graphs with average degree 2 . . . . . 75
3.7a Ring graph with average degree 2 . . . . . 75
3.7b Graph with triangles with average degree 2 . . . . . 75
3.8 Dataset spaces, $GoF$ and $E_s$ plots . . . . . 78
3.8a Goodness-of-Fit plot . . . . . 78

3.8b Sammon Stress $E_s$ plot . . . . . 78
3.8c *HPO* dataset space . . . . . 78
3.8d *WEA* dataset space . . . . . 78
3.9 Combining data properties . . . . . 83
3.9a Modeling accuracy for WEA-AVG . . . . . 83
3.9b Modeling accuracy for WEA-SUM . . . . . 83
3.10 Distribution Similarity Granularity . . . . . 85
3.10a Modeling accuracy for varying number of partitions . . . . . 85
3.10b Execution time of different phases . . . . . 85
3.10c Sammon Stress for varying dimensionality . . . . . 85
3.10d Modeling accuracy for varying dimensionality . . . . . 85
3.11 Approximate Similarity Matrix Evaluation . . . . . 86
3.11a Achieved speedup vs % datasets . . . . . 86
3.11b Relative modeling error vs % datasets . . . . . 86
3.12 Online Indexing Evaluation . . . . . 87
3.12a Relative Sammon Stress vs # of new datasets . . . . . 87
3.12b Relative Modeling Error vs # of new datasets . . . . . 87
3.13 Similarity Metrics Comparison for *TW* Dataset . . . . . 93
3.13a Spectral Radius . . . . . 93
3.13b Eigenvector C. . . . . . 93
3.13c Betweenness C. . . . . . 93
3.13d Edge Betweenness C. . . . . . 93
3.13e Closeness C. . . . . . 93
3.13f PageRank . . . . . 93
3.13g Execution Time (sec) . . . . . 93
3.14 Similarity Metric Comparison for Betweenness C. . . . . . 94
3.14a *AS* dataset . . . . . 94
3.14b *BA* dataset . . . . . 94

A.1 AURA Architecture . . . . . 138
A.2 Message exchange between different software modules and circular dependencies 140
A.3 Wordpress deployment graph with 1 Web Server and 1 Database Server . . . . . 143
A.4 Message exchange between modules of different multiplicity . . . . . 147
A.5 Wordpress deployment graph with 2 Web Servers and 1 Database Server . . . . . 152
A.6 Hadoop deployment graph with 1 master and 2 slave nodes . . . . . 152
A.7 Execution times of different deployment phases for varying module multiplicity using vanilla images . . . . . 154

A.8 Execution times of different deployment phases for varying module multiplicity using prebaked images . . . . . . . . . . . . . . . . . . . . . . . . . . . . 154
A.9 Number of script executions and configuration times for varying error probability 156
A.10 Overhead of AUFS and BTRFS for Wordpress . . . . . . . . . . . . . . . . . . . . . 158
A.11 Overhead of AUFS and BTRFS for Hadoop . . . . . . . . . . . . . . . . . . . . . 158

# List of Tables

2.1 Applications under profiling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
2.2 Synthetic performance functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
2.3 Accuracy vs function complexity . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
2.4 MSE decrease % for linear classifiers . . . . . . . . . . . . . . . . . . . . . . . . . 52
2.5 MSE and Cost for different cost weights . . . . . . . . . . . . . . . . . . . . . . . 54

3.1 Operators and Dataset Properties . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
3.2 Datasets and Operators . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76
3.3 Modeling Accuracy and Execution Speedup of Operators . . . . . . . . . . . . 80
3.4 Datasets overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 89
3.5 Modeling Errors and Execution Speedup for Different Sampling Rates . . . . . . 90

A.1 Requirements of AUFS and BTRFS . . . . . . . . . . . . . . . . . . . . . . . . . . 158
A.2 Relative Deployment Time for Openstack Heat vs AURA . . . . . . . . . . . . . 160

# Ευχαριστίες

Η παρούσα διατριβή εκπονήθηκε στο Εργαστήριο Υπολογιστικών Συστημάτων της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου, υπό την επίβλεψη των καθηγητών Δημήτριου Τσουμάκου και Νεκτάριου Κοζύρη. Περιέχει την έρευνα και τα αποτελέσματα των μεταπτυχιακών μου σπουδών κατά την παρουσία μου στη ΣΗΜΜΥ του ΕΜΠ. Καθώς λοιπόν αυτός ο κύκλος κλείνει, θα ήθελα να εκφράσω την ευγνωμοσύνη μου στους ανθρώπους που είχαν καθοριστική συμβολή στην ολοκλήρωση αυτής της διατριβής.

Αρχικά, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Νεκτάριο Κοζύρη για την ευκαιρία που μου έδωσε να ασχοληθώ με σύγχρονα ερευνητικά θέματα σε ένα άρτιο ακαδημαϊκό περιβάλλον. Η συνεχής υποστήριξή, τόσο υλική όσο και πνευματική, καθώς και η καθοδήγησή που μου παρείχε όλα αυτά τα χρόνια έπαιξαν καθοριστικό ρόλο στην ολοκλήρωση αυτής της προσπάθειας. Το φιλόξενο ερευνητικό περιβάλλον μέσα στο οποίο δούλεψα, έμαθα και ολοκλήρωσα τις σπουδές μου έχει δημιουργηθεί από εκείνον και προωθεί την καινοτομία και την υγιή ερευνητική σκέψη.

Θα ήθελα επίσης να ευχαριστήσω τον επιβλέποντα καθηγητή μου Δημήτρη Τσουμάκο για την πολύτιμη βοήθειά του και τη συνεχή προσπάθειά του καθ' όλη τη διάρκεια των σπουδών μου για την επίτευξη ενός ερευνητικού αποτελέσματος υψηλού επιπέδου. Πέρα από τις πολύτιμες συμβουλές του που με βοήθησαν να αναπτύξω όλες εκείνες τις ικανότητες που πρέπει να έχει ένας ερευνητής, όπως είναι η συγγραφή επιστημονικού κειμένου και η κριτική σκέψη, με έμαθε να σκέφτομαι ερευνητικά και μου μετέδωσε τη δίψα για συνεχή μάθηση. Η παρούσα διατριβή δεν θα είχε ολοκληρωθεί ποτέ χωρίς τη δική του συμβολή και για το λόγο αυτό του οφείλω την ευγνωμοσύνη μου.

Κατά τη διάρκεια των σπουδών μου, σημείο αναφοράς υπήρξε το Εργαστήριο Υπολογιστικών Συστημάτων. Η καθημερινή μου συναναστροφή με όλα τα μέλη του έπαιξε καθοριστικό ρόλο στην εξέλιξή μου και την διαμόρφωσή μου σε ερευνητή. Θέλω να ευχαριστήσω θερμά του μεταδιδακτορικούς ερευνητές Δρ. Γιάννη Κωνσταντίνου και Δρ. Κατερίνα Δόκα για την πολύτιμη βοήθειά τους, τις συζητήσεις μας και τις συμβουλές που μου προσέφεραν όλα αυτά τα χρόνια. Επίσης, θα ήθελα να ευχαριστήσω θερμά τον υ.δ. Γιάγκο Μυτιλήνη και τον Δρ. Νίκο Παπαηλίου, με τους οποίους μοιραστήκαμε την καθημερινότητα μας στο μεγαλύτερο μέρος των σπουδών μας. Οι συνεχείς συζητήσεις μας και η καθημερινή αλληλεπίδρασή μας με βοήθησε πολύ στην πορεία μου ως υποψήφιο διδάκτορα. Τέλος, θα ήθελα να ευχαριστήσω όλα τα μέλη του Εργαστηρίου Υπολογιστικών Συστημάτων για την συνεργασία τους και τη βοήθεια που μου προσέφεραν όλα αυτά τα χρόνια.

Για το τέλος, άφησα τους σημαντικότερους ανθρώπους στη ζωή μου, την οικογένειά μου, τη σύντροφο μου και τους φίλους μου οι οποίοι ήταν δίπλα μου σε όλη αυτή την πορεία. Οι άνθρωποι αυτοί δεν βοήθησαν στην συγγραφή αυτής της διατριβής. Βρίσκονταν όμως πάντα εκεί για να απορροφούν τους κραδασμούς, να μοιράζονται τη χαρά και τη λύπη και να δίνουν ελπίδα. Για τους λόγους αυτούς, τους ευχαριστώ μέσα από την καρδιά μου.

# Abstract

The Big Data revolution has created new requirements for the design of applications and operators that are able to handle the volume of the data sources. The adoption of distributed architectures and the increasing popularity of the Cloud paradigm has complexed their structure, making the problem of modeling their behavior increasingly difficulty. Moreover, the wide variety of the existing datasets have complicated the problem of selecting the *appropriate* inputs for a given operator, since the examination of the data utility for a given workflow is a largely manual process that requires exhaustive execution for the entirety of the available datasets. This thesis attempts to model the behavior of an arbitrary Big Data operator from two different viewpoints.

First, we wish to model the operator's performance when deployed under different resource configurations. To this end, we present an adaptive performance modeling methodology that relies on recursively partitioning the configuration space in disjoint regions, distributing a predefined number of samples to each region based on different region characteristics (i.e., size, modeling error) and deploying the given operator for the selected samples. The performance is, then, approximated for the entire space using a combination of linear models for each subregion. Intuitively, this approach attempts to compromise the contradicting aspects of exploring the configuration space and exploiting the obtained knowledge through focusing on areas with higher approximation error.

Second and in order to accelerate data analysis, we wish to model the operator's output when deployed over different datasets. Based on the observation that similar datasets tend to affect the operators that are applied to them similarly, we propose a content-based methodology that models the output of a provided operator for all datasets. Our approach measures the similarity between the different datasets in the light of some fundamental properties commonly used in

data analysis tasks, i.e., the statistical distribution, the dataset size and the tuple ordering. These similarities are, next, projected to a low dimensional metric space that is utilized as an input domain by Neural Networks in order to approximate the operator's output for all datasets, given the actual operator output for a mere subset of them.

Our evaluation, conducted using several real-world operators applied for real and synthetic datasets, indicated that the introduced methodologies manage to accurately model the operator's behavior from both angles. The adoption of a divide-and-conquer approach that equally respects space exploration and knowledge exploitation for the performance modeling part, proved to be the main reason that our scheme outperforms other state-of-the-art methodologies. On the same time, the construction of a low dimensional dataset metric space for the second part, proved to be particularly informative in order to allow Machine Learning models to approximate operator output for a wide variety of operators with diverse characteristics.

# Γλωσσάριο Τεχνικών Όρων

| **Αγγλικός Όρος** | **Ελληνικός Όρος** |
|---|---|
| Application | Εφαρμογή |
| Big Data | Μεγάλα Δεδομένα |
| Cloud Computing | Υπολογιστικές Νεφέλες |
| Dataset | Σύνολο Δεδομένων |
| Decision Tree | Δέντρο Απόφασης |
| Infrastructure (cloud) | Υποδομή (υπολογιστική) |
| Machine Learning | Μηχανική Μάθηση |
| Performance | Απόδοση |
| Resources | Υπολογιστικοί πόροι |
| Operator | Τελεστής |
| Virtual Machine (VM) | Εικονική Μηχανή |

# Εκτεταμένη Περίληψη

## 0.1 Εισαγωγή

### 0.1.1 Κίνητρο

Η έλευση της εποχής των Μεγάλων Δεδομένων έχει φέρει επανάσταση στον τρόπο με τον οποίο ο κόσμος βλέπει και αλληλεπιδρά με αυτά. Στις μέρες μας, ένας αυξανόμενος αριθμός επιχειρήσεων στηρίζεται στη συλλογή, διαχείριση και ανάλυση μεγάλων δεδομένων έτσι ώστε να παρέχει προσωποποιημένες υπηρεσίες υψηλού επιπέδου στους πελάτες τους. Το μεγάλο πλεονέκτημα αυτής της πρακτικής είναι ότι η λήψη αποφάσεων γίνεται όλο και περισσότερο βασισμένη στα δεδομένα: Η ανάλυση των Μεγάλων Δεδομένων αποκαλύπτει μοτίβα και παράγει γνώση που οδηγεί τις επιχειρήσεις να παίρνουν καλύτερες αποφάσεις και να δημιουργούν προϊόντα τα οποία ευθυγραμμίζονται με τις ανάγκες των πελατών τους. Παράλληλα η πληθώρα των πηγών δεδομένων που συνεχώς δημιουργούν νέα δεδομένα με υψηλό ρυθμό έχει δώσει την ώθηση στον ακαδημαϊκό κόσμο για να αναζητήσει πιο αποδοτικούς τρόπους να αποθηκεύει, να διαχειρίζεται και να αναλύει τα δεδομένα.

Παρόλα αυτά, η τεχνολογική αυτή επανάσταση έχει δημιουργήσει αρκετές προκλήσεις που αφορούν την σχεδίαση και αρχιτεκτονική τελεστών που εφαρμόζονται σε Μεγάλα Δεδομένα, όπως αναφέρεται και στη βιβλιογραφία [T$^+$13, AI13, LJ12]. Η ανάγκη διαχείρισης ενός μεγάλου όγκου δεδομένων έχει οδηγήσει στην αύξηση της πολυπλοκότητας τόσο της αρχιτεκτονικής όσο και των αλγορίθμων που υλοποιούν οι τελεστές, καθιστώντας το πρόβλημα της μοντελοποίησης της συμπεριφοράς τους αρκετά δυσκολότερο σε σύγκριση με παλιότερες εφαρμογές. Η μοντελοποίηση της συμπεριφοράς ενός τελεστή είναι ένα πολύ σημαντικό πρόβλημα, καθώς παρέχει γνώση σε σχέση με τη λειτουργικότητά του. Σε αυτή τη διατριβή θα ασχοληθούμε με το πρόβλημα της μοντελοποίησης ενός τελεστή Μεγάλων Δεδομένων, εξετάζοντας το από δυο σκοπιές. Πρώτον, θα ασχοληθούμε με τη μοντελοποίηση της απόδοσής του, συναρτήσει των παραμέτρων εγκατάστασης με τις οποίες εγκαθίσταται και δεύτερον, θα ασχοληθούμε με τη μοντελοποίηση της εξόδου του (δηλαδή με το αποτέλεσμά του) όταν αυτός εκτελείται για διαφορετικά σύνολα δεδομένων.

Η μοντελοποίηση της απόδοσης ενός τελεστή ή μιας εφαρμογής είναι ένα πρόβλημα το οποίο έχει μελετηθεί πολύ στη βιβλιογραφία. Στην περιοχή των Μεγάλων Δεδομένων, το πρόβλημα αυτό γίνεται ιδιαίτερα δύσκολο εξαιτίας της πολυπλοκότητας των τελεστών. Η πολυπλοκότητα αυτή οφείλεται σε δύο λόγους. Πρώτον, οι τελεστές Μεγάλων Δεδομένων ακολουθούν, κατά κανόνα, κατανεμημένη αρχιτεκτονική έτσι ώστε να έχουν την απαιτούμενη κλιμακωσιμότητα. Αυτό σημαίνει πως κάθε υπομονάδα του τελεστή μπορεί να εγκατασταθεί με διαφορετικό αριθμό διεργασίων, κάτι που με τη σειρά του αυξάνει τις επιλογές παραμετροποίησης του. Δεύτερον, οι τελεστές Μεγάλων Δεδομένων εγκαθίστανται, κατά κύριο λόγο, σε Cloud υποδομές για μια πληθώρα λόγων που αφορούν την άμεση δημιουργία και αξιοποίηση μεγάλου πλήθους υπολογιστικών πόρων, την ευκολότερη διαχείριση, κ.α. Η προγραμματιστική δημιουργία εικονικών πόρων δίνει τη δυνατότητα στον κάτοχο του τελεστή την δυνατότητα της παραμετροποίησης του περιβάλλοντός του με πολλή μεγάλη λεπτομέρεια. Οι δυο αυτές παράμετροι οδηγούν σε εκθετική αύξηση του χώρου παραμέτρων της εφαρμογής, με αποτέλεσμα οι παραδοσιακές τεχνικές μοντελοποίησης να απαιτούν εκθετικά περισσότερα δείγματα παραμέτρων έτσι ώστε να μπορέσουν να πετύχουν προσέγγιση της απόδοσης με υψηλή ακρίβεια.

Από την άλλη μεριά, η δυνατότητα πρόβλεψης του αποτελέσματος ενός τελεστή που εκτελείται σε ένα σύνολο δεδομένων χωρίς την φυσική εκτέλεσή του, μπορεί να επιταχύνει τη διαδικασία επιλογής ενός συνόλου δεδομένων που πληρεί κάποια κριτήρια και μεγιστοποιεί τη χρησιμότητα των δεδομένων για έναν τελεστή. Από τη σκοπιά αυτή, θέλουμε να μοντελοποιήσουμε την έξοδο ενός τελεστή συναρτήσει των συνόλων δεδομένων στα οποία μπορεί να εκτελεστεί. Η δυσκολία επίλυσης αυτού του προβλήματος έγκειται στο γεγονός ότι τα σύνολα δεδομένων εισόδου δεν ακολουθούν κάποια σχέση διάταξης μεταξύ τους και, συνεπώς, ο μετρικός χώρος εισόδου του προβλήματος μοντελοποίησης δεν ορίζεται. Με άλλα λόγια, απαιτείται η δημιουργία ενός διανύσματος χαρακτηριστικών για κάθε σύνολο δεδομένων το οποίο αποτελεί μοναδικό αναγνωριστικό του και μπορεί να κατασκευάσει έναν μετρικό χώρο που αποκτά σημασιολογία και αντανακλά τις συσχετίσεις μεταξύ των συνόλων δεδομένων.

Πρέπει να τονίσουμε ότι τα δυο προβλήματα που αντιμετωπίζουμε σε αυτή τη διατριβή αντανακλούν τις δυο κυρίαρχες διαστάσεις ενός σύγχρονου τελεστή Μεγάλων Δεδομένων. Αφενός, με τη μοντελοποίηση της απόδοσης θέλουμε να κατασκευάσουμε ένα προφίλ το οποίο μπορεί να βοηθήσει στη λήψη αποφάσεων για τη βέλτιστη λειτουργία του τελεστή από άποψη απόδοσης. Αφετέρου, με τη μοντελοποίηση της εξόδου του τελεστή, θέλουμε να δημιουργήσουμε ένα προφίλ δεδομένων που βοηθά την εύρεση των συνόλων δεδομένων με τη μέγιστη χρησιμότητα για την λειτουργία του.

### 0.1.2 Συνεισφορές

Στην διατριβή αυτή, παρουσιάζουμε δύο μεθοδολογίες που στοχεύουν στη μοντελοποίηση της συμπεριφοράς τελεστών Μεγάλων Δεδομένων, από δύο διαφορετικές οπτικές. Πρώτον, επιλύουμε το πρόβλημα της μοντελοποίησης της απόδοσης ενός τελεστή συναρτήσει των παραμέτρων εγκατάστασης με τις οποίες εκτελείται. Δεύτερον, επιλύουμε το πρόβλημα της μοντελοποίησης της εξόδου ενός τελεστή όταν εκτελείται για διαφορετικά σύνολα δεδομένων. Οι συνεισφορές της διατριβής αυτής συνοψίζονται στα εξής σημεία:

- Εξετάζουμε το πρόβλημα της μοντελοποίησης της συμπεριφοράς ενός τελεστή υπό το πρίσμα της απόδοσής του και της εξόδου του.
- Προτείνουμε μια τεχνική προσαρμοστικής μοντελοποίησης της απόδοσης ενός τελεστή που στηρίζει τη λειτουργία της στην αναδρομική διαμέριση του χώρου παραμέτρων της εφαρμογής, της προσαρμοστικής κατανομής του διαθέσιμου αριθμού δειγμάτων σε κάθε υπόχωρο ανάλογα με το σφάλμα προσέγγισης και το μέγεθός του και, τέλος, της κατασκευής ενός μοντέλου σαν συνδυασμό απλούστερων γραμμικών μοντέλων που αφορούν διαφορετικά χωρία. Η προτεινόμενη μεθοδολογία συμβιβάζει τις δύο αντίρροπες τάσεις της εξερεύνησης του χώρου παραμέτρων και την εκμετάλλευση της γνώσης που έχει αποκτηθεί για το χώρο με βάση τα δείγματα.
- Προτείνουμε μια τεχνική μοντελοποίησης της εξόδου ενός τελεστή συναρτήσει των συνόλων δεδομένων εισόδου που βασίζεται στην ανάλυση και σύγκριση του περιεχόμενου των συνόλων. Συγκεκριμένα, η μεθοδολογία μας συμπεριλαμβάνει τον υπολογισμό της ομοιότητας μεταξύ των διαφορετικών συνόλων δεδομένων, υπό το φως τριών θεμελιωδών ιδιοτήτων τους: (α) της στατιστικής κατανομής τους, (β) του μεγέθους τους και (γ) της σειράς εμφάνισης των στοιχείων τους. Με βάση αυτές τις ομοιότητες, η προσέγγισή μας παράγει έναν μετρικό χώρο που προβάλλει τα διαφορετικά σύνολα δεδομένα σε αυτόν και οι μεταξύ τους αποστάσεις σε αυτών δείχνουν την ομοιότητά τους. Η έξοδος του τελεστή προσεγγίζεται με τη χρήση Νευρωνικών Δικτύων που εκπαιδεύονται για το χώρο αυτό, με βάση εξόδους του τελεστή που έχουν συλλεχθεί για ένα μικρό δείγμα των συνόλων δεδομένων.
- Εκτελούμε λεπτομερή πειραματική αξιολόγηση και για τις δυο μεθοδολογίες, χρησιμοποιώντας πληθώρα πραγματικών και δημοφιλών τελεστών καθώς και διάφορα πραγματικά και συνθετικά σύνολα δεδομένων εισόδου. Η αξιολόγησή μας υποδεικνύει ότι οι προτεινόμενες μεθοδολογίες είναι ικανές να μοντελοποιήσουν τη συμπεριφορά ενός τελεστή και

από τις δυο εξεταζόμενες οπτικές με χαμηλό σφάλμα, ενώ παράλληλα η απόδοσή τους ξεπερνά άλλες ανταγωνιστικές τεχνικές μοντελοποίησης.

## 0.2 Προσαρμοστική Μοντελοποίηση Απόδοσης Τελεστή Μεγάλων Δεδομένων

Στην ενότητα αυτή θα ασχοληθούμε με το πρόβλημα της μοντελοποίησης της απόδοσης μιας εφαρμογής Μεγάλων Δεδομένων συναρτήσει των παραμέτρων που την επηρεάζουν. Η προτεινόμενη μεθοδολογία αντιμετωπίζει το πρόβλημα της μοντελοποίησης σαν ένα παραδοσιακό πρόβλημα προσέγγισης συνάρτησης: Η απόδοση της εφαρμογής αντιμετωπίζεται σαν μια μαθηματική συνάρτηση που έχει ως είσοδο ένα σύνολο παραμέτρων που αντιπροσωπεύουν ρυθμίσεις εγκατάστασης και έχει ως έξοδο μια τιμή που υποδηλώνει την απόδοση. Σκοπός της προτεινόμενης μεθοδολογίας είναι η εξέταση του χώρου παραμέτρων με τρόπο τέτοιο που επιτρέπει την εμβάθυνση σε περιοχές που η απόδοση λαμβάνει ακανόνιστη μορφή που είναι δύσκολο να μοντελοποιηθεί. Για το λόγο αυτό, ο χώρος παραμέτρων διαμερίζεται σε ξένα υποσύνολα, λαμβάνονται δείγματα από κάθε περιοχή για τα οποία η εφαρμογή εγκαθίσταται και εκτελείται και, τελικά, οι παραγόμενες τιμές απόδοσης χρησιμοποιούνται για την κατασκευή ενός Δέντρου Απόφασης που αναπαριστά το τελικό μοντέλο.

### 0.2.1 Θεμελίωση του προβλήματος

Έστω εφαρμογή $A$, η οποία λαμβάνει $n$ εισόδους και παράγει μία έξοδο. Κάθε μια από αυτές τις εισόδους αντιπροσωπεύει μια παράμετρο που μπορεί να επηρεάσει την απόδοσή της και μπορεί να λάβει τιμές από ένα προκαθορισμένο σύνολο τιμών, που θα συμβολίζουμε με $d_i$. Το καρτεσιανό γινόμενο όλων των εισόδων της εφαρμογής $D = d_1 \times d_2 \times \ldots d_n$ συνιστά τον *χώρο παραμέτρων* της εφαρμογής. Κάθε σημείο του χώρου αυτού αναπαριστά ένα μοναδικό συνδυασμό παραμέτρων με τον οποίο μπορεί να εγκατασταθεί η εφαρμογή. Η έξοδος της είναι μια πραγματική τιμή που είναι ενδεικτική της απόδοσης της εφαρμογής και συμβολίζεται με το γράμμα $P$. Με βάση αυτούς τους ορισμούς μπορούμε να ορίσουμε τη συνάρτηση απόδοσης της $A$ ως μια απεικόνιση $m : D \rightarrow P$. Για την *προσέγγιση* αυτής της συνάρτησης, αρκεί η επιλογή ενός υποσυνόλου $D_s \subseteq D$ για τα μέλη του οποίου η $A$ θα εγκατασταθεί και θα ληφθούν οι αντίστοιχες τιμές απόδοσης των σημείων. Η μοντελοποίηση της απόδοσης σε ολόκληρο το χώρο μπορεί, εν συνεχεία, να γίνει με χρήση στατιστικής ή τεχνικών Μηχανική Μάθησης που βοηθούν στη *γενίκευση* της απόδοσης σε ολόκληρο το χώρο παραμέτρων $D$.

Παρόλο που το συγκεκριμένο πρόβλημα είναι σύνηθες και συναντάται συχνά σε διάφορα προβλήματα μοντελοποίησης, στην περίπτωση της μοντελοποίησης της απόδοσης εφαρμογών Μεγάλων Δεδομένων εμφανίζει προκλήσεις που το κάνουν ιδιαίτερα δύσκολο. Οι σύγχρονες εφαρμογές Μεγάλων Δεδομένων επηρεάζονται από ένα μεγάλο αριθμό παραμέτρων που έχει ως συνέπεια την εκθετική αύξηση του χώρου παραμέτρων. Αυτό σημαίνει πως για την κατασκευή μοντέλων απόδοσης υψηλής ακρίβειας, απαιτείται ένας ολοένα αυξανόμενος αριθμός δειγμάτων. Παράλληλα, αν συνυπολογίσουμε πως η εγκατάσταση της εφαρμογής είναι μια χρονοβόρα και οικονομικά ακριβή διαδικασία, καθίσταται σαφές ότι η μεθοδολογία μοντελοποίησης πρέπει να διαχειρίζεται πολύ σοφά το διαθέσιμο αριθμό δειγμάτων που μπορεί να διαθέσει. Το πρόβλημα, λοιπόν, το οποίο αντιμετωπίζουμε μπορεί να εκφραστεί ως εξής: *Έστω εφαρμογή $A$ με χώρο παραμέτρων $D$, συνάρτηση απόδοσης $m : D \to P$ και μια θετική σταθερά $B$. Αναζητούμε ένα σύνολο $D_s$ μεγέθους $B$ το οποίο κατασκευάζει μια προσέγγιση της συνάρτησης απόδοσης $m' : D_s \to P$ με ελάχιστο σφάλμα, ή ισοδύναμα $|m - m'| \to 0$.*

### 0.2.2 Μεθοδολογία

Σύμφωνα με την προηγούμενη περιγραφή του προβλήματος, είναι σαφές ότι η εύρεση ενός *βέλτιστου* συνόλου $D_s$ το οποίο ελαχιστοποιεί το σφάλμα της μοντελοποίησης είναι ένα συνδυαστικό πρόβλημα το οποίο δεν μπορεί να λυθεί σε πολυωνυμικό χρόνο. Παρόλα αυτά, εκμεταλλευόμενοι τις ιδιαιτερότητες που παρουσιάζονται κατά τη μοντελοποίηση συναρτήσεων απόδοσης τελεστών Μεγάλων Δεδομένων, μπορούμε να προτείνουμε μια ευριστική λύση η οποία, όπως θα δούμε, παρουσιάζει αρκετά καλή απόδοση και τρέχει σε πολυωνυμικό χρόνο. Πιο συγκεκριμένα, μπορούμε να κάνουμε τις εξής παρατηρήσεις. Πρώτον, οι σύγχρονοι τελεστές Μεγάλων Δεδομένων είναι σχεδιασμένοι να τρέχουν σε κατανεμημένα περιβάλλοντα στα οποία ο υπολογιστικός φόρτος μπορεί να κατανέμεται σε περισσότερες διεργασίες. Η κλιμάκωση αυτή συχνά μπορεί να περιγραφεί με μια γραμμική ή τμηματικά γραμμική σχέση: Αναμένουμε ότι αν ένας τελεστής τρέξει σε διπλάσιο αριθμό μηχανημάτων θα χρειαστεί περίπου το μισό χρόνο εκτέλεσης. Η παρατήρηση αυτή θα αναφέρεται στο εξής ως *γραμμικότητα.* Δεύτερον, οφείλουμε να παρατηρήσουμε ότι η απόδοση ενός τελεστή αναμένεται να λαμβάνει παρόμοιες τιμές σε γειτονικές περιοχές του χώρου παραμέτρων. Αυτό σημαίνει πως αν επιλέξουμε δυο γειτονικά σημεία του χώρου παραμέτρων, η απόδοση που θα παρατηρήσουμε αναμένεται να είναι αρκετά παρόμοια. Η παρατήρηση αυτή θα αναφέρεται στο εξής ως *τοπικότητα.*

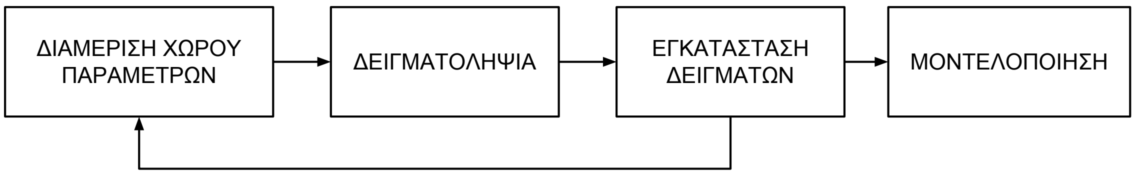


Figure 1: Επισκόπηση μεθόδου προσαρμοστικής μοντελοποίησης

Με βάση αυτές τις δυο σημαντικές παρατηρήσεις, μπορούμε τώρα να περιγράψουμε τη μεθοδολογία προσαρμοστικής μοντελοποίησης ενός τελεστή Μεγάλου Δεδομένων. Η Εικόνα 1 παρουσιάζει μια επισκόπηση της προτεινόμενης επαναληπτικής μεθοδολογίας. Αρχικά, στηριζόμενοι στην ιδιότητα της τοπικότητας, προχωράμε σε μια διαμέριση του χώρου παραμέτρων, στην αρχή κάθε επανάληψης του αλγορίθμου. Η διαμέριση αυτή έχει ως σκοπό την ομαδοποίηση περιοχών του χώρου στις οποίες η απόδοση φαίνεται να ακολουθεί γραμμική συμπεριφορά. Στη συνέχεια, λαμβάνονται δείγματα από κάθε χωρίο που έχει προκύψει στο προηγούμενο βήμα. Ο αριθμός των δειγμάτων που λαμβάνεται από κάθε χωρίο καθορίζεται με βάση το μέγεθος του χωρίου και το σφάλμα μοντελοποίησης της συνάρτησης απόδοσης. Στη συνέχεια, τα δείγματα αυτά εγκαθίστανται και η διαδικασία επαναλαμβάνεται για ένα προκαθορισμένο αριθμό φορών. Τελικά, η συνάρτηση απόδοσης μοντελοποιείται σε όλο το χώρο χρησιμοποιώντας ένα Λοξό Δέντρο Απόφασης.

Ας εξετάσουμε τώρα κάθε φάση του αλγορίθμου με μεγαλύτερη λεπτομέρεια:

**Διαμέριση χώρου παραμέτρων** Στόχος της συγκεκριμένης φάσης είναι η διαμέριση του χώρου παραμέτρων σε ξένα (μεταξύ τους) χωρία, έτσι ώστε τα δείγματα που ανήκουν σε κάθε χωρίο να κατασκευάζουν γραμμικά μοντέλα με όσο το δυνατό μικρότερο σφάλμα. Η διαδικασία διαμέρισης ομοιάζει με τη διαδικασία κατασκευής ενός Δέντρου Απόφασης: Κάθε χωρίο αντιστοιχεί σε ένα φύλλο το δέντρου. Κατά τη διαδικασία της διαμέρισης, το χωρίο (φύλλο) μετατρέπεται σε ένα σύνολο δύο χωρίων (ενδιάμεσος κόμβος Δέντρου Απόφασης) τα οποία χωρίζονται με βάση μια ευθεία διαμέρισης. Η επιλογή αυτής της ευθείας παίζει καταλυτικό ρόλο στην κατασκευή του δέντρου: Θέλουμε η ευθεία διαμέρισης να παράγει δυο χωρία τα δείγματα των οποίων ακολουθούν όσο το δυνατό πιο γραμμική συμπεριφορά, ή ισοδύναμα, να παράγουν γραμμικά μοντέλα με όσο το δυνατό μικρότερο σφάλμα μοντελοποίησης.

Στα παραδοσιακά Δέντρα Απόφασης, οι ευθείες διαμέρισης είναι παράλληλες σε έναν από τους άξονες του χώρου εισόδου. Κατά τη μελέτη μας είδαμε ότι αυτό είναι αρκετά δεσμευτικό και μειώνει την εκφραστικότητα του Δέντρου. Για το σκοπό αυτό, υιοθετήσαμε την τεχνική που συναντάται στα Λοξά Δέντρα Αποφάσεων, στα οποία δοκιμάζονται διαφορετικές ευθείες διαμέρισης που εμπεριέχουν περισσότερες από μια διαστάσεις του χώρου παραμέτρων. Η κατασκευή των υποψήφιων ευθειών διαμέρισης για κάθε φύλλο γίνεται με τη βοήθεια κάποιου αλγορίθμου βελτιστοποίησης (στην παρούσα περίπτωση του αλγορίθμου Προσομοιωμένης Ανόπτησης) όπου εγγυάται τη σύγκλιση σε μια λύση που βρίσκεται αρκετά κοντά στη βέλτιστη. Τελικά, η διαδικασία αυτή επαναλαμβάνεται για όλα τα χωρία που έχουν παραχθεί μέχρι στιγμής, κάτι που έχει ως αποτέλεσμα την επέκταση του δέντρου κατά ένα επίπεδο σε κάθε επανάληψη.

**Δειγματοληψία** Στο σημείο αυτό, ο χώρος έχει διαμεριστεί και πρέπει να επιλεγεί ένα σύνολο δειγμάτων προς εγκατάσταση από κάθε χωρίο. Σε κάθε επανάληψη, πρέπει να επιλεγούν $b$ δείγματα. Η ερώτηση που πρέπει να απαντηθεί σε αυτό το σημείο είναι η ακόλουθη: *Πόσα δείγματα πρέπει να επιλεγούν από κάθε χωρίο;* Δεδομένου ότι τα δείγματα αντιπροσωπεύουν τη γνώση που λαμβάνουμε για τη συνάρτηση απόδοσης, είναι σαφές ότι η λήψη περισσότερων δειγμάτων σε μια περιοχή μας δίνει περισσότερη πληροφορία για την συμπεριφορά της συνάρτησης απόδοσης στην περιοχή αυτή, εις βάρος όμως άλλων περιοχών. Γενικά, θέλουμε η πολιτική δειγματοληψίας να έχει τα ακόλουθα στοιχεία: Πρώτον, να μεγιστοποιεί την κάλυψη του χώρου παραμέτρων έτσι ώστε να μην υπάρχουν ανεξερεύνητες περιοχές και δεύτερον, να εμβαθύνει σε περιοχές που το σφάλμα μοντελοποίησης είναι υψηλό και, συνεπώς, απαιτείται περισσότερη πληροφορία.

Για την πραγματοποίηση αυτής της πολιτικής, δημιουργούμε μια συνάρτηση η οποία αποδίδει βάρη σε κάθε χωρίο (φύλλο το Δέντρου Απόφασης) σε κάθε επανάληψη του αλγορίθμου και, στη συνέχεια, κατανέμει τα $b$ δείγματα με βάση το βάρος κάθε χωρίου. Η συνάρτηση αυτή είναι ευθέως ανάλογη του μεγέθους του χωρίου, με τη λογική ότι μεγάλα χωρία καλύπτουν μεγάλες περιοχές του χώρου παραμέτρων και ένας υψηλός αριθμός δειγμάτων βοηθά την εξερεύνησή του, και αντιστρόφως ανάλογη του σφάλματος μοντελοποίησης, με τη λογική ότι μικρό σφάλμα είναι ενδεικτικό της γραμμικής συμπεριφοράς της απόδοσης και, συνεπώς, δεν χρειάζεται να εξερευνηθεί περαιτέρω το χωρίο. Μετά την κατανομή των $b$ δειγμάτων με βάση το βάρος κάθε χωρίου, εκτελείται δειγματοληψία σε κάθε ένα από αυτά, όπου τα δείγματα επιλέγονται με βάση μια ομοιόμορφη κατανομή. Για παράδειγμα, αν τα χωρία $i$ και $j$ έχουν βάρη $w_i$, $w_j$ αντίστοιχα, τότε το χωρίο $i$ θα λάβει $\frac{w_i}{w_i+w_j} \cdot b$ δείγματα ενώ το χωρίο $j$ θα λάβει $\frac{w_j}{w_i+w_j} \cdot b$ δείγματα.

**Εγκατάσταση δειγμάτων** Μετά την επιλογή των δειγμάτων σε κάθε επανάληψη, η εφαρμογή εγκαθίσταται για τους επιλεχθέντες συνδυασμούς παραμέτρων και λαμβάνονται οι τιμές της απόδοσης της εφαρμογής. Το βήμα αυτό είναι, κατά κανόνα, και το πιο χρονοβόρο: Η εγκατάσταση μιας εφαρμογής σε μια cloud υποδομή είναι μια διαδικασία η οποία απαιτεί πολύ χρόνο, καθώς προϋποθέτει τη δημιουργία εικονικών πόρων (Εικονικών Μηχανών, δίσκων, κλπ.), την ενορχήστρωση της διαδικασίας εγκατάστασης, την εκτέλεση της εφαρμογής και τη συλλογή των αποτελεσμάτων. Κατά την ολοκλήρωση της εκτέλεσης της εφαρμογής για κάθε ένα από τα επιλεχθέντα δείγματα, οι τιμές της απόδοσής της συλλέγονται από τον αλγόριθμο μοντελοποίησης και χρησιμοποιούνται είτε για την εκ νέου διαμέριση και δειγματοληψία του χώρου (αν υπάρχουν και άλλες επαναλήψεις) είτε για την τελική μοντελοποίηση της απόδοσης.

**Μοντελοποίηση** Τέλος, μετά την επανάληψη των τριών πρώτων βημάτων για ένα προκαθορισμένο αριθμό φορών, τα δείγματα που έχουν εγκατασταθεί κατά την εκτέλεση του αλγορίθμου χρησιμοποιούνται για την κατασκευή ενός Λοξού Δέντρου Απόφασης, το όποιο έχει ως φύλλα γραμμικά μοντέλα που μοντελοποιούν την απόδοση της εφαρμογής για τα χωρία όπου ορίζονται. Το Δέντρο αυτό είναι και το τελικό αποτέλεσμα του αλγορίθμου.

### 0.2.3 Παρατηρήσεις

Στο σημείο αυτό, ας μας επιτραπεί να κάνουμε μερικές παρατηρήσεις σχετικά με τη μεθοδολογία που περιγράψαμε παραπάνω.

Αρχικά, το αποτέλεσμα της μεθοδολογίας που περιγράψαμε είναι ένα Λοξό Δέντρο Απόφασης το οποίο αντιπροσωπεύει και το τελικό μοντέλο απόδοσης. Το μοντέλο αυτό μπορεί να χρησιμοποιηθεί με πολλούς διαφορετικούς τρόπους: Μπορεί να βοηθήσει σχετικά με την εύρεση των συνδυασμών παραμέτρων με την καλύτερη απόδοση, ή των παραμέτρων εκείνων που έχουν το μικρότερο αντίκτυπο στην απόδοση ενώ, παράλληλα, μπορεί να βοηθήσει και στην εύρεση των σημείων εκείνων που η απόδοση λαμβάνει τις χειρότερες τιμές της, δείχνοντας κακή κλιμάκωση. Παράλληλα, ακόμα και μια απλή μελέτη της δομής του Δέντρου μπορεί να δώσει πολύτιμες πληροφορίες σχετικά με τη συσχέτιση των διαφορετικών παραμέτρων και τον αντίκτυπό τους στην απόδοση. Οι ευθείες διαμέρισης που χωρίζουν το δέντρο στα υψηλότερα επίπεδα, τείνουν να έχουν μεγαλύτερη σημασία στην απόδοση της εφαρμογής συνολικά. Επομένως, μια παράμετρος που λαμβάνει μεγάλο βάρος σε μια ευθεία διαμέρισης που βρίσκεται ψηλά στη δεντρική δομή, σημαίνει ότι είναι σημαντικότερη σε σχέση με μια τιμή που βρίσκεται χαμηλότερα. Υπό αυτό το πρίσμα, μπορούμε επίσης να παρατηρήσουμε ότι παράμετροι με μικρή σημασία αγνοούνται από το Δέντρο, αφού δεν βοηθούν στην καλύτερη διαμέριση του χώρου. Αυτό επιτρέπει στη μεθοδολογία μας να λειτουργεί με μεγάλο αριθμό παραμέτρων και να μπορεί να εμβαθύνει μόνο σε αυτές που έχουν μεγαλύτερο αντίκτυπο στην απόδοση.

Δεύτερον, όπως συζητήσαμε και προηγούμενα, η εξαγωγή ενός βάρους κατά τη φάση της δειγματοληψίας για κάθε χωρίο του χώρου παραμέτρων, έχει ως σκοπό να συμπεριλάβει δυο αντίρροπες τάσεις: Αυτή της *εξερεύνησης* του χώρου παραμέτρων και αυτή της *εκμετάλλευσης* της γνώσης που έχει αποκτηθεί και την περαιτέρω εμβάθυνση σε περιοχές που υπάρχει έλλειψη γνώσης για την απόδοση της εφαρμογής. Η βαρύτητα κάθε μιας από αυτές τις τάσεις καθορίζεται από το χρήστη καθώς ανάλογα με τον τύπο και το είδος της συνάρτησης απόδοσης, μπορεί οι ανάγκες για εξερεύνηση και εκμετάλλευση να αλλάζουν. Επίσης, η χρήση μιας μαθηματικής έκφρασης για τον καθορισμό του βάρους κάθε χωρίου μας δίνει τη δυνατότητα να συνυπολογίσουμε και άλλες παραμέτρους που τυχόν ενδιαφέρουν το χρήστη. Για παράδειγμα, σε περίπτωση που μας ενδιαφέρει η ελαχιστοποίηση του κόστους της μοντελοποίησης, θα μπορούσαμε να εισάγουμε μια ακόμα ποσότητα στη μαθηματική έκφραση του βάρους του χωρίου που το συνδέει με το κόστος της εγκατάστασης με αντιστρόφως ανάλογο τρόπο (δηλαδή όσο πιο ακριβό είναι το κόστος εγκατάστασης ενός χωρίου τόσο μικρότερο να είναι το βάρος). Με τον τρόπο αυτό, μπορεί κάποιος να συμπεριλάβει και άλλες παραμέτρους στη μοντελοποίηση χωρίς να χρειάζεται να αλλάξει κάτι στην μεθοδολογία.

Τέλος, μια παραδοχή που κάναμε κατά τη θεμελίωση του προβλήματος είναι ότι η απόδοση μιας εφαρμογής μπορεί να περιγραφεί σαν μια μαθηματική συνάρτηση. Αυτό σημαίνει ότι κάθε φορά που η εφαρμογή εγκαθίσταται για ένα συγκεκριμένο συνδυασμό παραμέτρων, η ληφθείσα τιμή της απόδοσης θα είναι πάντα ίδια. Παρόλα αυτά γνωρίζουμε ότι σε ένα περιβάλλον cloud υπάρχουν πολλοί λόγοι για τους οποίους αυτό μπορεί να μην ισχύει: Ο διαμοιρασμός πόρων μεταξύ διαφορετικών χρηστών, οι εικονικοποιημένοι πόροι που προσθέτουν επιπλέον πολυπλοκότητα καθώς και τυχαία ή χρονικά φαινόμενα μπορούν να προσθέσουν θόρυβο στις μετρήσεις απόδοσης και να μειώσουν την επαναληψιμότητα των εγκαταστάσεων. Αυτή είναι μια διάσταση στο πρόβλημα η οποία βρίσκεται εκτός των στόχων της παρούσας διατριβής.

## 0.3 Βασισμένη στο Περιεχόμενο Μοντελοποίηση Τελεστών Ανάλυσης Δεδομένων

Στη σύγχρονη εποχή όπου ολοένα και περισσότερες πηγές δεδομένων είναι ελεύθερα διαθέσιμες προς χρήση, το πρόβλημα επιλογής των *κατάλληλων δεδομένων* για έναν τελεστή γίνεται σημαντικότερο από ποτέ. Η συνήθης τακτική που ακολουθείται από αναλυτές και μηχανικούς δεδομένων κατά τη σχεδίαση τελεστών είναι η εξαντλητική εκτέλεση ενός τελεστή για το σύνολο των διαθέσιμων δεδομένων και, στη συνέχεια, η επιλογή των δεδομένων που συγκεντρώνουν ορισμένα επιθυμητά χαρακτηριστικά για το συγκεκριμένο τελεστή. Λαμβάνοντας υπόψη ότι τόσο το πλήθος των διαθέσιμων πηγών δεδομένων όσο και ο αριθμός των τελεστών που χρησιμοποιούνται καθημερινά συνεχώς αυξάνει, γίνεται σαφές ότι η εξαντλητική εκτέλεση των τελεστών επί όλων των δεδομένων είναι ασύμφορη. Για το λόγο αυτό, στη διατριβή αυτή εξετάζουμε το πρόβλημα της μοντελοποίησης της συμπεριφοράς ενός τελεστή, όπως αυτή εκφράζεται από την έξοδο που αυτός παράγει, όταν εκτελείται για διαφορετικά σύνολα δεδομένων. Στην ενότητα αυτή διατυπώνουμε το πρόβλημα το οποίο επιλύουμε, εισάγουμε τη μεθοδολογία μοντελοποίησης και κάνουμε ορισμένες παρατηρήσεις σχετικά με τη λειτουργία της.

### 0.3.1 Θεμελίωση του προβλήματος

Έστω τελεστής $F$ και ένα σύνολο από σύνολα δεδομένων $D = \{D_1, D_2, \dots, D_n\}$. Τα διαφορετικά σύνολα δεδομένων αποτελούνται από πλειάδες οι οποίες έχουν το ίδιο σχήμα (αριθμό διαστάσεων και τύπο τιμών σε κάθε διάσταση). Θεωρούμε ότι ο τελεστής λαμβάνει ως είσοδο ένα σύνολο δεδομένων $D_i \in D$ και παράγει μια πραγματική τιμή, δηλαδή $F : D \to R$. Το πρόβλημα το οποίο καλούμαστε να επιλύσουμε είναι το εξής: *Δοθέντος ενός τελεστή $F$ και ενός συνόλου $D$, θέλουμε να προσεγγίσουμε την έξοδο του $F$ για κάθε $D_i \in D$.* Η προφανής απάντηση στο πρόβλημα αυτό είναι να εκτελεστεί ο $F$ για όλα τα σύνολα δεδομένων. Παρόλα αυτά, λαμβάνοντας υπόψη ότι ο αριθμός των δεδομένων $n$ μπορεί να είναι πολύ μεγάλος και ότι η υπολογιστική πολυπλοκότητα του $F$ μπορεί να είναι επίσης μεγάλη, καθίσταται σαφές ότι η εξαντλητική εκτέλεση του τελεστή είναι ακριβή και ως προς το χρόνο αλλά και ως προς το κόστος εκτέλεσης. Για το λόγο αυτό, θέλουμε να αντιμετωπίσουμε το πρόβλημα αυτό σαν ένα ακόμη προσεγγιστικό πρόβλημα: Θέλουμε να εκτελέσουμε τον τελεστή για ένα μικρό πλήθος των διαθέσιμων συνόλων δεδομένων, να εξάγουμε την τιμή εξόδου του τελεστή και να γενικεύσουμε για όλα τα σύνολα δεδομένων.

Σε αντίθεση με πριν όμως, το πρόβλημα που καλούμαστε να επιλύσουμε δεν μπορεί να λυθεί με κάποια γνωστή τεχνική προσέγγισης. Ο λόγος που συμβαίνει αυτό είναι ότι το σύνολο εισόδου της συνάρτησης που καλούμαστε να προσεγγίσουμε (δηλαδή το $D$) δεν αποτελεί μετρικό χώρο, αφού δεν είναι γνωστή καμία σχέση μεταξύ των $D_1, D_2, ..., D_n$. Υπενθυμίζουμε σε αυτό το σημείο ότι όλες οι τεχνικές στατιστικής προσέγγισης (συμπεριλαμβανομένων των τεχνικών Μηχανικής Μάθησης) προϋποθέτουν την υπάρξει μια σχέσης "απόστασης" για τα σημεία εισόδου της συνάρτησης που είναι υπό προσέγγιση. Στην προηγούμενη ενότητα, είδαμε ότι η υπό προσέγγιση συνάρτηση απόδοσης είχε σαν σύνολο εισόδου το χώρο παραμέτρων, τα σημεία του οποίου μπορούσαν να συγκριθούν. Αντίθετα, στο πρόβλημα αυτό τα σημεία $D_i$ ανήκουν σε ένα αταξινόμητο σύνολο, για το οποίο δεν υπάρχει άλλη πληροφορία.

### 0.3.2 Μεθοδολογία

Από τη διατύπωση του προβλήματος που προηγήθηκε, είναι σαφές ότι το πρώτο βήμα για να μετατρέψουμε το πρόβλημα σε μια μορφή που μπορεί να επιλυθεί με κλασσικές τεχνικές στατιστικής προσέγγισης, είναι να κατασκευάσουμε ένα μετρικό χώρο για τα δεδομένα εισόδου, ή ισοδύναμα, να ορίσουμε μια συνάρτηση "απόστασης" μεταξύ τους. Για το σκοπό αυτό, μπορούμε να κάνουμε μια καίρια παρατήρηση: Παρόμοια σύνολα δεδομένων τείνουν να επηρεάζουν τους τελεστές που εκτελούνται επάνω σε αυτά με παρόμοιο τρόπο. Για παράδειγμα, ας θεωρήσουμε έναν απλό αριθμητικό τελεστή που εκτελεί αριθμητικές πράξεις μεταξύ των πλειάδων ενός συνόλου δεδομένων και τρία σύνολα δεδομένων που έχουν προκύψει με τον εξής τρόπο: τα σύνολα $A$ και $B$ έχουν προκύψει από τυχαία κατανομές και το σύνολο $C$ είναι πιστό αντίγραφο του $A$, στο οποίο έχει προστεθεί θόρυβος ίσος με το 1% των τιμών των αρχικών πλειάδων. Σε αυτό το παράδειγμα, αναμένουμε οι τιμές του τελεστή όταν αυτός εκτελείται για τα σύνολα $A$ και $C$ να είναι παρόμοιες, ενώ αντίθετα η τιμή για το $B$ είναι απροσδιόριστη. Από αυτό το παράδειγμα, προκύπτει ότι η ομοιότητα μεταξύ συνόλων δεδομένων (όπως του

$A$ και του $C$), μπορεί να οδηγήσει σε ομοιότητα τιμών του τελεστή. Πως όμως μπορούμε να αξιολογήσουμε πόσο όμοια είναι τα σύνολα δεδομένων όταν δεν έχουμε καμία γνώση για αυτά;

Για την ποσοτικοποίηση της ομοιότητας μεταξύ συνόλων δεδομένων, μπορούμε να ακολουθήσουμε διαφορετικές προσεγγίσεις. Παρόλα αυτά, στη διατριβή αυτή εξετάζουμε την ομοιότητα υπό το πρίσμα τριών θεμελιωδών ιδιοτήτων: (α) της *στατιστικής κατανομής* τους, (β) της *σειράς διάταξης* των πλειάδων τους και (γ) του *μεγέθους* τους. Η ομοιότητα της κατανομής δυο συνόλων δεδομένων έχει ως σκοπό να εξετάσει το βαθμό στον οποίο οι πλειάδες των δυο συνόλων επικαλύπτονται, δηλαδή απαντώνται σε παρόμοιες περιοχές του χώρου των πλειάδων. Στο προηγούμενο παράδειγμα, οι πλειάδες των συνόλων δεδομένων $A$ και $C$ εμφάνισαν πολύ μεγάλη επικάλυψη, επειδή ακριβώς ο προστιθέμενος θόρυβος ήταν πολύ μικρός. Η ομοιότητα της σειρά διάταξης (εφόσον ορίζεται) εξετάζει το κατά πόσο δυο σύνολα δεδομένων ισαπέχουν εξίσου από ένα πλήρως διατεταγμένο σύνολο δεδομένων. Τέλος, η εξέταση του μεγέθους των συνόλων εξετάζει το κατά πόσο δυο σύνολα δεδομένων έχουν παρόμοιο μέγεθος. Οι τρεις αυτές ιδιότητες δεν είναι μοναδικές, καθώς θα μπορούσε κανείς να εξετάσει τα σύνολα δεδομένων και από διαφορετικές σκοπιές. Παρόλα αυτά, ισχυριζόμαστε πως οι ιδιότητες αυτές είναι θεμελιώδεις και εξετάζονται πολύ συχνά κατά την ανάλυση δεδομένων επειδή επηρεάζουν ένα πολύ μεγάλο φάσμα τελεστών.

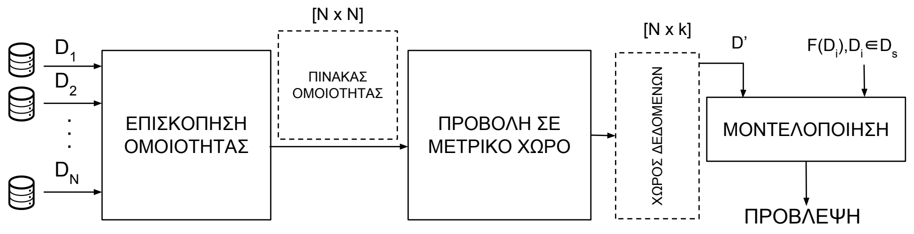


Figure 2: Επισκόπηση μεθοδολογίας μοντελοποίησης της εξόδου ενός τελεστή

Με βάση αυτές τις παρατηρήσεις, περιγράφουμε τώρα τη μεθοδολογία μοντελοποίησης της εξόδου ενός τελεστή. Στην Εικόνα 2 παραθέτουμε μια επισκόπηση της προτεινόμενης μεθοδολογίας. Σε πρώτη φάση, τα σύνολα δεδομένων συγκρίνονται μεταξύ τους ως προς τα χαρακτηριστικά που αναφέρθηκαν προηγουμένως και κατασκευάζεται ένας πίνακας ομοιότητας. Κάθε κελί $[i, j]$ του πίνακα περιέχει μια τιμή στο διάστημα $[0-1]$ που δείχνει την ομοιότητα των συνόλων $i$ και $j$ (το 0 δηλώνει πλήρη ανομοιότητα και το 1 υποδηλώνει πλήρη ταύτιση). Στη συνέχεια, ο πίνακας μετατρέπεται σε ένα μετρικό χώρο χαμηλής διάστασης, στον οποίο απεικονίζονται όλα τα σύνολα δεδομένων και οι αποστάσεις τους είναι αντιστρόφως ανάλογες των ομοιοτήτων που περιέχονται στον πίνακα. Τέλος, ο υπό εξέταση τελεστής εκτελείται για ένα μικρό πλήθος από τα διαθέσιμα σύνολα και οι παραγόμενες τιμές χρησιμοποιούνται από ένα Νευρωνικό Δίκτυο για την προσέγγιση των τιμών σε όλο το μετρικό χώρο και, συνεπώς, σε όλα τα σύνολα δεδομένων. Ας περιγράψουμε τώρα τη μεθοδολογία με μεγαλύτερη λεπτομέρεια:

**Εκτίμηση Ομοιότητας** Σκοπός αυτού του τμήματος της μεθοδολογίας είναι η ποσοτικοποίηση της ομοιότητας μεταξύ όλων των συνόλων δεδομένων και η δημιουργία ενός πίνακα ομοιότητας που περιέχει αυτή την πληροφορία. Όπως είπαμε προηγούμενα, στην διατριβή αυτή εξετάζουμε την ομοιότητα από τρεις σκοπιές: (α) τη στατιστική κατανομή, (β) τη σειρά διάταξης και (γ) το μέγεθος. Ανεξαρτήτως της εξεταζόμενης ιδιότητας, θεωρούμε ότι για την ομοιότητα δυο συνόλων δεδομένων ισχύει $S(A, B) = S(B, A)$ και ότι $0 \leq S(A, B) \leq 1$. Ας δούμε τώρα πως γίνεται ο υπολογισμός για κάθε ιδιότητα ξεχωριστά.

**Κατανομή** Η ομοιότητα στην κατανομή δυο συνόλων δεδομένων ποσοτικοποιεί το βαθμό στον οποίο τα δύο σύνολα εμφανίζουν πλειάδες στις ίδιες ή σε κοντινές περιοχές του χώρου τους. Για την ποσοτικοποίηση αυτή, χρησιμοποιήσαμε μια κανονικοποιημένη εκδοχή του συντελεστή Bhattacharyya [CRM00]:

$$Distribution(A, B) = \frac{\sum_{i=1}^{l} \sqrt{A_i B_i}}{\sqrt{|A||B|}}$$

ο οποίος, με βάση μια διαμέριση του χώρου των πλειάδων σε $l$ τμήματα, υπολογίζεται με τον παραπάνω τύπο. Ο αριθμός $A_i$ συμβολίζει τον αριθμό των πλειάδων που βρίσκονται στο χωρίο $i$, ενώ το $|A|$ συμβολίζει το μέγεθος του συνόλου $A$. Για τον υπολογισμό ενός "δίκαιου" σχήματος διαμέρισης που χωρίζει τις πλειάδες με δίκαιο τρόπο, επιλέξαμε την διαμέριση με τη βοήθεια του αλγορίθμου k-means++ [AV07], που εκτελείται σε ένα σύνολο δεδομένων που περιέχει ένα μικρό δείγμα των πλειάδων όλων των συνόλων.

**Σειρά Διάταξης** Η σειρά διάταξης υπολογίζεται με μια παραλλαγή του Kendall $\tau$ [Ken48]:

$$Order(A, B) = \frac{concord(a, b) - discord(a, b)}{n(n-1)} + \frac{1}{2}$$

όπου $a$ και $b$ αντιπροσωπεύουν τους πίνακες που περιέχουν τις σχετικές θέσεις των στοιχείων των συνόλων $A$ και $B$ αντίστοιχα, η σχέση $concord(a, b)$ επιστρέφει τον αριθμό των ζευγαριών $(a_i, b_i)$ και $(a_j, b_j)$, $i \neq j$ για τα οποία ισχύει ότι αν $a_i > a_j$ τότε $b_i > b_j$ ή αν $a_i < a_j$ τότε $b_i < b_j$ και η σχέση $discord(a, b)$ είναι η άρνηση της σχέσης $concord$.

**Μέγεθος** Η σχέση του εκφράζεται με την ακόλουθη σχέση:

$$Size(A, B) = \frac{\min(|A|, |B|}{\max(|A|, |B|)}$$

**Προβολή σε μετρικό χώρο** Ο πίνακας ομοιότητας που παράχθηκε στο προηγούμενο βήμα παρέχει πολύτιμες πληροφορίες σχετικά με τη σχέση των συνόλων δεδομένων. Παρόλα αυτά, ο τρόπος απεικόνισης αυτής της πληροφορίας είναι προβληματικός για τους ακόλουθους λόγους: (α) ο πίνακας μεγαλώνει πολύ γρήγορα με τον αριθμό των δεδομένων, (β) η γραφική απεικόνιση του πίνακα μπορεί να γίνει με heatmaps που δεν επιτρέπουν την απεικόνιση της πληροφορίας σε κανονική κλίμακα και (γ) τα περισσότερα μοντέλα Μηχανικής Μάθησης προϋποθέτουν την ύπαρξη κανονικών συντεταγμένων για τα σημεία εισόδου και όχι σχέσεων ομοιότητας/απόστασης μεταξύ ζευγών αυτών. Για το λόγο αυτό, καλούμαστε να απεικονίσουμε την πληροφορία που περιέχεται στον πίνακα ομοιότητας σε ένα μετρικό χώρο (ιδανικά χαμηλής διάστασης), ο οποίος απεικονίζει κάθε σύνολο δεδομένων σε ένα σημείο. Η απόσταση μεταξύ δυο σημείων είναι αντιστρόφως ανάλογη της ομοιότητας των αντίστοιχων συνόλων: παρόμοια σημεία θα βρίσκονται κοντά, ενώ ανόμοια σημεία θα έχουν μεγάλη απόσταση.

Για την επίλυση αυτού του προβλήματος, χρησιμοποιήσαμε δυο ευρέως γνωστές τεχνικές που εφαρμόζονται διαδοχικά. Αρχικά, το πρόβλημα της απεικόνισης μπορεί να προσεγγιστεί σαν πρόβλημα βελτιστοποίησης που μπορεί να λυθεί με τη βοήθεια της ιδιοανάλυσης. Η τεχνική του Multidimensional Scaling [Gow66] κανονικοποιεί τον πίνακα ομοιότητας και απεικονίζει τα σύνολα δεδομένων σε σημεία των οποίων οι διαστάσεις έχουν φθίνουσα σημασία. Στο σημείο αυτό, μπορεί να αποφασιστεί ένας (συνήθως μικρός) αριθμός διαστάσεων που αρκεί για την απεικόνιση με ικανοποιητική ακρίβεια. Από εδώ προκύπτει και ένα πρώτο σύνολο συντεταγμένων των σημείων. Σε δεύτερη φάση, εφαρμόζεται ο μετασχηματισμός Sammon [Sam69] που έχει ως σκοπό την περαιτέρω επανατοποθέτηση των σημείων ώστε να απεικονίζουν την πληροφορία του πίνακα ομοιότητας με μεγαλύτερη ακρίβεια. Το τελικό αποτέλεσμα είναι ένα σύνολο συντεταγμένων για κάθε ένα από τα αρχικά σύνολα δεδομένων.

**Μοντελοποίηση** Τέλος, στη φάση της μοντελοποίησης, ο τελεστής εκτελείται για ένα μικρό πλήθος από τα διαθέσιμα σύνολα δεδομένων και οι τιμές της εξόδου του μαζί με τις συντεταγμένες των αντίστοιχων σημείων δίνονται σαν σύνολο εκπαίδευσης σε ένα Νευρωνικό Δίκτυο. Το Νευρωνικό Δίκτυο μπορεί να προσεγγίσει τις τιμές του τελεστή για όλα τα σύνολα δεδομένων.

### 0.3.3 Παρατηρήσεις

Στο σημείο αυτό, είναι σκόπιμο να γίνουν ορισμένες παρατηρήσεις για τη μεθοδολογία που περιγράψαμε παραπάνω στο σύνολό της. Αρχικά, οφείλουμε να παρατηρήσουμε ότι το υπολογιστικά ακριβό κομμάτι της διαδικασίας, που αφορά την εξαγωγή των ομοιοτήτων και τη δημιουργία του μετρικού χώρου, παραμένει ανεξάρτητο του τελεστή που είναι προς εξέταση. Αυτό σημαίνει πως η εξαγωγή των ομοιοτήτων δεν συσχετίζεται με τον τελεστή και, ως αποτέλεσμα, ο ίδιος μετρικός χώρος μπορεί να χρησιμοποιηθεί για τη μοντελοποίηση διαφορετικών τελεστών χωρίς να χρειάζεται επανάληψη της διαδικασίας από την αρχή. Αυτή η πολύ ενδιαφέρουσα παρατήρηση καταδεικνύει την δυνατότητα επαναχρησιμοποίησης της γνώσης που έχουμε για τα δεδομένα σε άλλους τελεστές. Με τον τρόπο αυτό, η επιτάχυνση που μπορούμε να επιτύχουμε μέσω της μοντελοποίησης κατά τη διαδικασία της ανάλυσης δεδομένων μεγιστοποιείται καθώς το αρχικό κόστος που πληρώνουμε για τον υπολογισμό του μετρικού χώρου αποσβένεται με τη χρήση περισσότερων τελεστών. Αυτός είναι και ο ακρογωνιαίος λίθος της ιδέας πίσω από τη μεθοδολογία: Παρόλο που ο αριθμός των τελεστών που μπορούν να εκτελεσθούν σε κάποια σύνολα δεδομένων είναι πολύ μεγάλος, οι ιδιότητες από τις οποίες αυτοί επηρεάζονται είναι αρκετά λιγότερες και μπορούν εύκολα να μελετηθούν.

Δεύτερον, οι ιδιότητες οι οποίες εξετάστηκαν για την εξαγωγή του πίνακα ομοιότητας προέκυψαν επειδή διαπιστώθηκε πειραματικά (και για κάποιες περιπτώσεις αποδείχθηκε και θεωρητικά) ότι επηρεάζουν σε πολύ μεγάλο βαθμό το αποτέλεσμα διαφορετικών τύπων τελεστών που χρησιμοποιούνται καθημερινά σε διάφορες εργασίες ανάλυσης δεδομένων. Παρόλα αυτά, αναγνωρίζουμε ότι η λίστα των ιδιοτήτων δεν είναι εξαντλητική: Διαφορετικοί τελεστές μπορεί να επηρεάζονται περισσότερο ή λιγότερο από κάποιες ιδιότητες των δεδομένων, αλλά πάντα οι ιδιότητες αυτές θα είναι λιγότερες από το πλήθος των τελεστών και πιο εύκολα μετρήσιμες για την έκφραση της ομοιότητας. Σκοπός της συγκεκριμένης μεθοδολογίας, λοιπόν, είναι η δημιουργία μιας "βιβλιοθήκης" συναρτήσεων ομοιότητας που εξετάζουν δημοφιλείς ιδιότητες και ο συνδυασμός πινάκων ομοιότητας με τρόπο τέτοιο ώστε να προκύψουν πιο σύνθετοι πίνακες που εμπεριέχουν περισσότερη πληροφορία.

Τέλος, μια παραδοχή που κάναμε από την αρχή αυτής της ενότητας είναι ότι τα σύνολα δεδομένων έχουν την μορφή συνόλων που αποτελούνται από πλειάδες ίδιου σχήματος. Παρόλα αυτά, κανένα χαρακτηριστικό της προτεινόμενης μεθοδολογίας δεν τη δεσμεύει να εφαρμοστεί μόνο σε τέτοιου τύπου δεδομένα. Πράγματι, δοκιμάσαμε να μοντελοποιήσουμε δεδομένα που βρίσκονται σε μορφή γράφων στα οποία εκτελούνται αντίστοιχοι τελεστές. Η συνάρτηση ομοιότητας για την περίπτωση αυτή ήταν η ομοιότητα της κατανομής των βαθμών των κόμβων των γράφων, μια ιδιότητα η οποία χαρακτηρίζει μοναδικά τους διαφορετικούς τύπους γράφων. Το ενδιαφέρον αποτέλεσμα από αυτή τη διαδικασία ήταν ότι η μεθοδολογία μας μπόρεσε να μοντελοποιήσει διαφορετικούς τελεστές με πολύ μεγάλη ακρίβεια. Το μόνο σκέλος της προσέγγισης που χρειάστηκε να αλλάξει για την εκτέλεση σε δεδομένα γράφων ήταν η συνάρτηση ομοιότητας. Στη γενική περίπτωση, λοιπόν, αυτή η παρατήρηση είναι ενδεικτική της δυνατότητας γενίκευσης της συγκεκριμένης μεθοδολογίας σε διαφορετικούς τύπους δεδομένων: Εφόσον μπορεί να οριστεί μια σχέση ομοιότητας μεταξύ ενός ζεύγους του συνόλου δεδομένων, η προτεινόμενη μεθοδολογία μπορεί να εφαρμοστεί και να μοντελοποιήσει συμπεριφορά ενός τελεστή που εφαρμόζεται στα υπό εξέταση δεδομένα.

## 0.4 Συμπεράσματα

Σε αυτή τη διατριβή, εξετάσαμε το πρόβλημα της μοντελοποίησης της συμπεριφοράς ενός τελεστή Μεγάλων Δεδομένων από δύο διαφορετικές οπτικές. Πρώτον, αντιμετωπίσαμε το πρόβλημα της μοντελοποίησης της απόδοσής του, για διαφορετικούς συνδυασμούς παραμέτρων που τον επηρεάζουν και, δεύτερον, προτείναμε μια μεθοδολογία βασισμένη στο περιεχόμενο που μοντελοποιεί την έξοδό του όταν αυτός εφαρμόζεται σε διαφορετικά σύνολα δεδομένων. Παρόλο που οι συνεισφορές μας προσεγγίζουν το πρόβλημα από διαφορετικές προοπτικές, ο συνδυασμός τους παρέχει μια ολοκληρωμένη λύση του προβλήματος της μοντελοποίησης της συμπεριφοράς ενός σύγχρονου τελεστή.

Η πρώτη συνεισφορά μας σχετίζεται με τη μοντελοποίηση της απόδοσης ενός τελεστή όταν αυτός εγκαθίσταται με διαφορετικούς συνδυασμούς παραμέτρων. Παρόλο που αυτό είναι ένα πρόβλημα που έχει μελετηθεί ευρέως, οι προτεινόμενες ερευνητικές προσεγγίσεις δεν λαμβάνουν υπόψη την ολοένα αυξανόμενη πολυπλοκότητα της δομής της εφαρμογής, απαιτώντας έναν αυξανόμενο αριθμό δειγμάτων που καθιστούν τη μοντελοποίηση αναποτελεσματική και κοστοβόρα. Για το σκοπό αυτό, η ερευνητική μας προσπάθεια επικεντρώθηκε στην παροχή μιας μεθοδολογίας που μπορούν να οδηγήσουν το μοντέλο να μεγιστοποιήσει την ακρίβειά για ένα δεδομένων μέγιστο αριθμό δειγμάτων.

Η προτεινόμενη ”διαίρει-και-βασίλευε” προσέγγιση αποδείχτηκε ιδιαίτερα αποτελεσματική για αυτό το πρόβλημα. Λαμβάνοντας υπόψη ότι, λόγω του σχεδιασμού τους, τα κατανεμημένα συστήματα τείνουν να έχουν απόδοση που προσεγγίζονται εύκολα με γραμμικές ή τμηματικά γραμμικές συναρτήσεις, η διαμέριση του χώρου σε περισσότερες και μικρότερες περιοχές διευκόλυνε τη μοντελοποίηση και επέτρεψε την εστίαση σε κάθε μία χωριστά.

Επιπλέον, η αποδοτική διανομή των διαθέσιμων δειγμάτων με στόχο τόσο την εξερεύνηση του χώρου όσο και την εκμετάλλευση της παραγόμενης γνώσης μας επέτρεψε να αποφύγουμε την υπερβολική προσήλωση σε περιοχές με ανωμαλίες, χωρίς όμως να μας απαγορεύει την εμβάθυνση για την περαιτέρω διαμέριση με μεγαλύτερη λεπτομέρεια. Τέλος, η υιοθέτηση λοξών δέντρων αποφάσεων αύξησε την εκφραστικότητα του τελικού μοντέλο με το κόστος της απαίτησης περισσότερων υπολογισμών. Αυτή η εκφραστικότητα μεταφράζεται σε κέρδη ακρίβειας της μοντελοποίησης όταν ο αριθμός των διαθέσιμων δειγμάτων είναι μικρός. Ωστόσο, μια ενδιαφέρουσα παρενέργεια της υιοθέτησης των λοξών διαχωριστικών γραμμών είναι ότι μοτίβα όπως είναι οι ασυνέχειες, τα μέγιστα κλπ., που περιλαμβάνουν περισσότερες από μία διαστάσεις εισόδου ταυτόχρονα, αναγνωρίζονται ταχύτερα.

Η δεύτερη συμβολή αυτής της εργασίας σχετίζεται με τη μοντελοποίηση της εξόδου ενός τελεστή όταν αυτός εκτελείται για διαφορετικά σύνολα δεδομένων. Η απουσία οποιασδήποτε σχέσης μεταξύ των συνόλων δεδομένων εισόδου καθιστά το πρόβλημα ιδιαίτερα δύσκολο. Με γνώμονα την παρατήρηση ότι παρόμοια σύνολα δεδομένων επηρεάζουν τους τελεστές με παρόμοιο τρόπο, η έρευνά μας εστιάστηκε στη δημιουργία ενός μετρικού χώρου δεδομένων που αντανακλά τις μεταξύ τους σχέσεις, μετρούμενες υπό το φως τριών θεμελιωδών ιδιοτήτων: τη στατιστική κατανομή, το μέγεθος του συνόλου δεδομένων και την σειρά διάταξης.

Η προτεινόμενη μεθοδολογία κατάφερε να κατασκευάσει μετρικούς χώρους χαμηλής διάστασης που παρέχουν χρήσιμες πληροφορίες που γίνονται αντιληπτές ακόμα και με απλή οπτική εξέτασή του. Η καρδιά της μεθοδολογίας μας βρίσκεται στην επιλογή κατάλληλων συναρτήσεων ομοιότητας. Παρόλο που η πληροφορία που παρέχεται από το μετρικό χώρο είναι ξένη ως προς τον τελεστή, οι ιδιότητες οι οποίες πρέπει να εξεταστούν για την εξαγωγή των ομοιοτήτων μεταξύ των συνόλου δεδομένων πρέπει να επηρεάζουν τον υπό εξέταση τελεστεί για να μπορούν να παράγονται χρήσιμα μοντέλα. Στην εργασία μας απομονώσαμε τρεις ιδιότητες που επηρεάζουν διαφορετικές κατηγορίες τελεστών. Υποστηρίζουμε ότι οι ιδιότητες αυτές είναι πολύ λιγότερες από το σύνολο των πιθανών τελεστών που μπορούν να εφαρμοστούν επάνω στο σύνολο δεδομένων. Συνεπώς απομονώνοντας και συνδυάζοντας ένα μικρό σύνολο από ιδιότητες που επηρεάζουν πολλούς δημοφιλείς τελεστές μπορούμε να επιταχύνουμε κατά πολύ την ανάλυση δεδομένων. Παράλληλα η θεώρηση διαφορετικών τύπων δεδομένων, όπως είναι για παράδειγμα τα δεδομένα γράφων, υπερτονίζει την εφαρμοσιμότητα της μεθοδολογίας μας για διαφορετικά σενάρια χρήσης.

CHAPTER 1

# Introduction

## 1.1 Motivation

The advent of the Big Data era has revolutionized the way the world views and interacts with data. Nowadays, an increasing number of enterprises rely on collecting, handling and analyzing Big Data in order to provide personalized services of higher quality to their customers. The key asset of this practice is that decision making becomes more and more data-driven [Loh12]: Big Data analysis reveals patterns and produces knowledge that leads industries to take better decisions and create products that align with their customers' needs. Simultaneously, the plethora of data sources that constantly generate new data with high rates has given the boost to the academic world to search for more efficient ways to store, manage and analyze data.

Data oriented research fields, such as Data Management, Data Science and Machine Learning have been gaining an increasing amount of attention in the last years. This interest is far from declining: The latest research achievements of Deep Learning [LBH15] that bases its success on even higher data volumes than traditional Machine Learning approaches, the constant increase in IoT [WF15] devices that generate data streams of high velocity and the increasing adoption of alternative data formats (e.g., images, videos) in communication and social networks pose the necessity to keep creating systems and algorithms that are not only able to efficiently manage Big Data, but also analyze them in higher detail and produce meta-knowledge more efficiently.

According to the NIST definition of Big Data [G$^+$15], this term can describe data that have the following key properties (also referred to as the *4 V's*): *Volume*, *Velocity*, *Variety* and *Variability*.

Volume refers to the massive data size that makes any centralized database system inappropriate to cope with it. Velocity refers to the rate with which new data are created. Variety refers to their diverse origins, schemas and representations of different data sources and, finally, variability refers to the rate that existing data are modified. Note that even though different Big Data definitions propose a varying number of characteristics that may reach up to 10 [10v] or even 42 V's [42v], the consensus among the research community is that a system qualifies as a "Big Data" one when at least the aforementioned properties are met. Figure 1.1 depicts the key properties of Big Data.

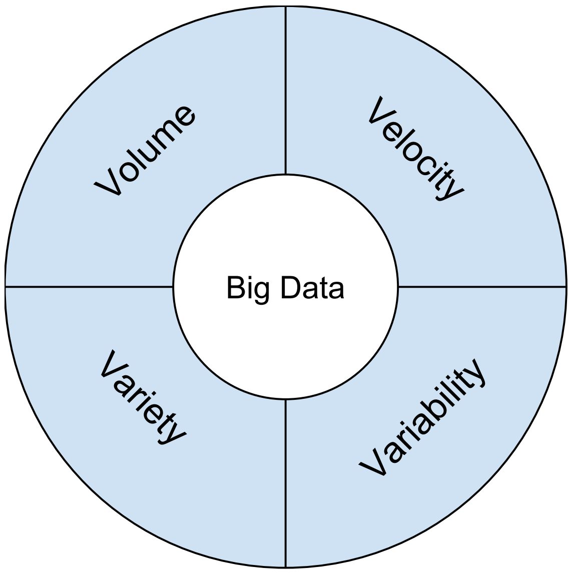


Figure 1.1: The 4 V's of Big Data

The 4 V's render traditional, centralized techniques for managing and analyzing Big Data inappropriate. To this end, Big Data systems usually exploit distributed architectures, in order to extend their scalability limits and cope with increasing data sizes efficiently. Moreover, Big Data systems are commonly deployed over cloud infrastructures, in order to fully exploit its merits: Dynamically allocated resources billed in a pay-as-you-go manner, reduced administrative costs, high availability, practically zero infrastructure administration costs and the ability to elastically scale with load are some of the reasons that explain why the Cloud paradigm [AFGo09] is gaining increasing attention for Big Data applications and is, thus, preferred against on-premise installations.

Nevertheless, the adoption of the Big Data principles did not came with zero cost. Big Data systems and operators, i.e., binaries and executable files that read from one or more datasets and produce outputs of different types, consist of different cooperating software pieces and become increasingly complex in order to achieve their goals. This inevitable complexity propagates to the general behavior of an operator and make it unpredictable and hard to *model.* Even though one can identify multiple dimensions in the term "operator behavior", the fundamental components that best characterize it and, eventually, determine the operator's success are two: Its *performance* as a piece of software and its ability to produce *insightful outputs* that facilitate decision making.

Although in the general case these two aspects are orthogonal, an operator's success relies on both: Slower operators and systems that produce smart results are not more usable than faster systems that produce results of lower quality. In the quest of satisfying both behavior dimensions, *modeling* plays a crucial role. The ability to forecast an operator's behavior given a set of parameters that affect it, is crucial for running it in an efficient and cost effective way and, on the same time, guarantee that its output is of high quality. This is the main motivation behind this dissertation: It presents an approach that aims at capturing both dimensions of an operator's behavior and successfully deal with the underlying complexity without requiring prior knowledge or any insight regarding its functionality.

Focusing on the first dimension, performance modeling is a long discussed problem the origins of which date back to the appearance of the first computer programs. The ability to model an application's performance based on the resources it occupies is crucial for taking multiple decisions throughout its lifecycle. Such a performance model can facilitate deploying the application using the most effective and cost efficient configuration, it can help with identifying performance bottlenecks, it can assist elastic scaling and, in general, support any decision that requires an estimation of the application's performance for a given configuration. The existing approaches can be broadly divided in two categories: The analytical and the "black-box" approaches. The analytical or "white-box" approaches (e.g., [LZZ$^+$10, LZK$^+$11, MWS$^+$07, WCB10]) study the application architecture and describe the way that different components interact with each other using mathematical expressions. The final model represents a function that projects the application's inputs (which are defined by the analyst) to its output. On the contrary, "black-box" approaches (e.g., [DPIK18, MAJ$^+$17, GCMS15, GTPK15]) make no assumption on application structure but, instead, follow a function approximation formulation. Specifically, they deploy the application using different input configurations, they collect the respective performance metrics and use statistical or Machine Learning methodologies to approximate application performance for the entire configuration space.

Although methodologies of both categories have produced satisfying results for certain use cases, their inability to cope with the increasing application complexity of modern Big Data operators either reduce their practicality or accuracy. Specifically, analytical modeling of complex Big Data workflows that comprise many elements is a non-trivial task that requires deep understanding of the architecture of each component. On the other hand, the distributed nature of modern operators exponentially increases the application configuration space and, thus, massively increases the number of configurations that need to be tested in order to obtain a model of acceptable accuracy. The absence of a performance modeling methodology that combines practicality with accuracy when modeling Big Data operators is the main motivation point for the first part of this dissertation.

Considering the quality of the operator's output, the original motivation that lead to the Big Data era was to exploit the massive amounts of data in order to build smarter systems that are able to take better decisions based on them. Nevertheless, massive data volumes do not always lead to smarter systems and algorithms: Noisy and irrelevant data, missing data points and untrustworthy sources are some of the reasons that undermine data *utility* for a given operator. Interestingly so, many researchers have introduced the concept of *Medium data* [tab17, blo14] in order to highlight that it is the data utility rather than the volume of the data per se that benefits modern operators and enable them to drive strategic decisions. This is not only related to data sources themselves, but it is also highly affected by the purpose of the operator.

As an example, take the case of Security Information and Event Management (SIEM) systems [MHH+10], commonly deployed to identify and mitigate cyberattacks. These systems are reported to increasingly fail to identify and stop advanced persistent attacks because of their inability to cope with the increasing amount of available datasets utilized to train them [Pet16]. They lack the analytics capabilities to process a vast amount of data and, hence, their administrator needs to make a key decision: Which datasets out of the available ones should be used in order to train such a system to stop cyberattacks of a certain type in tight time constraints? In other words, which dataset has the *highest utility* for this workflow?

Many research approaches have attempted to extract meta-knowledge out of a given set of datasets, in order to provide hints to a data analyst regarding their content and possible usages. *Data Integration* approaches (e.g., [Len02, HSG+17, MSJ+16]) conduct statistical analysis to the existing datasets (that form a Data Lake) and attempt to extract a set of metadata that characterize their contents. *Data Exploration* methodologies [JGMP16, SCJ16, BGM+17] are mostly focused on identifying a region of the data space that best satisfies a set of given constraints. Note that, neither of these methodologies connect the underlying datasets with the operators. Moreover, even the same operators may be benefited from different data properties under different occasions, e.g., a clustering operator may require datasets of different densities when used for different tasks. For this reason, a methodology that models an operator's outcome when applied to each of the different datasets without exhaustively executing it for each dataset, can highly accelerate data analysis.

Taking into consideration that modern data sources are consumed by multiple operators and that new datasets are constantly born, the gains of avoiding exhaustive operator executions pile up and accelerate data analysis tasks even further. The lack of a methodology with these goals in the literature motivated the design and implementation of the second part of this dissertation. This contribution enables a data analyst to quickly obtain a prediction of an operator's output when applied over different datasets. This can be further utilized for ranking different datasets (e.g., *Give me the top-k datasets with the highest first eigenvalues*), identifying datasets with specific properties (e.g., *Give me all datasets that 10% of their tuples are outliers*), filtering (e.g., *Give me all the datasets where the averages of column $i$ and $j$ are less than $c$*), etc.

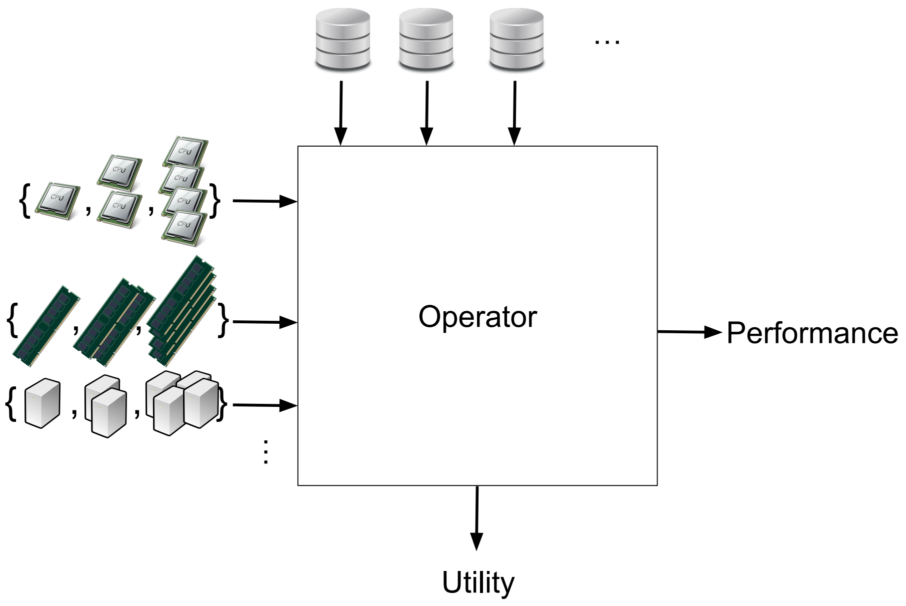


Figure 1.2: Two dimensions of modeling operator behavior

The two challenges that this dissertation addresses represent two sides of the same problem. The success of any modern service equally relies on both of these parameters: It should not only target to perform within certain standards but also produce results of high quality, as a consequence of utilizing the *right data*. Figure 1.2 visualizes how these two problem dimensions are related. On a high level, both modeling problems seek for an expression of either the performance or the operator's outcome in relation to the input resource configurations and datasets respectively.

## 1.2 Contribution

This dissertation models the behavior of an arbitrary, provided Big Data operator from two different perspectives. First, it models its performance, measured using an indicator that quantifies the extent to which the operator fulfills its objectives, in relation to its deployment configuration. Even though the deployment configuration is a relatively loose term that may contain different parameters, the types of dimensions that are studied in this thesis are related to: (a) the resources (e.g., number of cores), (b) the workflow (e.g., dataset size) and (c) application level parameters (e.g., amount of cache). Second, in order to be able to quickly estimate the utility of a dataset for a provided operator, it models its outcome when it is applied to a set of different datasets. In contrast to the previous case, the modeling problem here is not well defined: The different datasets belong to an unordered set where no relationship between them is known, i.e., they do not belong to a *metric space*. This makes the problem particularly challenging, since the traditional statistical inference approaches require the input and output domains to be metric spaces.

For addressing the first challenge, this thesis presents a performance modeling methodology the goal of which is to select the most representative points of the application configuration space and train a model that approximates performance for the entire space. The goal of this approach is to tackle all the challenges encountered when modeling Big Data applications: (a) Narrow down the enormous amount of possible configurations that are exponentially increased with application complexity, (b) reduce the cost of deployment, (c) prioritize configurations according to their importance and (d) identify interesting performance areas. Selecting the *appropriate* training points plays a dominant role for tackling these challenges. Nevertheless, this problem is not well explored in the performance modeling area and to the best of our knowledge, this methodology is the first that investigates the strategy of selecting good sample configurations.

Towards the direction of modeling an operator's outcome when applied over different datasets, a content-based modeling methodology is proposed, aiming at constructing a metric space for the examined datasets and applying Machine Learning techniques for approximating an operator's outcome for each of them. Based on the observation that similar datasets tend to impact an operator's output in similar ways, the proposed methodology quantifies dataset similarity, constructs a metric space that projects these relationships and exploits the power of Machine Learning in order to predict the operator's output for all of them. The goal of this approach is to address the challenges of common data analysis tasks: (a) avoid exhaustive operator execution for an enormous number of datasets, (b) minimize number of executions for operators of high complexity, (c) accommodate newly added datasets and (d) satisfy time-critical workflows that demand quick decisions based on data utility. To the best of our knowledge, this is the first approach that studies this problem from this point of view and, as discussed later in this dissertation, it presents encouraging results for a plethora of operators with diverse characteristics.

Hence, the work of this dissertation can be divided in two major parts:

**Performance Modeling**

In the first part of this thesis, an adaptive performance modeling methodology [GTK17, GTK18b] that focuses on Big Data applications is provided. Driven by the observation that many popular operators tend to present linear behavior on their performance for neighboring deployment setups, a divide-and-conquer modeling approach is employed, aiming at decomposing the configuration space in smaller disjoint regions and study each one separately. Our methodology relies on the mechanics of Oblique Decision Trees. At first, a set of randomly sampled configurations are obtained and the operator is executed for each one. Based on the obtained performance values, the space is partitioned in two regions with respect to maximizing the fit of each sample to a linear hyperplane (one for each subregion). The per-iteration deployment budget, i.e., the number of samples that are selected at each iteration, is distributed to each region considering a set of region properties such as its size, the modeling error, etc. Finally, the process is repeated until the global deployment budget exceeds. Intuitively, this methodology attempts to adaptively "zoom-in" to the regions of the configuration space that present certain peculiarities and are hard to model, without overlooking, though, to equally explore the entire configuration space in order to capture patterns for its entirety. Our experimental evaluation, conducted for multiple real-world, popular Big Data operators and various synthetic performance functions indicates that this approach outperforms other state-of-the-art modeling methodologies that solely focus either on exploring the configuration space or exploiting the obtained knowledge to further focus on space abnormalities. Moreover, the adoption of Oblique Decision Trees proves particularly efficient for identifying space abnormalities that entail multiple input dimensions.

**Content-Based Modeling of Analytics Operators**

In the second part of this dissertation, a content-based methodology [GTK18a, BGTK18] that aims at modeling an operator's outcome when applied for different datasets is introduced. Based on the observation that datasets with similar properties tend to affect the operators that are applied to them in similar ways, the approach attempts to quantify the similarity between the different datasets in the light of three foundational properties: their *statistical distribution*, their *size* and their *tuple ordering*. After calculating the pairwise similarities for all datasets, Multidimensional Scaling is utilized to convert the obtained Similarity Matrix to a low-dimensional *metric space* each point of which represents one dataset. The distance between different datasets in this metric space provide an estimate of their similarity (smaller distances implying higher similarity). Finally, the provided operator is executed for a small number of the available datasets and using the constructed metric space and the obtained operator outputs, a Neural Network is trained that approximates the operator outputs for all datasets. It should be stressed that the computationally intensive part of the scheme, i.e., the construction of the metric space, remains entirely operator agnostic, since the similarity estimation does not involve the execution of the operator. This property makes the approach scalable to an increasing number of operators, a property that is especially important considering the ever-increasing number of operators that are utilized in typical data analysis tasks. Our experimental evaluation has demonstrated that the proposed methodology can efficiently express the relationship between different datasets and accurately model the behavior of different types of operators with a minimal number of operator executions. Moreover, the introduced methodology is demonstrated to be applicable for datasets of different type, including tuple-based and graph-based datasets, only requiring the utilization of an appropriate similarity function.

The contributions made by this dissertation as a whole can be summarized as follows:

- This dissertation studies the problem of modeling the behavior of Big Data operators from two different angles including their performance for varying deployment configurations and their output for different datasets.

- It proposes an adaptive performance modeling methodology that aims at identifying and focusing on the areas of the performance function that present irregularities and are, thus, hard to be modeled. The proposed methodology particularly tackles the problem of selecting appropriate configurations to be deployed in order to maximize the modeling accuracy, given a fixed maximum number of configurations. Albeit the problem of constructing a training set with particular properties is discussed in other learning problems, this approach is the first that provides a solution for this problem for application performance modeling.

- It presents a content-based data modeling methodology that aims at modeling the output of an arbitrary operator when applied for a set of different datasets. The introduced approach particularly tackles the problem of creating a metric space that reflects the relationship between the different datasets when examined in the light of different data properties. This approach is the first that attempts to conduct such "black-box" modeling without involving the operator in the most computationally expensive part.

- All of the proposed methodologies have been implemented and are freely available as open source projects [sou18]. Their extensive experimental evaluation for a variety of operators and different datasets demonstrates their ability to model a wide variety of real-world operators from two different perspectives: The operator's performance and its output. The presented results demonstrate that both approaches construct models of high modeling accuracy and outperform other state-of-the-art methodologies.

## 1.3 Document Outline

The rest of this dissertation is organized as follows:

Chapter 2 presents the first contribution of this thesis that is the adaptive performance modeling methodology for Big Data applications. After the formal problem introduction, in this Chapter, a thorough description of the proposed methodology is provided and evaluated using both real-world applications and synthetic performance functions.

Chapter 3 introduces the second contribution of this dissertation, which is a content-based approach for modeling analytics operators. After providing the problem formulation, we proceed with a detailed analysis of the suggested workflow, provide an interesting extension to graph-based data and, finally, conclude with a thorough evaluation of the introduced scheme.

Chapter 4 compares this work with the related work in the literature, where Chapter 5 provides the conclusions and possible directions for future research.

Finally, Appendix A presents AURA, a fully automated cloud deployment system with error-recovery capabilities. AURA has been developed in order to accelerate application deployments conducted for the performance modeling methodology; Nevertheless, the problem solved by AURA is radically different from the one presented at Section 2 and is, thus, presented separately.

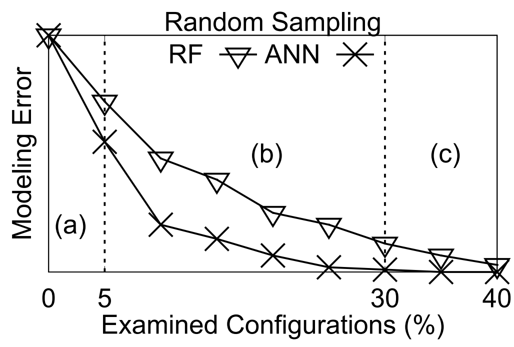
CHAPTER 2

# Adaptive Performance Modeling of Big Data Applications and Operators

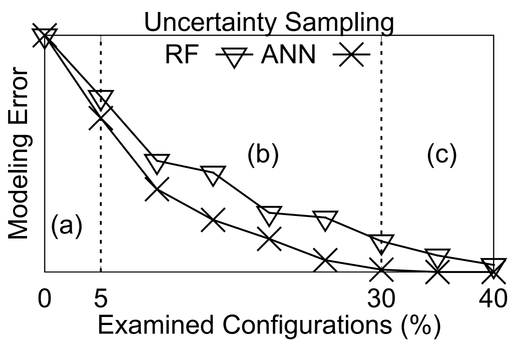
This chapter describes a performance modeling methodology for Big Data Applications and Operators. Our work models application performance using a divide-and-conquer approach: The application configuration space is decomposed to smaller regions recursively and each partition is modeled independently. According to the achieved accuracy, each partition is sampled and a few representative application configurations are deployed. Using these deployment values, the Performance Function for the entire space is approximated. Intuitively, our methodology attempts to maximize the configuration space *coverage* and, on the same time, “zooms-in” the regions of it that require further examination in order to be accurately modeled. A thorough evaluation, conducted for both real-world applications and synthetic performance functions, indicates that the proposed methodology is capable of producing highly accurate performance models than other, end-to-end profiling approaches. The key reason for that is that it efficiently achieves to bridge the gap between the necessity of *exploring* the application configuration space, i.e., select representative values and *exploiting* the previously obtained knowledge so that the available deployment budget is spent wisely.

## 2.1 Overview

As a motivating example, take the case of Wordcount, one of the most popular Big Data operators. Wordcount accepts as input a number of text files and produces a dictionary containing tuples in the form `<word, counter>`, where `word` represents any word that was encountered in any of the text files and `counter` refers to the number of its occurrences to them. For this batch operator, the performance metric, i.e., the measure that quantifies how well the operator functions, is the execution time. The execution time is affected by a wealth of input parameters: The number of machines it utilizes, the amount of memory each machine has, its CPU, etc. A traditional "black-box" modeling approach requires the operator to be executed for a handful, *representative* configurations, so that the performance metric for them becomes available, and then approximate the remainder configurations through statistical inference or a Machine Learning model. The building blocks of this scheme are two: (a) the *sampling* algorithm, that decides which configurations should be deployed and (b) the *modeling* approach, that determines how the sample configurations will be utilized in order to provide high-quality inference for all configurations. Figure 2.1 provides an estimate of the achieved modeling accuracy when two state-of-the-art sampling algorithms and modeling methodologies are utilized. Figure 2.1a depicts the modeling error of Random Forests (RF) and Artificial Neural Networks (ANN) when the configuration sampling is done in a Uniform fashion, whereas Figure 2.1b represents the case where Uncertainty Sampling [LC94] is utilized.



(a) Random sampling



(b) Uncertainty sampling

Figure 2.1: Modeling errors of different profiling schemes

Note that regardless of the selection of the sampling and modeling schemes, both figures can be decomposed in three areas. Area (a) represents the case where a very small number of configurations has been tested and, hence, little knowledge for the operator's behavior is available. Evidently, the modeling error is massive and independent of the underlying modeling tools. On the contrary, area (c) represents the opposite case: Many configurations are tested and, hence, the operator is profiled with high detail. Although different modeling and sampling schemes may produce models of different accuracy, the obtained knowledge at this point is massive and one can easily model the operator's performance. This knowledge, though, came with a cost: One has to deploy a massive amount of operator configurations, a practice which is cumbersome in terms of time and cost. Area (b), though, is located in an intermediate state between (a) and (c). At this point, the selection of the appropriate sampling and modeling policies can have a massive impact to the model's accuracy. A wise selection of *representative* configurations, can help a smart model to build the application's profile faster and with reduced costs. The task of finding representative samples, though, is not trivial as it is strongly operator-dependent (as different operators may be benefited from different sampling distributions) and computationally infeasible to calculate: The selection of a "representative" set of configurations is a combinatorial problem that entails incremental trial of different application setups that should be available at first place.

To tackle this challenge, in this work, we propose an adaptive performance modeling methodology that aims at iteratively refining the sampling policy in order to concurrently satisfy two key objectives: (a) maximize the configuration space coverage in order to capture all patterns and (b) focus on areas where the performance presents a more obscure behavior and it is modeled harder. Out methodology relies on the mechanics of Decision Trees. The main idea behind it is to decompose the configuration space in disjoint regions and model each one independently. Each region claims a specific portion of the *deployment budget*, i.e., number of samples, according to its size and modeling accuracy. The final outcome is a composite model that combines the results of simpler linear models, each of which covers a specific region of the configuration space. Intuitively, our approach attempts to model the entire space and adaptively "zoom-in" to areas where the performance function presents irregularities and it is hard to model.

In summary, we make the following contributions:

- We propose an adaptive, accuracy-driven profiling technique for Big Data applications that utilizes oblique Decision Trees. Our method natively decomposes the multi-dimensional input space into disjoint regions, naturally adapting to the complex performance application behavior in a fully automated manner. Our scheme utilizes three unique features relative to the standard Decision Tree algorithm: First, it proposes a novel expansion algorithm that constructs oblique Decision Trees by examining whether the obtained samples fit into a linear model. Second, it allows developers to provide a compromise between *exploring* the configuration space and *exploiting* the previously obtained knowledge. Third, it adaptively selects the most accurate modeling scheme, based on the achieved accuracy.

- We perform an extensive experimental evaluation over diverse, real-world applications and synthetic performance functions of various complexities. Our results showcase that our methodology is the most efficient, achieving modeling accuracies even $3\times$ higher that its competitors and, at the same time, it is able to create models that reflect abnormalities and discontinuities of the performance function orders of magnitude more accurately. Furthermore, our sampling methodology proves to be particularly beneficial for linear classifiers, as linear models trained with samples chosen by our scheme present even 38% lower modeling error.

## 2.2 Preliminaries

### 2.2.1 Problem formulation

Application profiling can be formulated as a function approximation problem [CMS16, GTPK15]. The application is viewed as a black-box that receives a number of inputs and produces a single (or more) output(s). The main idea behind constructing the performance model is to predict the relationship between the inputs and the output, without making any assumption regarding the application's architecture. Inputs reflect any parameters affecting application performance, such as the number and quality of different types of resources (e.g., cores/memory, number of nodes, etc.), application-level parameters (e.g., cache used by an RDBMS, HDFS block size, etc.), workload-specific parameters (e.g., throughput/type of requests) and dataset-specific parameters (e.g., size, dimensionality, distribution etc.).

Assume that an application comprises $n$ inputs and one output. We assume that the $i^{th}$ input, $1 \leq i \leq n$, receives values from a predefined finite set of values, denoted as $d_i$. The Cartesian product of all $d_i, 1 \leq i \leq n$ is the *Deployment Space* of the application $D = d_1 \times d_2 \times \cdots \times d_n$. Similarly, the application's output reflects values that correspond to a performance metric, indicative of its ability to fulfill its objectives. The set of the application's output will be referred to as the application's *Performance Space*. Based on the definitions of $D$ and $P$, we define the performance model $m$ of an application as a function $m : D \to P$. The estimation of the performance model entails the estimation of the performance value $b_i \in P$ for each $a_i \in D$. However, $|D|$ increases exponentially with $n$ (as $|D| = \prod_{i=1}^{n} |d_i|$), thus the identification of all performance values becomes prohibitive, both in terms of time and cost. A common approach to tackle this challenge is the extraction of a subset $D_s \subseteq D$ ($|D_s| \ll |D|$) and the estimation of the performance points $P_s$ for each $a_i \in D_s$. Using $D_s$ and $P_s$, model $m$ can be approximated, creating an approximate model $m'$. While $m' \to m$, the approximation is more accurate.

Note that this formulation is valid only under the assumption that distinct deployments are *reproducible*, i.e., in case where a given Deployment Space point is redeployed, the measured outcome is identical. For many reasons, such as the "noisy neighbor" effect [GVCL14], network glitches, power outages, etc., such an assumption can be violated when the application is deployed to cloud environments, because of the introduced unpredictability that distorts the application's behavior. The treatment of this dimension of the problem is outside the scope of our work. This works tackles the complexity introduced by the excessive dimensionality of the Deployment Space. The presented methodology can be, thus, directly applied to predictable environments with reduced interference, such as private cloud installations that, according to [rig], remain extremely popular, since they will host half of the user generated workloads for 2017, maintaining the trend from the previous years.

### 2.2.2 Decision Trees

*Classification and Regression Trees (CART)* [BFSO84], or Decision Trees (DT), are a very popular classification and regression approach. They are formed as tree structures containing intermediate (*test*) and *leaf* nodes. Each test node represents a boundary of the data space and each leaf node represents a class, if the DT is used for classification, or a linear model, if used for regression. The boundaries of the DT divide the original space into a set of disjoint regions.

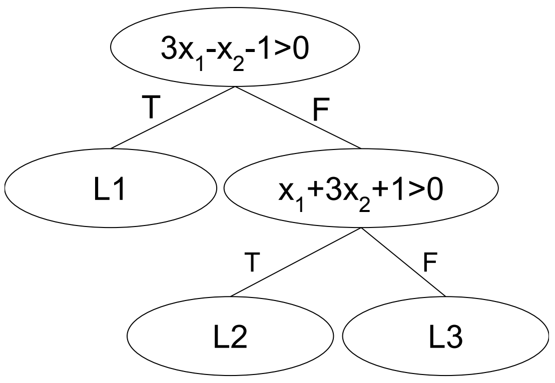
The construction of a DT is based on recursively partitioning the space of the data so as to create disjoint groups of points that maximize their intra-group homogeneity and minimize their inter-group homogeneity. The homogeneity metric of a group differs among the existing algorithms: *GINI impurity* has been used by the CART algorithm [BFSO84], *Information Gain* has been used by ID3 and C4.5 [Qui86] for classification, whereas the Variance Reduction [BFSO84] is commonly used for regression. These heuristics are applied to each leaf to decide which dimension should partitioned and at which value. The termination condition of the DT construction varies between different algorithms as the tree height is either pre-defined or dynamically decided, i.e., the tree grows until new leaves marginally benefit its accuracy.

Each boundary of a DT is parallel to one axis of the data, since it involves a single dimension of the data space, i.e., the boundary line is expressed by a rule of the form $x_i = c$, where $c$ is a constant value. Generalizing this rule into a multivariate line, we obtain the *oblique* DTs that consist of lines of the form:

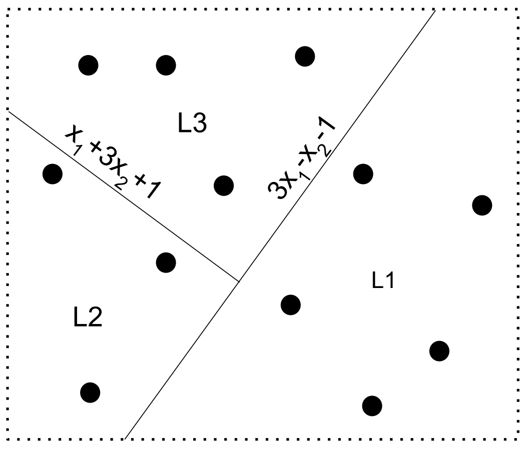
$$\sum_{i=1}^{n} c_i x_i + \gamma = 0$$

To better illustrate this, in Figure 2.2 we provide an example where we showcase the tree structure along with the respective partitions of the Deployment Space for a tree that has 3 leaves and 2 test nodes. Original (or *flat*) Decision Trees can be considered as a special case of the oblique Decision Trees. The multivariate boundaries boost the expressiveness of a DT, since non axis-parallel patterns can be recognized and expressed, as discussed in [HKS93].

In this work, we utilize oblique DTs in two ways: First, they are employed to create an approximate model of the performance function. Second, their construction algorithm is modified



(a) Oblique Decision Tree structure



(b) Space partitioning

Figure 2.2: Example of an Oblique Decision Tree

to adaptively sample the Deployment Space of the application, focusing more on regions where the application presents a complex behavior and ignoring regions where it tends to be easily predictable.

## 2.3 Profiling Methodology

### 2.3.1 Method overview

The main idea of the suggested algorithm is to partition the Deployment Space by grouping samples that better fit different linear models, infer knowledge about the performance function through sampling the Deployment Space, deploying the selected configurations and, when a predefined number of deployments is reached, model the performance function utilizing different linear models for each partition. This is schematically represented in Figure 2.3. Specifically, at each step, the Deployment Space is partitioned by grouping the already obtained samples according to their ability to create a linear model that accurately approximates the application performance. Estimates are then created regarding the intra-group homogeneity, which corresponds to the prediction accuracy of the performance function for the specified region. Therefore, poorly approximated regions need to be further sampled in order to be better approximated, whereas accurate regions need not to be further explored.

Intuitively, the suggested algorithm is an attempt to adaptively "zoom-in" to areas of the Deployment Space where the behavior of the performance function is more *obscure*, in the sense that it is unpredictable and hard to model. This enables the number of allowed application deployments to be dynamically distributed inside the Deployment Space, leading to more accurate

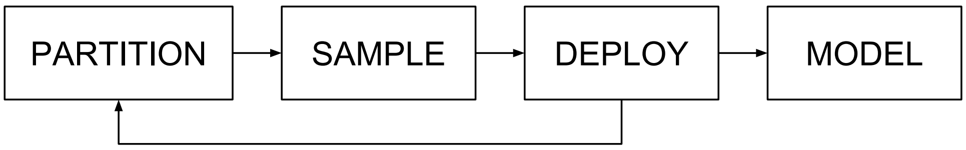


Figure 2.3: Method overview

predictions as more samples are collected for performance areas that are harder to approximate. This smart distribution of the deployed samples requires the closer examination of regions of the Deployment Space independently, something that is inherently conducted by DTs. Their properties, such as their scalability to multiple dimensions, robustness, their innate divide-and-conquer functionality, etc., render them a perfect fit for the application profiling problem and facilitate the decomposition of the Deployment Space in an intuitive and efficient manner. In Algorithm 1 we provide the pseudocode of the suggested methodology.

**Algorithm 1** DT-based Adaptive Profiling Algorithm

```
1: procedure DTAdaptive($D$, $B$, $b$)
2:   $tree \leftarrow$ treeInit($\emptyset$), $samples \leftarrow \emptyset$
3:   while $|samples| \leq B$ do
4:     $tree \leftarrow$ partition($tree$, $samples$)
5:     $s \leftarrow$ sample($D$, $tree$, $samples$, $b$)
6:     $d \leftarrow$ deploy($s$)
7:     $samples \leftarrow samples \cup d$
8:   $model \leftarrow$ createModel($samples$)
9:   return $model$
```

Our Algorithm takes three input parameters: (a) the application's Deployment Space $D$, (b) the maximum number of samples $B$ and (c) the number of samples selected at each algorithm iteration $b$. The *tree* variable represents a DT, while the *samples* set contains the obtained samples. While the number of obtained samples is less than $B$, the following steps are executed: First, the leaves of the tree are examined and tested whether they can be replaced by subtrees that further partition their regions (Line 4). The leaves of the expanded tree are, then, sampled with the *SAMPLE* function (Line 5), the chosen samples are deployed (Line 6) according to the sample's deployment configuration and performance metrics are obtained. Note that, $s \subseteq D$ whereas $d \subseteq D \times P$, i.e., $s$ contains the sampled configurations and $samples$ contains the deployment configurations along with their performance value. Finally, when $B$ samples have been chosen, the final model is created (Line 8).

### 2.3.2 Decision Tree Expansion

The main idea behind the expansion of a DT lies on the identification of a split line that maximizes the homogeneity between the two newly created leaves. The selection of an appropriate line greatly impacts the accuracy of the partitioning, ergo the future partitions. Recall that, the utilization of *oblique* DTs allows the test node to be represented by a multivariate line and enhances their adaptability to different performance functions. Nevertheless, calculating an optimal multivariate split line is NP-complete [HKS93]. Since we want to exploit the expressiveness of the oblique partitions without introducing a prohibitive computation cost for their estimation, we express the problem as an optimization problem [HKS93] and utilize Simulated Annealing (SA) [VLA87] to identify a near-optimal line. In Algorithm 2, we present the PARTITION function. For each leaf of the tree (Line 3), SA is executed to identify the best multivariate split line (Line 5), considering solely the samples whose Deployment Space-related dimensions ($d.in$) lie inside the specified leaf (Line 4). The samples of the specified leaf are then partitioned in two disjoint sets according to their position (Lines 7-11), in which the symbol $\preceq$ indicates that a sample $s$ is located below $line$. Finally, a new test node is generated (Line 12), replacing the original leaf node of the tree and the new tree is returned.

**Algorithm 2** Partitioning Algorithm

```
1:  procedure PARTITION($tree$, $samples$)
2:    $newTree \leftarrow tree$
3:    for $l \in$ leaves($tree$) do
4:      $v \leftarrow \{d|d \in samples, d.in \in l\}$
5:      $line \leftarrow SA(v)$
6:      $L_1 \leftarrow \emptyset, L_2 \leftarrow \emptyset$
7:      for $s \in v$ do
8:        if $s \preceq line$ then
9:          $L_1 \leftarrow L_1 \cup s$
10:       else
11:         $L_2 \leftarrow L_2 \cup s$
12:     $testNode \leftarrow \{line, L1, L2\}$
13:     $newTree \leftarrow$replace($l$, $testNode$)
14:   return $newTree$
```

SA is a prominent methodology for solving complex optimization problems. It functions in an iterative manner and is based on the idea of generating possible solutions (split lines), evaluating their goodness using a score function and, finally, returning the best. The key factor that diversifies SA from a simple random search lies on the targeted creation of candidate solutions.

Assuming a split line of the form

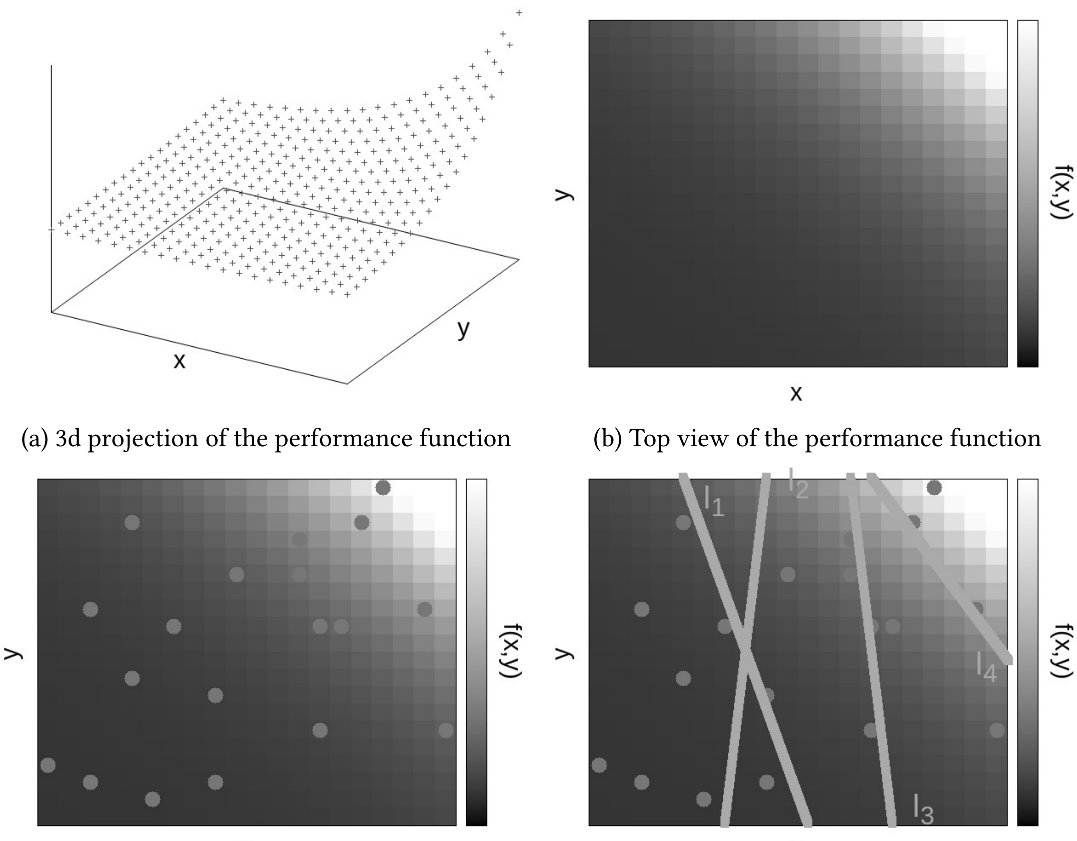
$$l = a_1 x_1 + \cdots + a_n x_n + a_{n+1} = 0$$

SA seeks for a solution vector $v = (a_1, \cdots, a_{n+1})$ that minimizes an expression $Score(v)$. During the first algorithm steps ($i$), the consecutive candidate solutions present great differences , i.e., the difference $|v_i - v_{i+1}|$, is large, but for an increasing number of iterations, $|v_i - v_{i+1}| \to 0$ and SA converges to a close-to-the-optimal solution. During the algorithm's execution, SA may pick worse solutions, i.e., split lines with worse scores, as the best examined this-far. This guarantees that the algorithm will not be trapped into a local minimum. Both the probability of substituting a better solution with a worse one and the size of the neighborhood of solutions that SA examines at each step is determined by *Temperature* factor, decaying with time. Intuitively, higher temperatures mean that SA attempts diverse solutions and accepts worse solutions with high probability and lower temperature implies that SA converges to the neighborhood of the best solution.

The cornerstone of SA's effectiveness is the score function that quantifies the efficacy of each candidate solution. The methodologies utilized by various DT construction algorithms make no assumption regarding the nature of the data. Although this enhances their adaptability to different problem spaces, their utilization in this work resulted in poorly partitioned Deployment Spaces. The data that our work attempts to model belong to a performance function. Intuitively, if all the performance function points were available, one would anticipate that a closer observation of a specific region of the Deployment Space would present an approximately linear behavior, as "neighboring" Deployment Space points are anticipated to produce similar performance. When focusing on a single neighborhood, this "similar performance" could be summarized by a linear hyperplane. A split line is considered to be "good" when the generated leaves are best summarized by linear regression models or, equivalently, if the samples located in the two leaves can produce linear models with low modeling error. Given a Deployment Space $D$, a line $l$ and two sets $L_1$, $L_2$ that contain $D$'s samples after partitioning it with $l$ (as in lines 6-10 of the algorithm), we estimate two linear regression models for $L_1$ and $L_2$ and estimate their residuals using the coefficient of dzetermination $R^2$. $l$'s score is:

$$Score(l) = -\frac{|L_1| R^2_{L_1} + |L_2| R^2_{L_2}}{|L_1| + |L_2|} \tag{2.1}$$

Note that, $0 \leq R^2 \leq 1$ and a value of $1$ represents perfect fit to the linear model. $Score(l)$ is minimized when both $L_1$ and $L_2$ generate highly accurate linear models or, equivalently, if the two sets can be accurately represented by two linear hyperplanes. We weight the importance of each set according to the number of samples they contain as an inaccurate model with more samples has greater impact to the accuracy than an inaccurate model with fewer samples. The negative sign is employed in order to remain aligned with the literature, in which SA seeks for minimum points. Finally, one SA property not discussed this far is the ability to achieve a customizable compromise between the modeling accuracy and the required computation. Specifically, prior to SA's execution, the user defines a maximum number of iterations to be executed for the identification of the best split line. When one wants to maximize the quality of the partitioning, many iterations are conducted. On the contrary, the number of iterations is set to lower values in order to allow for a rapid split line estimation.



(a) 3d projection of the performance function

(b) Top view of the performance function

(c) Top view with samples

(d) Candidate split lines

Figure 2.4: An example of Decision Tree Expansion for $f(x, y) = (x + y)\exp(x + y)$

In order to better illustrate the functionality of the Decision Tree expansion algorithm, consider the example depicted in Figure 2.4. Figure 2.4a depicts a synthetic three dimensional performance function where $f(x, y) = (x + y)\exp(x + y)$. Figure 2.4b depicts a top view of it, where the horizontal and vertical axis represent $x$ and $y$ of Figure 2.4a respectively and the colors represent the value of the function (lighter colors means larger $f$ values). For example, one can see that when both $x$ and $y$ obtain large values, $f$ is maximized and this is highlighted with lighter colors on the top right corner of Figure 2.4b. Figure 2.4c provides the same top view, but now we assume that the values of the performance functions for the depicted points are known.

Figure 2.4d provides an example of execution of Algorithm 2 for these samples. Initially, the algorithm picks a random line ($l_1$) that is likely to produce a poor space partition. The values $R^2_{L_1}, R^2_{L_2}$ and the size of the two regions (i.e., the one on the left and on the right of $l_1$) are calculated and, subsequently, the Score of Equation 2.1 is estimated. Observe that although $L_1$ presents linear behavior and, hence, $R^2_{L_1}$ receives values close to 1, $L_2$ contains the non-linear region where the Performance Function presents a strong exponential increase, hence the values of $R^2_{L_2}$ will be much lower. Simulated Annealing generates other possible split lines (e.g., $l_2$, $l_3$, etc.) until a close-to-optimal line (i.e., $l_4$) is found. By comparison of Figures 2.4a and 2.4d, the reason of selecting this line is evident: It separates the Performance Function in two subregions each of which is accurately represented by a linear hyperplane. Note that the separation objective is not to create a perfect fit for any one of the two generated hyperplanes. The utilization of the partition size in Equation 2.1 facilitates weighting the achieved modeling accuracy. For example, the modeling accuracy for the left partition when $l_1$ was utilized was higher than the respective accuracy for $l_4$. However, if we consider the modeling accuracies at both partitions, $l_4$ produces better results and was, thus, selected.

### 2.3.3 Adaptive Sampling

The previously described construction of the Decision Tree is based on the principle that areas of the Deployment Space that make the Performance Function present a linear behavior, i.e., the values of the latter can be expressed as a linear combination of the dimensions of the former with high accuracy, should belong to the same partition(s). What happens, though, for areas where the linear approximation presents poor results? In this section, we examine how our method approaches these strongly non-linear areas with the help of an adaptive sampling scheme.

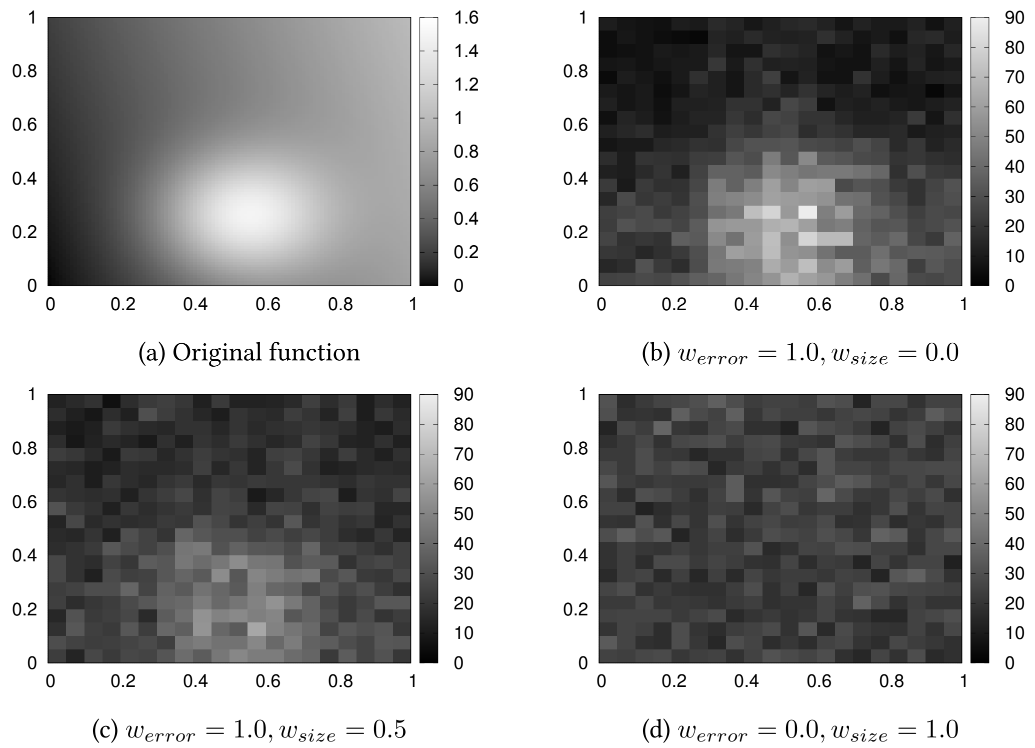
One initial observation that should be made is that any continuous non-linear function can be approximated by a piecewise linear function with accuracy that is proportional to the number of segments of the later. More segments make the piecewise function to better approximate the non linear one. In theory, if the number of segments tends to infinity, the achieved accuracy is 100%, or, equivalently, the approximation error tends to zero. Nevertheless, given that the computation of each segment requires (at least) two points of the non-linear line, more segments can only be employed if more points of the original function are estimated. If we apply the same reasoning to our multidimensional problem, a non-linear curve (which might be a part of the Performance Function under examination) can be decomposed to smaller linear hyperplanes only if enough Deployment Space points are deployed and available for this region. Intuitively, this means that the decomposition described in the previous section can only take place only when more *information* becomes available for the region where the behavior of the Performance Function is harder to approximate. Thus, the sampling algorithm should favor such areas and select more samples from regions that are poorly approximated.

On the contrary, one should not overlook that some parts of the Deployment Space may present certain peculiarities that are not resolved by selecting more samples. For example, take the case of a two-tier Web Application that runs a given workload with the minimum available resources. Some workloads may fail or others may take an enormous time to finish for a variety of reasons (e.g., thrasing, OOM kills, etc.). Points of the Performance Function close to that region (i.e., the one that represents minimum resources setups) will present a random behavior that cannot by realistically modeled by any approximation technique. One should not invest a big portion of the available deployment budget to areas of this type, since they do not offer any knowledge regarding the performance function and they only waste the given budget. Evidently, there is a clear trade-off that needs to be compromised in order to maximize the modeling efficiency: More samples should be selected for areas that present non-linear behavior but there should be a constraint on how much samples one should deploy for each region. Given these notes, we now provide the sampling algorithm that compromises this trade-off.

**Algorithm 3** Sampling algorithm

```
1:  procedure SAMPLE($D$, $tree$, $samples$, $b$)
2:    $errors, sizes \leftarrow \emptyset, maxError, maxSize \leftarrow 0$
3:    for $l \in$ leaves($tree$) do
4:      $points \leftarrow \{d | d \in samples, d.in \in l\}$
5:      $m \leftarrow$ regression($points$)
6:      $errors[l] \leftarrow$ crossValidation($m$,$points$)
7:      $sizes[l] \leftarrow |\{e | e \in D \cap l\}|$
8:      if $maxError \leq errors[l]$ then
9:        $maxError \leftarrow errors[l]$
10:     if $maxSize \leq sizes[l]$ then
11:       $maxSize \leftarrow sizes[l]$
12:   $scores, newSamples \leftarrow \emptyset, sumScores \leftarrow 0$
13:   for $l \in$ leaves($tree$) do
14:     $scores[l] \leftarrow w_{error} \cdot \frac{errors[l]}{maxError} + w_{size} \cdot \frac{sizes[l]}{maxSize}$
15:     $sumScores \leftarrow sumScores + scores[l]$
16:   for $l \in$ leaves($tree$) do
17:     $leafNoDeps \leftarrow \lceil \frac{scores[l]}{sumScores} \cdot b \rceil$
18:     $s \leftarrow$ RANDOMSELECT($\{d | d \in D \cap l\}, leafNoDeps$)
19:     $newSamples \leftarrow newSamples \cup s$
20:   return $newSamples$
```

After the tree expansion, the SAMPLE function is executed (Algorithm 3). The algorithm iterates over the leaves of the tree (Line 3). For the samples of each leaf, a new linear regression model is calculated (Line 5) and its residuals are estimated using Cross Validation [Gei93]: The higher the residuals, the worse the fit of the points to the linear model. The size of the specified leaf is then estimated. The size is equal to the number of acceptable deployment combinations, i.e., the points of the Deployment Space lying inside the region of the specified leaf. Thus, the portion of the Deployment Space that is represented by the leaf is an estimate of its size. Different estimates could also be used (e.g., measuring the surface of the leaf). This representation was chosen due to its simplicity. After storing both the error and the size of each leaf into a map, the maximum leaf error and size are calculated (Lines 8-11). Subsequently, a score is estimated for each leaf (Lines 13-15). The score of each leaf is set to be proportional to its scaled size and error. This normalization is conducted so as to guarantee that the impact of the two factors is equivalent. Two coefficients $w_{error}$ and $w_{size}$ are used to assign different weights to each measure. These scores are accumulated and used to proportionally distribute $b$ to each leaf (Lines 16-19). In that loop, the number of deployments of the specified leaf is calculated and new samples from the subregion of the Deployment Space are randomly drawn with the *RANDOMSELECT* function, in a uniform manner. Finally, the new samples set is returned.


(a) Original function

(b) $w_{error} = 1.0, w_{size} = 0.0$

(c) $w_{error} = 1.0, w_{size} = 0.5$

(d) $w_{error} = 0.0, w_{size} = 1.0$

Figure 2.5: Distribution of samples for different weights

To further analyze the sampler's functionality, we provide an example of execution for a known performance function of the form $y = 0.8x_1 + 0.2x_2$. On a randomly selected point, an abnormality is introduced, modeled by a Gaussian function. In Figure 2.5 (a) a projection of the performance function is provided. The horizontal and vertical axes represent $x_1$ and $x_2$ respectively and the colors represent $y$ values, where the lighter colors demonstrate higher $y$ values. Algorithm 1 is executed for different $w_{error}$, $w_{size}$ during the $\mathrm{SAMPLE}$ step. We assume a maximum number of deployments $B$ of 100 points out of the 2500 available points and a per-iteration number of deployments $b$ of 10 points. In Figures 2.5 (b), (c) and (d) we provide the distribution of the selected samples, for different weight values. Each dimension is divided in 20 intervals and for each execution we keep count of the samples that appear inside each region. The color of the regions demonstrate the number of the samples within the region (lighter colors imply more samples).

The adaptiveness of the proposed methodology lead the samples to immediately identify the abnormal area. In Figure 2.5 (b) the score of each leaf is only determined by its error. Most samples are gathered around the Gaussian distribution: The first leaves that represent the area of the Gaussian function produce less accurate models since they cannot express the performance function with a linear model. Since the score of each leaf is only determined by its error, these leaves claim the largest share of $b$ at each step, thus the samples are gathered around the abnormality. On the contrary, when increasing $w_{size}$ as in Figure 2.5 (c), the gathering of the samples around the abnormality is neutralized as more samples are now distributed along the entire space, something that is intensified in Figure 2.5 (d), where the abnormality is no longer visible.

The consideration of two factors (error and size) for deciding the number of deployments spent at each leaf targets the trade-off of *exploring* the Deployment Space versus *exploiting* the obtained knowledge, i.e., focus on the abnormalities of the space and allocate more points to further examine them. This is a well-known trade-off in many fields of study [SB98]. In our approach, one can favor either direction by adjusting the weights of leaf error and size, respectively. Note that this schemes enables the consideration of other parameters as well, such as the deployment cost through the extension of tre score function (Line 14). This way, more "expensive" deployment configurations, e.g., ones that entail multiple VMs with many cores, would be avoided in order to regulate the profiling cost.

Taking this one step further, using a score function allows us to take multiple parameters into consideration. Big Data applications are inherently deployed to cloud environment and, thus, the deployment cost is one such criterion. Deployment coordinates are not equally costly as providers charge for specific resources. For example, VMs with more cores or main memory may cost more. In order to profile an application taking into consideration the monetary cost as well, we could assume a score function of the following form, $l$ being the specified leaf:

$$score(l) = w_{error} \cdot error(l) + w_{size} \cdot size(l) - w_{cost} \cdot cost(l) \tag{2.2}$$

in which the cost of the leaf is calculated as the average cost of the deployment points inside the leaf. Such a function would penalize leaves that contain more expensive deployment configurations. Adjusting the way each factor affects the scoring function allows developers to achieve a satisfactory compromise between the contradicting parameters of focusing on the most unpredictable regions (error), exploring the space (size) and minimizing the monetary cost of deployments (cost).

### 2.3.4 Modeling

After $B$ samples are returned by the profiling algorithm, they are utilized by the *CREATEMODEL* (Algorithm 1, Line 8) function to train a new DT. The choice of training a new DT instead of expanding the one used during the sampling phase is made to maximize the accuracy of the final model. When the first test nodes of the former DT were created, only a short portion of the samples were available to the profiling algorithm and, hence, the original DT may have initially created inaccurate partitions. Moreover, in cases where the number of obtained samples is comparable to the dimensionality of the Deployment Space, the number of constructed leaves is extremely low and the tree degenerates into a linear model that covers sizeable regions of the Deployment Space with reduced accuracy. To overcome this limitation, along with the final DT, a set of Machine Learning classifiers are also trained, keeping the one that achieves the lowest Cross Validation error. However, when the DT is trained with enough samples, it outperforms all the other classifiers. This is the main reason for choosing the DT as a base model for our scheme: The ability to provide higher expressiveness by composing multiple linear models in areas of higher unpredictability, make them a perfect choice for modeling a performance function. Note that, the linearity of this model does not compromise its expressiveness: Non-linear performance functions can also be accurately approximated by a piecewise linear model, through further partitioning the Deployment Space.

The case where the tree is poorly partitioned during the initial algorithm iterations, can also produce vulnerabilities during the sampling step. Specifically, the Deployment Space may be erroneously partitioned on the first levels of the tree and then, as new samples arrive, more deployments will be spent on creating more and shorter regions that, would otherwise have been merged into a single leaf. To address this problem, commonly anticipated when constructing a Tree in an "online" fashion, several solutions have been proposed [TÖ09]. In our approach, the DT is recreated from scratch prior to the sampling step. This optimization, albeit requiring extra computation, boosts the performance of the profiling methodology, as better partitioning of the space leads to more representative regions and better positioned models. Nevertheless, the time needed for this extra step is marginal compared to the deployment time for most modern distributed applications.

### 2.3.5 Complexity analysis

In order to estimate the complexity of Algorithm 1, let us first, estimate the complexity of its building blocks, i.e., the `PARTITION` and `SAMPLE` functions provided in Algorithms 2 and 3 respectively.

Considering a $d$-dimensional Dataset Space with a maximum number of samples $B$ and a maximum number of Simulated Annealing iterations equal to $N_{SA}$, the simplified complexity of Algorithm 2 is

$$\mathcal{O}(d^2 \cdot B^2 \cdot N_{SA}).$$

The most computationally expensive part of Algorithm 2 takes place at line 5, where SA produces and evaluates candidate split lines. For each generated line, two linear regression models are estimated (in time $\mathcal{O}(d^2 \cdot B)$ in the worst case) and evaluated and this is repeated at worst $N_{SA}$ times and for all the available tree leaves, i.e., $B$, hence the above. Note that this is the worst case; In the case where the Decision Tree is balanced, i.e., the samples are equally divided to all leaves), the number of leaves are $\log(B)$ and, hence, fewer computations are required. Nevertheless, since this strongly dependent to the objective performance function, we continue our analysis estimating the worse complexity. The sampling step, has a complexity of

$$\mathcal{O}(d^2 \cdot B^2)$$

which is calculated considering that a linear regression model is constructed for each leaf of the tree, i.e., $B$ leaves at worst and $d^2 \cdot B$ computations for linear regression. Since this complexity is smaller than the Tree expansion step, this factor can be omitted.

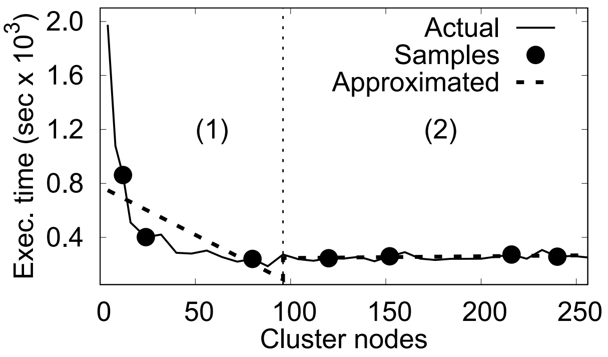
Finally, considering that the `PARTITION` and `SAMPLE` steps take place at each algorithm iteration, i.e., $\frac{B}{b}$ times, the total algorithm complexity equals:

$$\mathcal{O}(\frac{d^2 \cdot B^3 \cdot N_{SA}}{b}).$$

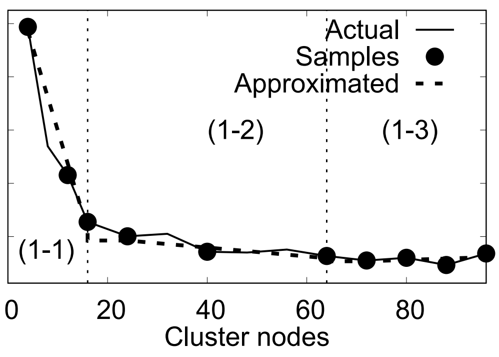
Evidently, the above complexity seems prohibitive for Deployment Spaces of high dimensionality $d$ and large deployment budgets $B$ as the Algorithm's execution time grows rapidly with these quantities (i.e., in a quadratic and cubic way respectively). Nevertheless, the following considerations should be made. First, we do not anticipate the Deployment Space dimensionality to grow beyond limits. Most typical performance modeling approaches analyze the applications in the light of a handful of parameters, that do not typically surpass 5 dimensions. It is, thus, not expected that the application dimensionality is soon to become the bottleneck of this approach. Second, the cubic factor on the number of samples implies that Algorithm 1 scales badly with an increasing number of samples. However, the entire idea behind this algorithm is to *minimize* this quantity: The algorithm was designed with the principle that $B$ will be relatively small and complex operations will be executed in order to choose samples wisely. Finally, this analysis did not consider the cost of running the DEPLOY step in line 6 of Algorithm 1 since this is expected to remain unaffected by $B$, $d$ and $N_{SA}$. In the reality, though, this step is the most time consuming since $B$ deployments need to take place in a cloud platform, which is a complex and error-prone procedure. From this perspective, we consider the above complexity to be marginal in comparison to the time needed to actually deploy a cloud application in numerous different ways.

### 2.3.6 An end-to-end example

Finally, before proceeding to evaluating Algorithm 1, it is useful to provide an example of the Algorithm's execution that highlights how different algorithm steps evolve through time. For this example, consider the execution of the Wordcount operator. The operator is executed for a 100G synthetic dataset, on a Hadoop cluster of varying sizes, i.e., the Deployment Space is 1-d and it represents the number of nodes of the Hadoop cluster (4–256 nodes) and the Performance Space represents the execution time. We execute our methodology for a budget of $B = 14$ configurations and $b = 7$ for each iteration. In Figures 2.6 (a) and (b) we depict the actual and approximated performance functions for the first and second algorithm iterations, respectively.



(a) First iteration



(b) Second iteration

Figure 2.6: Algorithm execution for Wordcount

During the first iteration, 7 samples are randomly picked and the first partitioning of the Deployment Space takes place (for a cluster of 96 nodes) generating the areas (1) and (2) of Figure 2.6 (a). Note that, the performance function in area (2) presents linear behavior and, hence, is extremely accurately approximated with only 4 tested configurations. On the contrary, the performance function is strongly non-linear in area (1) and, hence, poorly approximated. The scores of the two Deployment Space areas are, then, calculated, and in the following iteration all the 7 available configurations are assigned to area (1), because the error of the area (2) is zero (we assume that $w_{size} = 0$).

In Figure (b) we only depict area (1) as the rest of the Deployment Space remains the same. The new samples contribute in the creation of more and shorter partitions which, in turn, make the model accurately approximate the Actual performance function using a piecewise linear function. The final model achieved to approximate the actual performance function extremely accurately, examining only a mere 5% of the Deployment Space. This example is indicative of our methodology's power: Our divide-and-conquer approach allows us to sample and model each region separately, assigning different level of detail to different regions of the Deployment Space.

Moreover, the piecewise linear function is extremely effective for the performance functions of typical Big Data applications and operators, as the one examined in this example, since the behavior observed in this example (i.e., the "Actual" line) is commonly encountered, especially when the Deployment Space consists of resource-related dimensions. This enables us to provide extremely accurate approximations with a minimal number of deployments. Finally, even when addressing performance functions of high complexity and strongly non-linear behavior, the utilization of more and shorter partitions (as in the area (1) of Figure 2.6 (a)) guarantees that the *actual* line can be accurately approximated through the deployment of more samples.

## 2.4 Experimental Evaluation

At this point, the previously discussed methodology is evaluated. The evaluation aims at demonstrating that the proposed algorithm:

- outperforms other, state of the art, end-to-end profiling methodologies (Section 2.4.1).
- scales to performance functions of varying dimensionality and complexity with satisfying accuracy (Section 2.4.2).
- can be configured in order to accelerate modeling or provide higher modeling accuracy (Section 2.4.3).
- exploits oblique decision trees in order to approximate regions of the deployment space with abnormalities and discontinuities (Section 2.4.4).
- can take into consideration other deployment parameters such as the cost of the selected deployment configurations (Section 2.4.5).

**Methodology and Applications:** To evaluate the accuracy of our profiling algorithm, we test it over various real and synthetic performance functions. The *Mean Squared Error* (MSE) and *Mean Absolute Error* (MAE) metrics are utilized for the comparison, estimated over the *entire* Deployment Space, i.e., we exhaustively deploy all possible combinations of each application's configurations, so as to ensure that the generated model successfully approximates the original function for the entire space. We have deployed four different popular real-world Big Data operators and applications, summarized in Table 2.1. We opted for applications with diverse characteristics with Deployment Spaces of varying dimensionality (3 – 7 dimensions). The first three operators are implemented in Spark (k-means, Bayes) and Hadoop (Wordcount) and they are deployed to a YARN cluster. In all cases, the performance metric corresponds to the execution time. MongoDB is deployed as a sharded cluster and it is queried using YCSB [CST+10]. The sharded deployment of MongoDB consists of three components: (a) A configuration server that holds the cluster metadata, (b) a set of nodes that store the data (MongoD) and (c) a set of loadbalancers that act as endpoints to the clients (MongoS). Each application was deployed in a private Openstack cluster with 8 nodes aggregating 200 cores and 600GB of RAM.

Table 2.1: Applications under profiling

| **Application (perf. metric)** | **Dimensions** | **Values** |
|---|---|---|
| Spark k-means (execution time) | YARN nodes | 2–20 |
| | # cores per node | 2–8 |
| | memory per node | 2–8 GB |
| | # of tuples | 200–1000 ($\times 10^3$) |
| | # of dimensions | {1,2,3,5,10} |
| | data skewness | 5 levels |
| | k | {2,5,8,10,20} |
| Spark Bayes (execution time) | YARN nodes | 4–20 |
| | # cores per node | 2–8 |
| | memory per node | 2–8 GB |
| | # of documents | 0.5–2.5 ($\times 10^6$) |
| | # of classes | 50–200 classes |
| Hadoop Wordcount (execution time) | YARN nodes | 2–20 |
| | # cores per node | 2–8 |
| | memory per node | 2–8 GB |
| | dataset size | 5–50 GB |
| MongoDB (throughput) | # of MongoD | 2–10 |
| | # of MongoS | 2–10 |
| | request rate | 5–75 ($\times 10^3$) req/s |

We have also generated a set of synthetic performance functions using mathematical expressions, listed in Table 2.2. Each function maps the n-dimensional Deployment Space to a single dimensional metric and consists of $n$ dimensions such that $\mathbf{x} = (x_1, x_2, \cdots, x_n)$. The listed functions are chosen with the intention of testing our profiling algorithm against multiple scenarios of varying complexity and dimensionality. To quantify each function's complexity, we measure how accurately can each function be approximated by a linear hyperplane. Linear performance functions can be approximated with only a handful of samples, hence we regard them as the least complex case. For each of the listed functions, we calculate a linear model that best represents the respective data points and test its accuracy using the coefficient of determination $R^2$, whose value indicate the linearity of the respective function (1 indicating total linearity and 0 the opposite). Based on $R^2$ values for each case, we generate three complexity classes: Functions of *LOW* complexity (when $R^2$ is higher than 0.95), functions of *AVERAGE* complexity (when $R^2$ is between 0.5 and 0.7) and functions of *HIGH* complexity when $R^2$ is close to 0. The values of $R^2$ depicted in Table 2.2 refer to two-dimensional Deployment Spaces.

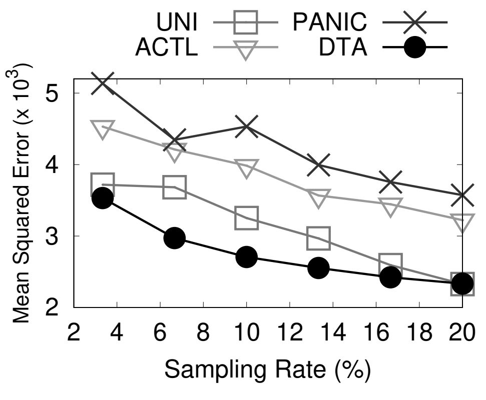
Table 2.2: Synthetic performance functions

| **Complexity** | **Name** | **Function** | $R^2$ |
|---|---|---|---|
| LOW | LIN | $f_1(\mathbf{x}) = a_1 x_1 + \cdots + a_n x_n$ | 1.00 |
| | POLY | $f_2(\mathbf{x}) = a_1 x_1^2 + \cdots + a_n x_n^2$ | 0.95 |
| AVG | EXP | $f_3(\mathbf{x}) = e^{f_1(\mathbf{x})}$ | 0.65 |
| | EXPABS | $f_4(\mathbf{x}) = e^{\lvert f_1(\mathbf{x})\rvert}$ | 0.62 |
| | EXPSQ | $f_5(\mathbf{x}) = e^{-f_1(\mathbf{x})^2}$ | 0.54 |
| HIGH | GAUSS | $f_6(\mathbf{x}) = e^{-f_2(\mathbf{x})}$ | 0.00 |
| | WAVE | $f_7(\mathbf{x}) = \cos(f_1(\mathbf{x})) \cdot f_3(\mathbf{x})$ | 0.00 |
| | HAT | $f_8(\mathbf{x}) = f_2(\mathbf{x}) \cdot f_6(\mathbf{x})$ | 0.00 |

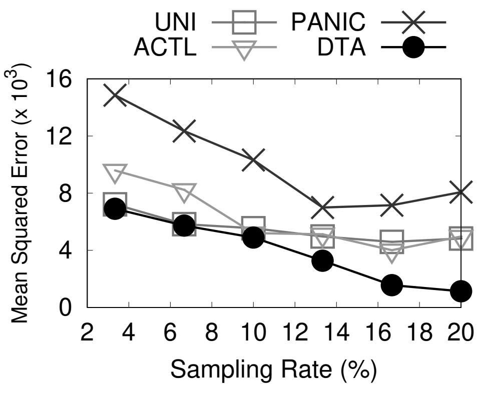
Finally, we compare our profiling methodology against other end-to-end profiling schemes. Our approach is referred to as DT-based Adaptive methodology (*DTA*). Active Learning [Set10] (*ACTL*) is a Machine Learning field that specializes on exploring a performance space by obtaining samples assuming that finding the class or the output value of the sample is computationally hard. We implemented *Uncertainty Sampling* that prioritizes the points of the Deployment Space with the highest uncertainty, i.e., points for which a Machine Learning model cannot predict their class or continuous value with high confidence. PANIC [GTPK15] is an adaptive approach that favors points belonging into steep areas of the performance function, utilizing the assumption that the abnormalities of the performance function characterize it best. Furthermore, since most profiling approaches use a randomized sampling algorithm [GCMS15, CMS16, KRDZ10, KRG+12] to sample the Deployment Space and different Machine Learning models to approximate the performance, we implement a profiling scheme where we draw random samples (*UNI*) from the performance functions and approximate them using the models offered by WEKA [HFH+09], keeping the most accurate in each case. In all but a few cases, the Random Committee [Rok10] algorithm prevailed, constructed using Multi-Layer Perceptron as a base classifier. For each of the aforementioned methodologies, we execute the experiments 20 times and present the median of the results.

## 2.4.1 Comparison of end-to-end profiling methods for varying Sampling Rate

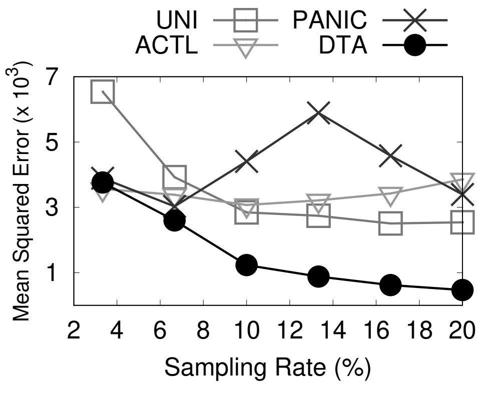
We first compare the four methods against a varying Sampling Rate, i.e., the portion of the Deployment Space utilized for approximating the performance function ($SR = \frac{|D_s|}{|D|} \times 100\%$). SR varies from 3% up to 20% for the tested applications. In Figure 2.7 we provide the accuracy of each approach measured in terms of MSE and in Figure 2.8 in terms of MAE.



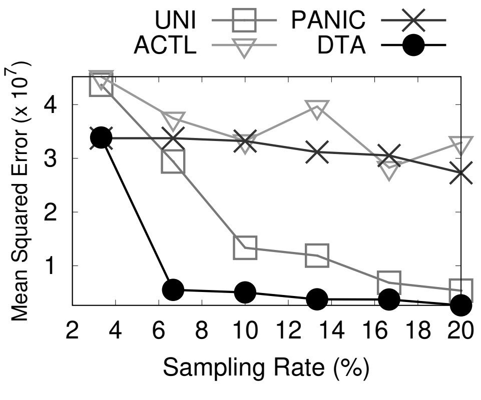
(a) k-means



(b) Bayes



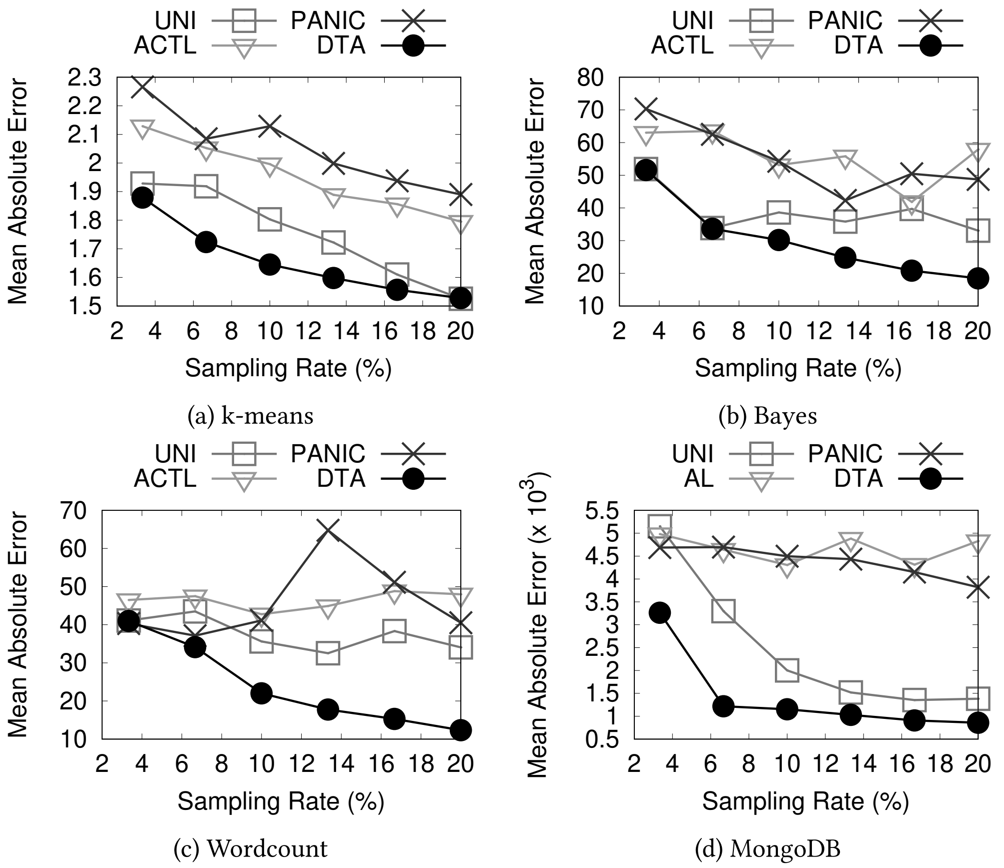
(c) Wordcount



(d) MongoDB

Figure 2.7: Accuracy vs sampling rate (MSE)

Figure 2.7 showcases that DTA outperforms all the competitors for increasing $SR$, something indicative of its ability to distribute the available number of deployments accordingly so as to maximize the modeling accuracy. In more detail, all algorithms benefit from an increase in $SR$ since the error metrics rapidly degrade. Both in k-means and Bayes, when the $SR$ is around 3% UNI and DTA construct models of the highest accuracy. As mentioned in Section 2.3.4, for



(a) k-means

(b) Bayes

(c) Wordcount

(d) MongoDB

Figure 2.8: Accuracy vs sampling rate (MAE)

such low $SR$ the linearity of the DT would fail to accurately represent the relationship between the input and the output dimensions, thus a Random Committee classifier based on Multi-Layer Perceptrons is utilized for the approximation. The same type of classifier also achieves the highest accuracy for the rest of the profiling algorithms (ACTL, PANIC) that also present higher errors due to the less accurate sampling policy at low SR. As $SR$ increases, the DT obtains more samples and creates more leaves, which contributes in the creation of more linear models that capture a shorter region of the Deployment Space and, thus, producing higher accuracy. Specifically, for $SR \geq 3\%$, DT created a more accurate prediction than other classifiers and was preferred.

In the rest of the cases, DTA outperforms its competitors for the Wordcount application and, interestingly, this is intensified for increasing $SR$. Specifically, DTA manages to present $3\times$ less modeling error than UNI when $SR = 20\%$. Finally, for the MongoDB case, DTA outperforms the competitors increasingly with SR. In almost all cases, DTA outperforms its competitors and creates models even 3 times more accurate (for Bayes when $SR = 20\%$) from the best competitor. As an endnote, the oscillations in PANIC's and ACTL's behavior are explained by the aggressive exploitation policy they implement. PANIC does not explore the Deployment Space and only follows the steep regions, whereas ACTL retains a similar policy only following the regions of uncertainty, hence the final models may become overfitted in some regions and fail to capture most patterns of the performance function. Our work identifies the necessity of both exploiting the regions of uncertainty but also for exploring the entire space. This trade-off is only addressed by DTA and explains its dominance for difficult to approximate applications.

The respective plots depicted in Figure 2.8, where the error is expressed in terms of Mean Absolute Error, provide a similar picture as the ones in Figure 2.7. DTA is, again, showcased to be the most efficient profiling methodology and the accuracies achieved by the different approaches are equivalent to the ones discussed previously. However, one notable difference with the plots of Figure 2.7, is that the methodologies with the highest errors (e.g., UNI for Wordcount, PANIC for Bayes) tend to present lower errors when measured with MAE instead of MSE. Taking into consideration that MSE is heavily weighted by outliers [BC01], we are lead to the conclusion that the less accurate methods present highly misclassified deployment configurations, i.e., poorly approximated outliers. On the contrary, DTA presents similar behavior for both error metrics. This is indicative of DTA's ability to avoid strongly outlying points, i.e., points for which the error is too large. The inherent divide-and-conquer functionality of DTA and its ability to focus on each region of the Deployment Space separately and, thus, avoid strong outliers.

## 2.4.2 Impact of performance function complexity and dimensionality

We now compare the accuracy of the profiling algorithms against synthetic profiling functions with varying complexity and dimensionality. We create synthetic performance functions using the ones presented in Table 2.2 for 2 and 10 dimensions, employing 2 different $SR$ (0.5% and 2%). Specifically, we assume that there exist 2 dimensions that highly impact the function's output and, in the 10-d case, the remainder dimensions are of much lower significance, i.e., their coefficients tend to zero. This simulates the real-world use case where one wants to profile an application with many dimensions, but only a handful of them are, indeed, significant to the output performance. The results are depicted in Table 2.3. Since the output of each dataset is in different scale, we normalize all the results, dividing the error of each methodology with the error produced by UNI. All methodologies approximated LIN, POLY, EXP and EXPSQ with minimal error and hence are not provided. The lowest errors for each case are demonstrated in bold.

Table 2.3 showcases that *all* the synthetic functions were more accurately approximated by DTA than the rest of the profiling schemes. Specifically, when the dimensionality of the space is low, DTA achieves considerably lower modeling errors than the UNI case, whereas ACTL and PANIC produce much worse results, compared to UNI, something intensified with increasing $SR$. This is ascribed to the fact that ACTL and PANIC only exploit the space abnormalities

Table 2.3: Accuracy vs function complexity

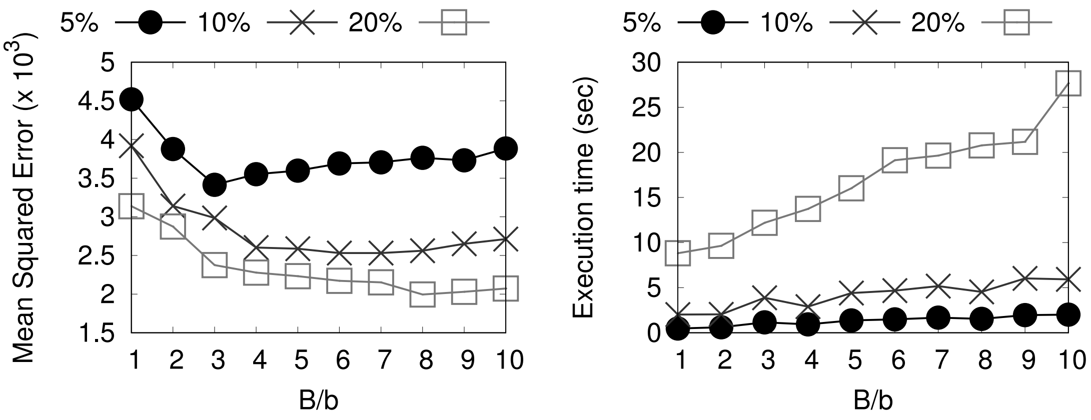
| | **n** | **ACTL** | | **PANIC** | | **DTA** | |
|---|---|---|---|---|---|---|---|
| | | 0.5% | 2% | 0.5% | 2% | 0.5% | 2% |
| EXPABS | 2 | 1.99 | 5.29 | 1.29 | 2.63 | **0.58** | **0.52** |
| | 10 | 0.40 | 0.31 | 0.33 | 0.28 | **0.13** | **0.11** |
| GAUSS | 2 | 1.55 | 5.91 | 1.32 | 5.63 | **0.68** | **0.52** |
| | 10 | **0.42** | 0.39 | 0.45 | 0.42 | **0.42** | **0.36** |
| WAVE | 2 | 0.98 | 2.34 | 1.01 | 2.51 | **0.71** | **0.79** |
| | 10 | 0.30 | 0.28 | 0.31 | 0.28 | **0.18** | **0.15** |
| HAT | 2 | 1.17 | 6.49 | 1.25 | 4.85 | **0.60** | **0.42** |
| | 10 | 0.52 | 0.43 | 0.49 | 0.45 | **0.32** | **0.29** |

whereas UNI focuses on space exploration. In cases where the space consists of 2 dimensions with high impact, UNI is, understandably, better than the exploitation-centric ACTL and PANIC approaches, DTA standing in the middle and producing the best results.

Interestingly, though, when the dimensionality of the space increases, UNI becomes insufficient, as in these cases all the other approaches exhibited a considerably more accurate behavior. The spaces now consist of 10 dimensions, the 8 of which are of low importance: All the approaches that focus on the abnormalities of the space (ACTL and PANIC) benefit from their exploitation-based functionality since they ignore deployment space regions with uninteresting, i.e., non oscillating, behavior. DTA, on the other hand, achieves the same results, but for another reason: The construction of the DT during the sampling and modeling phases, is constantly ignoring the unimportant dimensions, executing a form of *Dimensionality Reduction* to the deployment space and only focusing on the dimensions of higher interest. This functionality renders our approach suitable both for cases where there exist several non-interesting dimensions and need to be ignored and cases where there exist equally important dimensions that should be evaluated. This discussion highlights that DTA manages to elegantly compromise the contradicting aspects of exploration and exploitation. Furthermore, the recursive Deployment Space partitioning manages not only to distribute the deployment budget appropriately, but also ignore the dimensions that present no interest.

### 2.4.3 Impact of per-iteration number of deployments

We now evaluate $b$'s impact, i.e., the number of deployments spent in each algorithm iteration, in DTA's performance. In Figure 2.9, we provide results where $b$ is set as a portion of $B$ (the total number of allowed deployments) for three different $SR$. The horizontal axis represents the ratio $B/b$. The Figure on the left depicts the MSE while the right one represents the respective execution time of DTA.



(a) Accuracy (b) Execution time

Figure 2.9: Per-iteration number of deployments evaluation

When $B/b = 1$, the algorithm degenerates into UNI, since the tree is constructed in one step. At this point, the algorithm presents the highest error and the lowest execution time, since the tree is only constructed once and the most erroneous leaves are not prioritized. When the ratio increases, the algorithm produces more accurate results and the error decrease intensifies for increasing $SR$. For example, when $SR = 20\%$ the error decreases more than 35% for increasing $B/b$. However, when $SR = 5\%$ and $SR = 10\%$ and for low $b$ values (e.g., $B/b = 10$) an interesting pattern appears: MSE starts to increase, neutralizing the effect of lower $b$. This occurs in cases where $b$ is extremely small, compared to the dimensionality of the Deployment Space. In such cases, the per-iteration budget becomes too small in order to be efficiently distributed among the DT leaves, and hence, many iterations need to take place before the tree is further expanded and analyzed. This leads to a suboptimal distribution of the $b$ samples and, finally, a suboptimal configuration selection policy.

The performance gain presented by increasing $B/b$ is ascribed to two factors: (a) the final DT has more accurate cuts and, hence, well-placed models and (b) the samples are properly picked during the profiling. To isolate the impact of each factor, we repeat the same experiment for all applications and train different ML classifiers instead of a DT. This way, we isolate the impact of sampling and only examine its importance. We identified that the classifiers that consist of linear models, i.e., regression classifiers such as OLS (Ordinary Least Squares) [DL06], or an Ensemble of Classifiers created with Bagging [Qui96] (BAG) that utilize linear models as base improve their accuracy with increasing $B/b$. In Table 2.4 we provide our findings for two classifiers (OLS and BAG), expressing the percentage decrease of MSE of each case, compared to the $B/b = 1$ case.

The table demonstrates that for all cases, increasing $B/b$ benefits the linear models, something that showcases that our sampling algorithm itself achieves better focus on the interesting

Table 2.4: MSE decrease % for linear classifiers

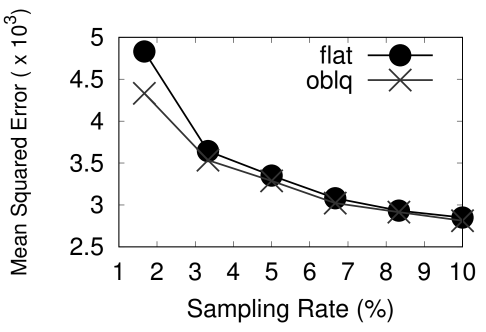
| **Application** | **Classifier** | **B/b** | | | |
|---|---|---|---|---|---|
| | | **2** | **5** | **8** | **10** |
| k-means | OLS | 8% | 10% | 12% | 12% |
| | BAG | 3% | 8% | 8% | 9% |
| Bayes | OLS | 23% | 27% | 28% | 29% |
| | BAG | 23% | 22% | 21% | 23% |
| Wordcount | OLS | 23% | 17% | 19% | 23% |
| | BAG | 21% | 28% | 41% | 38% |
| MongoDB | OLS | 19% | 13% | 12% | 7% |
| | BAG | 6% | 12% | 11% | 2% |

Deployment Space regions. Nevertheless, we notice that an increasing $B/b$ ratio does not always lead to linear error reduction. When $B/b = 8$ and $B/b = 10$, one can notice that the error either degrades marginally (for k-means and Bayes) or slightly increases (for MongoDB) when compared to $B/b = 5$. This is, again, ascribed to the extremely low per-iteration number of deployments $b$ that almost equals the dimensionality of the space. In summary, our findings demonstrate that even utilizing solely the sampling part of our methodology can be particularly useful in cases where linear models, or a composition of them, are employed.

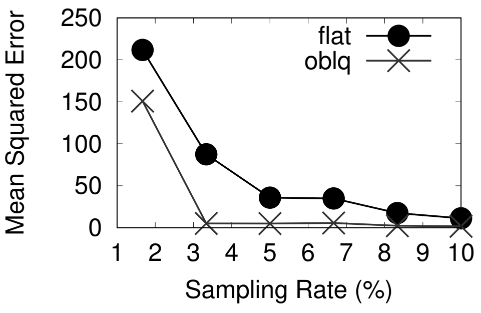
### 2.4.4 Impact of oblique boundaries

We now evaluate the impact of flat versus oblique DTs in the profiling accuracy and execution time. In Figure 2.10 (a), we model k-means for varying $SR$. The oblique tree introduces a slight accuracy gain (of about 10% for low $SR$) compared to the flat tree and demands about twice the time for algorithm execution. The accuracy gains fade out when the $SR$ increases.

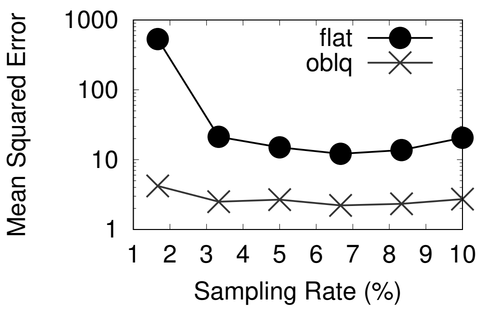
This is owed to the fact that when a higher $SR$ is employed, more leaves are created and the profiling algorithm functions in a more fine-grained manner. At this point, the structure of the leaf nodes is not important. However, when the algorithm must work with fewer leaves, i.e., lower $SR$, the importance of the leaf shape becomes crucial, hence the performance boost of the oblique cuts. However, a point not stressed in the results so far is that oblique DTs, apart from a minor performance boost, offer the ability to accurately approximate points of the performance function with patterns spanning to multiple dimensions of the Deployment Space. When measuring the accuracy of the model in the entire space, as in Figure 2.10 (a), this effect can be overlooked, yet, it appears when focusing on the region containing the pattern. To showcase this, we conduct the following experiment: Assume EXPABS of Table 2.2, minimized when $a_1x_1+\cdots+a_nx_n = 0$. We again compare the flat and oblique approaches for a two-dimensional Deployment Space but now use two different test sets: (i) points from the entire space (as before) depicted in Figure 2.10 (b) and (ii) points close to the aforementioned line (less than $\epsilon = 10^{-3}$) in Figure 2.10 (c).



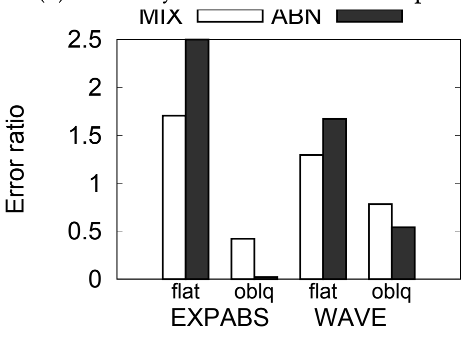
(a) Accuracy for k-means



(b) Accuracy for EXPABS - entire space



(c) Accuracy for EXPABS - ABN space



(d) Relative error for different test sets

Figure 2.10: Accuracy for flat and oblique cuts

While the difference in (b) is considerable for this case (a gain of 30% to 90%), when testing against points close to the abnormality we observe that the oblique version manages to achieve error rates that are orders of magnitude lower than those achieved by the flat DT. To further evaluate the impact of the test set into the measured accuracy for different performance functions, we repeat the previous experiment for EXPABS and WAVE (that also contains similar complex patterns) for a $SR = 2\%$. We measure the accuracy of the model when the test points are picked from (a) the entire Deployment Space (ALL), (b) close to the "abnormal" region (ABN) and (c) both (MIX). For MIX, half of the test points are far from the abnormality and the other half is located in a distance less than $\epsilon$. Figure 2.10 (d) showcases the respective results. The produced errors for MIX and ABN are divided with the error of ALL for each execution. Our results demonstrate that when more points are picked around the abnormality region, the flat DT produce higher modeling errors and oblique DT achieve much lower errors. A similar to the flat DT behavior is also verified for the rest of the profiling algorithms as well (UNI, ACTL and PANIC): None of them was able to approximate the abnormal regions of the synthetic functions with satisfying accuracy, rendering DTA the only profiling methodology with this feature.

### 2.4.5 Cost-aware profiling

So far, we have assumed that all Deployment Space points are equivalent, in the sense that we have not examined the deployment configuration they represent: A point representing a deployment configuration of 8 VMs, each of which has 8 cores, is equivalent to a point representing 1 VM with 1 core. However, since the choice of a point results in its actual deployment, it is obvious that this choice implicitly includes a (monetary) cost dimension that has not been addressed. We now examine DTA's ability to adapt when such a cost consideration exists. Let us define the following cost models:

- k-means: $|\text{nodes}| \times |\text{cores}|$
- Bayes: $|\text{nodes}| \times |\text{cores}|$
- Wordcount: $|\text{nodes}| \times |\text{cores}|$
- MongoDB: |MongoS| + |MongoD|

We have chosen realistic cost models, expressed as functions of the allocated resources (VMs and cores). Let us recall that MongoDB utilizes unicore VMs. Hence, their cost is only proportional to the number of allocated VMs.

We execute DTA using the Equation 2.2, presented in Section 2.3.3 where the error and size parameters positively influence the score of each leaf and cost is a negative factor. For $w_{error} = 1.0$ and $w_{cost} = 0.5$, we alter the weight of the cost parameter between $0.2$ and $1.0$ for SR of $3\%$ and $20\%$. We provide our findings in Table 2.5, in which we present the percentage difference in the profiling error (measured in MSE) and cost for each case, against the case of $w_{cost} = 0.0$.

Table 2.5: MSE and Cost for different cost weights

| **App/tions** | **SR** | **MSE** | | | **Cost** | | |
|---|---|---|---|---|---|---|---|
| | | **0.2** | **0.5** | **1.0** | **0.2** | **0.5** | **1.0** |
| **k-means** | 3% | -1% | -3% | +3% | -1% | -5% | -6% |
| | 20% | -6% | -2% | -8% | -7% | -11% | -26% |
| **Bayes** | 3% | +1% | -1% | -2% | -1% | -1% | -1% |
| | 20% | +4% | +10% | +5% | -7% | -9% | -12% |
| **Wordcount** | 3% | -1% | -5% | -1% | -4% | -6% | -1% |
| | 20% | 0% | +13% | +19% | -6% | -8% | -18% |
| **MongoDB** | 3% | +6% | +12% | +13% | -2% | -3% | -4% |
| | 20% | -1% | -7% | -7% | -6% | -9% | -12% |

For low $SR$ , increasing values for $w_{cost}$ does not heavily influence the profiling cost. Specifically, for the MongoDB case, the cost reduces by a marginal factor (around $4\%$ in the most extreme case) whereas the error increases by $13\%$. On the contrary, for high $SR$, the consideration of the cost increases its impact as the application profiles are calculated even $26\%$ less expensive than the case of $w_{cost} = 0$, e.g., in the k-means case for $w_{cost} = 1.0$. Furthermore, it is obvious that for increasing $w_{cost}$, the cost becomes a more important factor for the leaf score and, hence, the cost degradation becomes more intense. Regarding the profiling accuracy, in most cases the error remains the same or its increase does not exceed 10%. A notable exception from this is the Wordcount case, where we can see that the MSE increases rapidly with increasing $w_{cost}$ values and even reaches a growth of 19% in the case where $w_{cost} = 1.0$. From this analysis, we can conclude that cost-aware sampling becomes particularly effective for high $SR$, which is also desirable since high $SR$ entail many deployments, i.e., increased cost. In such cases, the cost-aware algorithm has more room to improve the profiling cost whereas, on the same time, the accuracy sacrifice is totally dependent on the nature of the performance function; However, from our evaluation we can conclude that the accuracy degradation is analogous to the cost reduction, allowing the user to choose between higher accuracy or reduced deployment cost.

## 2.5 Discussion

Let us now discuss in more detail some delicate aspects of our methodology. First, from a high-level, Algorithm 1 attempts to compromise two contradicting aspects regarding knowledge extraction from an unknown space: *Exploration* and *exploitation*. There exist many research fields that face the same challenge. Reinforcement Learning [SB98], in particular, presents the same dilemma during the training phase of an agent that has to select among different actions, each of which has a certain penalty or reward. In the domain of performance modeling, though, the research efforts were mainly focused either solely on exploring the space, i.e., adopting Uniform or similar space covering sampling algorithms, or exploiting the obtained knowledge. Methodologies of the latter type are mainly encountered in domains were the global maximum/minimum of a performance function is required. To the best of our knowledge, our work is the first that manages to both *explore* and *exploit* the extracted knowledge in an efficient way, also presenting superior results than other, competitive approaches. The open nature of the proposed solution, that is serialized in a simple expression (i.e., Equation 2.2), leaves great room for extensions; In Section 2.4 we already demonstrated how expressions regarding the deployment cost may be accommodated. Similarly, one could easily consider alternative factors such as deployment *speed*, peculiarities of specific configurations (e.g., assign higher scores to uni-core configurations), etc.

Second, our work is based on the assumption that the deployment of a given application configuration (i.e., a point of the Deployment Space) is deterministic and always produces the same performance value. This is a common assumption behind all "black-box" modeling techniques that guarantees that a performance function has all these properties that make it able to be approximated by a statistical method. In practice, though, the reproducibility of the deployments that run on real-world cloud infrastructures, are not guaranteed. Interference caused by virtualization and the shared nature of the cloud resources, may introduce noise to the performance values and distort the function under approximation. Even though this problem is outside the scope of out work, one can overcome the introduced noise with different techniques: Running deployment for the same configurations multiple times and use the mean performance value as a point, running a sample of the selected configurations in order to verify the correctness of the obtained performance values, or even utilizing a more sophisticated model with higher tolerance to noisy data as [FS99], could help overcoming this challenge. However, as cloud platforms develop and mature, they achieve higher level of isolation and utilize more efficient scheduling algorithms that minimize such interference.

Finally, one aspect that was not discussed so far, relates to the practicality of the introduced methodology, from the cloud deployment point of view. Our analysis so far was mainly focused on explaining the partitioning, sampling and modeling algorithms, overlooking the procedure through which a selected configuration is deployed to the cloud. Even though this procedure does not present any theoretical challenges, it requires multiple systems to coordinate in an orderly fashion in order to generate a new application instance, run a given workload to it and collect the performance value. In practice, we observed that a non negligible percentage of the attempted deployments ended up in an inconsistent state. The main reasons behind these failed attempts were transient errors, timeouts to the underlying services, bad synchronization between the cloud services, etc. Correcting such errors proved to be highly impractical, especially because it required manual intervention to fix a partially executed workflow and correcting it in place, or cleaning everything and starting over.

Motivated by this observation, we developed *AURA*, a cloud deployment system with error-recovery capabilities. An application deployment is expressed as a Directed Acyclic Graph the edges of which represent the discrete deployment steps while the nodes represent the intermediate states that different application modules reach during the deployment. If a transient error occurs during the application deployment, AURA attempts to overcome it through re-executing the failed deployment parts. In order to ensure idempotence of the deployment scripts, it employs a file system snapshot mechanisms, that guarantees that a failed script execution leaves no side-effects to file system related resources (configuration files, binaries, etc.). A detailed presentation of AURA can be found in Appendix A.

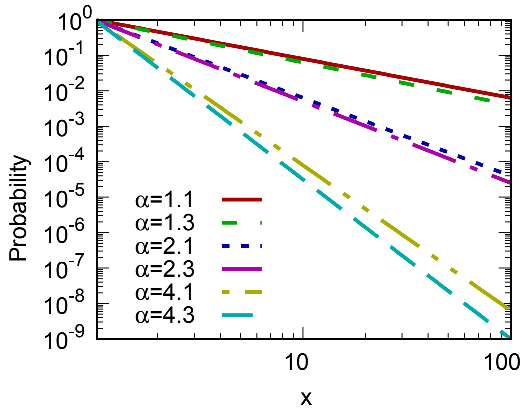
CHAPTER 3

# A Content-Based Approach for Modeling Analytics Operators

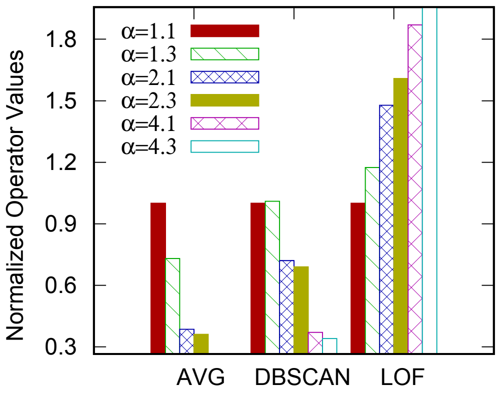
This chapter describes a content-based approach for modeling analytics operators. The methodology is based on the idea that datasets that have similar characteristics influence the operators that are applied to them in similar ways and, hence, their outputs are should be similar. Instead of focusing on operators, this work takes a different view on the problem: It attempts to quantify the similarity of different datasets, in the light of a handful of fundamental properties and constructs a *Dataset space* that represents this knowledge. A given operator is, subsequently, applied to a few of the available datasets and the outcome of the operator for the remainder, non-tested datasets is approximated using Neural Networks. Our evaluation, conducted for a variety of real-world datasets that include monitoring metrics, weather data and time-series with stock prices, indicate that our approach models the outcome of different operators with remarkable accuracy and massively accelerates data analysis by a factor of up to $20\times$. This novel idea is also showcased to be successfully applied to graph-based datasets as well, indicating that the suggested scheme is generally applicable regardless of the data representation and it is modular in the sense that it can easily adapt to different domains, when different dataset similarity metrics are employed.

## 3.1 Overview

To introduce our approach, let us provide an illustrative example that showcases that there exists a firm relationship between one of the most commonly studied data properties, i.e., data *distribution* and the behavior of three popular data operators. Assume six 1-d datasets that follow the Pareto distribution with different $\alpha$ values (i.e., exponents) and three operators: *AVG*, that provides the mean value of each dataset, *DBSCAN* [EK+96] that executes the popular clustering algorithm and return the number of formed clusters and *Local Outlier Factor* (referred to as *LOF*) that executes the algorithm presented in [BKNS00] and returns the percentage of tuples that collect a score greater than 2 and are, thus, considered outliers. Figure 3.1a provides the probability density function of each dataset in a log-log plot and Figure 3.1b provides the operators' values for each of the three operators, normalized with the value obtained for the dataset where $\alpha = 1.1$. There exists a clear monotonic relationship between $\alpha$ and the produced operator values: The bigger the exponent the smaller the score for the *AVG* and the *DBSCAN* operators and the larger for Local Outlier Factor. As increasing exponents contribute to the creation of datasets with narrower ranges and values closer to 0, the behavior of all three operators is highly correlated to $\alpha$ which controls the dataset distribution. In all three operators, the datasets with high distribution similarity tend to present similar values for each operator. As a consequence, one could *predict* the behavior of each dataset, only by examining the behavior of its most similar one. For example, the values of the operators for $\alpha$=4.3 highly resemble the $\alpha$=4.1 case.



(a) Statistical distribution of 6 datasets



(b) Output of operators for the 6 datasets

Figure 3.1: Relationship between Data Distribution and Operator behavior

The above example demonstrates that the examination of the similarity of such a fundamental property such as the statistical distribution can often provide invaluable hints for predicting the output of different operators. This is a reasonable finding: All the operators provided in the above example are affected by the statistical distribution of the underlying datasets and hence when the distributions between two datasets are similar, the respective operator output is anticipated to be similar as well. If we want to generalize this finding, we should ask: Which dataset properties should be used for measuring the similarity? And how can this similarity information be utilized in order to *predict* the outcome of a given operator when applied to different datasets?

In order to answer the above questions, we propose a content-based methodology for modeling analytics operators. It relies on quantifying the similarity of a set of datasets in the light of different properties, encoding this information to a low-dimensional dataset space that reflects the similarity between them, executing a given operator for a mere subset of the available datasets, and, finally, approximating the operator's output for the untested datasets using Machine Learning. Our methodology is generic, in the sense that different similarity metrics can be accommodated without requiring any changes to the codebase. Nevertheless, our study is focused on three key dataset properties that influence many operators. These properties are the *statistical distribution* of the datasets, the *order* of their tuples and their *size*, i.e., the number of tuples they consist of. Using these three key properties, we are able to create informative dataset spaces that, as our evaluation demonstrates, may be used as input spaces by Neural Networks in order to model the behavior of a set of operators with diverse characteristics. Interestingly enough, our methodology seems to be applicable not only to tabular data, i.e., datasets that comprise tuples, but also to graph-based data as well.

The contributions of our work are summarized as follows:

- We present an efficient, operator-agnostic dataset profiling methodology, that estimates the similarity among the available datasets, constructing a *dataset space* that best reflects their properties and modeling the output of the applied operators utilizing Neural Networks trained using this dataset space as input.

- We offer an open source Go prototype [sou18] of our work, through which a user can execute the dataset profiling and export the generated ML models for integration with other systems.

- We conduct a thorough evaluation of this approach, utilizing a variety of datasets with different characteristics. models the behavior of a wide variety of operators with remarkable accuracy (less than 2% of modeling error in the best case and less than 15% in most cases when a mere 4% of the datasets are examined). Moreover, it presents massive speedups (more than 20× in the best case) in comparison to exhaustively executing the operators for the entirety of available datasets. Finally, it can be customized in order to accelerate data analysis and conduct less detailed dataset examination or increase modeling accuracy when higher execution time is affordable.

## 3.2 Preliminaries

### 3.2.1 Problem Description

Assume a set of datasets $D = \{D_1, D_2, \cdots, D_N\}$ and an operator $F$. Each dataset $D_i, 1 \leq i \leq N$, consists of tuples with the same number of columns, containing arithmetic values. $F$ accepts a single dataset as input and produces a scalar output value:

$$F : D \rightarrow \mathbb{R} \tag{3.1}$$

Each operator can be viewed as a function that projects any dataset $D_i$ to a scalar value $F(D_i)$. The problem that this work addresses, is the following: *We seek for an approximation of the expression $F(D_i)$ without exhaustively executing F for all datasets.* This resembles a typical function approximation problem: One can sample $D$, execute $F$ for the selected subset of datasets and utilize regression to approximate $F$ for the rest of the datasets. However, this method cannot be applied in this problem because $D$ represents an unordered set of datasets that do not belong to a *metric space* and the relationships between them are unknown. Since all function approximation approaches require $D$ to be a metric space, i.e., the distances between datasets to be known, regression cannot be used.

Albeit constructing a metric space for any given $D$ is possible for a given distance function for each dataset pair in $D$, the quality of the approximation is heavily affected by the choice of this function. Ideally, the desired distance function must reflect the distance between two datasets $D_i$, $D_j$, $1 \leq i, j \leq N$, both in the aforementioned metric space and in the operator's output domain, i.e., if $|D_i - D_j| < \epsilon$ ($|.|$ denoting the Euclidean norm) then $|F(D_i) - F(D_j)| < \epsilon$ as well. If this property applies, the constructed metric space can be accurately used by a model in order to *predict* an operator's output, since the topology of the datasets themselves provide excellent hints on the behavior of the latter. Although this may initially seem an operator-dependent procedure, we argue that only a handful of distance functions that examine specific dataset properties suffice to generate highly informative *dataset spaces*, that facilitate modeling the behavior of diverse real-world operators.

### 3.2.2 Operators and Dataset Properties

Associating how a property affects an operator's behavior generally requires extensive knowledge regarding the operator's design. Nevertheless, we argue that there exist some *fundamental* properties that, if examined, they can produce invaluable insight regarding an operator's behavior. Indeed, examining data interrelationships in the light of a handful of fundamental dataset properties can generate a knowledge basis through which the behavior of different operators can not only be explained but also predicted. These properties examined in this work are: (a) the statistic al *distribution* of the datasets, (b) the dataset *size* and (c) the *order* of their tuples. *Distribution* refers to the positioning of a dataset's tuples and it is a fundamental property that is implicitly or explicitly examined during any data analysis task, as it uniquely characterizes the statistical behavior of a dataset. *Size* is an expression of the cardinality of the dataset and it is commonly examined when the behavior of an operator is affected by it. *Order* expresses the ranking of the dataset's tuples and is frequently examined when sequence matters.

Table 3.1: Operators and Dataset Properties

| **Operators** | | **Affected by** |
|---|---|---|
| **Class** | **Name** | |
| Aggregate Functions | AVG | Distribution |
| | SUM<br>COUNT | Distribution + Size |
| Density | DBSCAN [EK$^+$96]<br>Local Outlier Factor [BKNS00] | Distribution |
| ML | Linear Regression | Distribution |
| Spectrum | Eigenvalue Estimation | Distribution |
| Time-Series Forecast | Holt-Winters [Cha78]<br>ARIMA [BJRL15] | Distribution + Order |

These three fundamental properties highly affect a magnitude of real-world operators. In order to showcase this, in this work, we use popular operators from diverse domains, which are frequently encountered either as isolated components or as part of greater and more complex analytics workflows. The considered operators along with the respective properties they are affected by are summarized in Table 3.1. Each operator is formulated as per Equation 3.1. Specifically, the *Aggregate Functions* represent mathematical operations that are applied to one or more dataset columns, producing a scalar value. The *Density* class comprise *DBSCAN* [EK$^+$96] that executes the popular clustering algorithm and return the number of formed clusters and *Local Outlier Factor* (referred to as *LOF*) that executes the algorithm presented in [BKNS00] and returns the percentage of tuples that collect a score greater than 2 and are, thus, considered to be outliers. The *ML* class consists of a fundamental Machine Learning operator, i.e., Linear Regression. This operator is trained using a dataset from $D$ and return an estimate of the training error when the model is tested with a given external dataset. The *Spectrum* class contains an operator that returns the $i$-th eigenvalue (for a given $i$) of the provided dataset, a procedure executed in various workflows from dimensionality reduction [Jol86] to clustering. Finally, the *Time-Series Forecast* class comprises two operators [Cha78, BJRL15] that forecast the $i$-th value of a provided Time-Series dataset.

The reason behind the choice of examining these operators is threefold. First, all of these operators are *popular* and extensively utilized in Data Science and Machine Learning applications, either as part of data preprocessing (e.g., outlier detection, statistical analysis, etc.) or core learning workflows (e.g., supervised/unsupervised learning). Second, their *diverse* characteristics enforce us to design a generic solution that makes no assumptions regarding their internals. Third, Machine Learning workflows, parts of which are frequently constructed with the above operators, are an *excellent domain* for the problem this work addresses. As data scientists need to be able to classify an increasing number of datasets [BGRS17] without actually executing their complex workflows to them, the identification of the *Right Data* [BY13], i.e., data of high utility which are essential for driving strategic decisions, is crucial. Finally, it should not be overlooked that Table 3.1 does not contain a complete list of operators. Although our work uses the mentioned operators for evaluation, our methodology can be used for any operator affected by the three mentioned data properties without modifications.

## 3.3 Methodology

### 3.3.1 Methodology Overview

The key observation that datasets with similar specific properties impact certain operators in similar ways and, hence, make them producing similar outputs, highlights a new dimension to the problem under investigation: If one quantifies the similarity between all pairs of datasets and executes an operator for only a handful of them, a first idea of $F$'s domain would become available, as datasets with high similarity would present similar behavior. Let us generalize this idea: Given the relationship between dataset similarity and an operator's output, we seek for a projection of the datasets in $D$ into a metric space $D'$ (also referred to as *dataset space*) that best reflects the resemblance among them. $D'$ can be then utilized by $F$ as the domain space – according to Equation (3.1) – in order to project the original datasets into the anticipated values. Interestingly so, the relationships between datasets are *independent* of $F$, allowing different operators to be applied to a unique $D'$. For each operator, one could sample $D$, estimate $F$'s values for the selected datasets $D_i \in D_s \subseteq D$ and approximate $F$ for the rest of the datasets utilizing Machine Learning (ML) techniques. Although $F$ is applied to some of the original datasets, i.e., $F(D_i), D_i \in Ds$ is calculated, the ML model is trained using $D'$ as the input space and the approximated operator $F'$ is defined as: $F' : D' \rightarrow \mathbb{R}$. Essentially, $D'$ comprises a set of *features* that best characterize the datasets' interrelationships. Figure 3.2 depicts an overview of the suggested methodology.

The *Similarity Estimation* module quantifies the similarities between datasets $D_1, \cdots, D_N$. The outcome of this process is a symmetrical $N \times N$ similarity matrix whose ($i$,$j$) cell represents

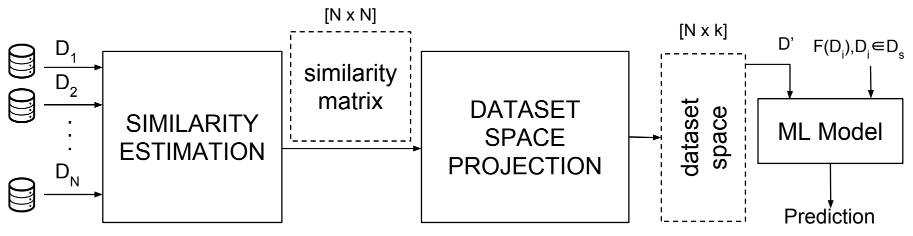


Figure 3.2: Methodology workflow

the similarity between $D_i$ and $D_j$. The similarity matrix is then accessed by the *Dataset Space Projection* module which transforms the original similarities into a metric space. In this step, Multidimensional Scaling (MDS) [Gow66] transforms the similarity matrix into a set of points in a low-dimensional ($k$-dimensional) space, with the property that the distances between the points of the space approximate the similarity represented by the original matrix. The final outcome of the process is a $N \times k$ matrix that represents the coordinates of each dataset in the dataset space. Finally, an operator $F$ can be executed for a subset of datasets $D_s$. Using the dataset coordinates and the respective operator values, a Neural Network is trained in order to approximate $F$ for all datasets. Based on the approximated dataset scores, interesting questions can be answered: Which are the dataset(s) with the highest/lowest $F$ values (e.g., with the highest first eigenvalue), how many datasets' output is around a given $F$ value (e.g., dataset with approximately 5 DBSCAN clusters), retrieve the top-k datasets under certain output criteria (e.g., the top-10 datasets with highest percentage of outliers), etc.

Essentially, our approach attempts to shift the computational burden in the first phase of data analysis: The workflow presented in Figure 3.2 is executed once in an *offline* manner for all datasets. This offline part is entirely *operator-agnostic*: The similarity estimation does not imply the execution of any operator and only the relationships between the raw datasets are evaluated without considering the type of the operator that may be applied to them. Whenever a new operator emerges, it is executed for a mere subset of the available datasets and its behavior is rapidly approximated with minimal computation. In the long run, the overhead introduced by the suggested approach is amortized and the avoided computation linearly increases with the number of operators that need to be executed for the analyzed datasets.

### 3.3.2 Similarity Estimation

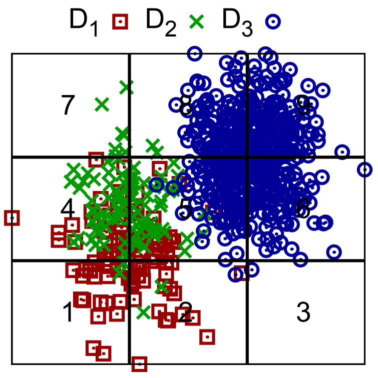
The notion of similarity adopted in this work focuses on three characteristics, namely the *distributions* of the datasets, their *size* and tuple *ordering*. Based on these primitive properties, one can compose arbitrary similarity expressions that efficiently express multiple dataset aspects. The similarity between two datasets is a real number in the interval $[0, 1]$, $0$ indicating total dissimilarity and $1$ indicating perfect similarity. We now examine the mechanism that quantifies the similarity for each property in detail.

▷**Distribution:** There exist several methods used to quantify the similarity of the distributions between two datasets. One of the most popular methods used during data analysis, entails the identification of the distribution in a closed form for each dataset separately and the comparison of the probability density functions (pdf) in order to extract the relationships between them. The extraction of a pdf usually occurs in a trial-and-error manner, as different distribution types are tested for each dataset, keeping the one that maximizes a statistical fitness measure such as the $p$-value. Although this analysis can provide great insight in the behavior of a statistical sample, it also presents certain limitations. First, the size of the sample plays a determinant role, as er datasets may not present enough evidence to accurately identify the most suitable pdf. Second, distribution-free datasets may be erroneously represented. Third, the number of tested distributions needs to be high in order to provide accurate data representations, increasing the complexity of the computations. Yet for our problem, we are not interested in the identification of each dataset's pdf per se, but we only want to estimate the extent at which the tuples of two datasets overlap, i.e., their relative positions in the space. If two datasets feature a large percentage of their tuples inside the same regions, then they should have similar distributions.

Quantifying the overlap between two different statistical samples (datasets) is the main idea behind the Bhattacharyya coefficient (BC) [CRM00]. Its estimation relies on partitioning two datasets into $l$ disjoint partitions, identifying the number of tuples that belong to each of them and, finally, summarizing the root of the product of the number of tuples for each region: $BC = \sum_{i=1}^{l} \sqrt{A_i B_i}$, where $A_i$ and $B_i$ denote the number of tuples located in the $i$-th partition for datasets $A$ and $B$ respectively. For two given datasets $A$ and $B$, the upper bound of the $BC$ value is obtained when $l = 1$. In order to compare $BC$ values between different pairs of datasets that might enumerate different tuples, we normalize $BC$ with this upper bound, reaching the following Distribution similarity function:

$$Distribution(A, B) = \frac{\sum_{i=1}^{l} \sqrt{A_i B_i}}{\sqrt{|A||B|}} \tag{3.2}$$

Let us now provide an example that highlights how Equation 3.2 works. Assume three datasets that comprise 2-d tuples and follow three distributions: $D_1 \sim Gauss((1,1),(1,1))$ with 100 tuples, $D_2 \sim Gauss((1,2),(1,1))$ also with 100 tuples and $D_3 \sim Gauss((4,3),(1,1))$ with 500 tuples. The left part of Figure 3.3 presents the position of the tuples of each dataset in the 2-d dataset domain. We apply grid partitioning and count the number of tuples each dataset presents in each partition, presented the right part of Figure 3.3. If we use Equation 3.2 for the computation of the similarity of each pair of datasets, we can see that $Distribution(D_1, D_2) = 0.83$, $Distribution(D_1, D_3) = 0.27$ and $Distribution(D_2, D_3) = 0.40$, whereas the denormalized $BC$ scores are: $BC(D_1, D_2) = 83$, $BC(D_1, D_3) = 61.1$ and $BC(D_2, D_3) = 89.3$. We observe that Equation 3.2 manages to successfully express that $D_1$ and $D_2$ are far more similarly distributed that $D_3$, something also visually evident from the dataset domain.



| $l$ | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|
| 1 | 21 | 2 | 0 |
| 2 | 23 | 2 | 1 |
| 3 | 0 | 0 | 1 |
| 4 | 30 | 43 | 0 |
| 5 | 26 | 44 | 122 |
| 6 | 0 | 0 | 117 |
| 7 | 0 | 7 | 0 |
| 8 | 0 | 2 | 108 |
| 9 | 0 | 0 | 151 |

Figure 3.3: Distribution Similarity Example

One parameter that highly affects Equation 3.2 is the partitioning setup. Specifically, both the partitioning algorithm and the number of partitions affect the equation's behavior, in different ways. The choice of the partitioning algorithm is crucial for ensuring fairness among different datasets, since certain partitioning schemes may boost specific distributions and be unfair to others. It also needs to be scalable to multiple dimensions, in order to successfully consider datasets of high dimensionality. Voronoi partitioning [Bow81] is a popular partitioning setup that adheres to these properties. Each partition (or *cell*) is represented by a centroid (seed). Each dataset tuple is assigned to the partition represented by its closest centroid. The partition problem is, thus, transformed to finding a set of seeds. Albeit the estimation of the optimal seeds is NP-Hard, one can generate them using k-means [LBG80]. Since k-means' solutions are largely affected by the initial seed selection, in our work k-means++ [AV07] was utilized. In order to maximize fairness between datasets and minimize any bias inserted by skewed distributions during the centroid selection, we sample all the available datasets, keeping a percentage (1 − 5%) of each and generate a new one that consists of all the sampled tuples. k-means is, then, executed for this "merged" dataset, centroids are extracted and then used for partitioning each dataset separately. The number of partitions ($l$) is determined by the user. More partitions lead to a more fine-grained examination whereas fewer partitions provide greater abstraction.

From a technical perspective, after the execution of k-means, the estimated Voronoi diagram is utilized to partition all the available datasets and, subsequently, the number of the tuples inside each leaf is estimated. For each dataset, an in-memory array is generated: The index of each array element represents the Voronoi cell id and the value of the array element represents the number of tuples that reside in the cell. The estimation of Equation 3.2 is then reduced to obtaining the dot product of the respective arrays for $A$ and $B$. Repeating the procedure for every dataset pair provides the final similarity matrix. Note that, the construction times of the arrays, which entails the processing of the raw data, increases linearly with the number of datasets. Essentially, these arrays act as synopses of the datasets: The more the partitions, the more accurate the representation and the more time-consuming is the estimation of the dot product between each pair. Specifically, the complexity of constructing a similarity matrix using Equation 3.2 equals $O(Nnld' + Nnl + N^2l)$, where $N$ is the number of datasets, $n$ is the maximum dataset size, $l$ is the number of partitions and $d'$ is the dimensionality of the domain of the datasets. The first factor refers to the Voronoi diagram creation, the second one refers to the synopsis creation for all datasets and the third refers to the matrix.

▷**Order:** The identification of the similarity between the ordering of two datasets entails the estimation of the rank correlation coefficient [Kru58], i.e., the measurement of the ranking between their members. First, a sorted copy of the datasets is created: The columns that are utilized for the sorting step are defined by the user, along with their importance in the comparisons (by default all columns are utilized in increasing column-id order), i.e., if $X[1] > Y[1]$ then $X > Y$. Based on the respective sorted copy for each dataset, the *rank* array is generated. The $i$-th element of the rank array represents the position of the $i$-th element in the sorted copy, e.g., if the original array contains the elements $X = [100, 99, 102]$ the respective rank array is $x = [2, 1, 3]$. Given that, we provide the rank similarity measure utilized in this work, that resembles the Kendall rank correlation coefficient ($\tau$) [Ken48]:

$$Order(A, B) = \frac{concord(a, b) - discord(a, b)}{n(n-1)} + \frac{1}{2} \tag{3.3}$$

in which, $a$ and $b$ represent the rank arrays of $A$ and $B$ respectively, $concord(a, b)$ returns the number of pairs $(a_i, b_i)$ and $(a_j, b_j)$, $i \neq j$ for which the rank of both elements agree, i.e., if $a_i > a_j$ then $b_i > b_j$ or if $a_i < a_j$ then $b_i < b_j$. $discord(a, b)$ returns the respective number of pairs whose rank disagree. When two pairs share the same rank they are considered neither concordant nor discordant. Note that the Kendall $\tau$ receives values in the interval $[-1, 1]$, in which $1$ means that the two ranks completely match and $-1$ means that the two datasets are sorted in reverse order. Since, in our case, the similarity metric is expressed in the interval $[0, 1]$, we scale the Kendall $\tau$ accordingly, producing the above expression. The complexity of constructing a similarity matrix using Equation 3.3 equals $O(Nn\log(n) + N^2 n\log(n))$. The first factor relates to the sorting step of all the datasets and the second factor relates to the estimation of $Order$ for each pair of datasets. Note that, although $Order$ seems to require the investigation of every pair of tuples of the two datasets (quadratic time to the dataset size, i.e., $O(n^2)$), a merge-sort based implementation requires $O(n \log n)$ steps [Kni66].

▷**Size:** The similarity between the size of two datasets is evaluated, using the following expression:

$$Size(A, B) = \frac{\min(|A|, |B|)}{\max(|A|, |B|)} \tag{3.4}$$

Note that, $Size(A, B) \to 1$, if $|A| \to |B|$. The complexity of creating a similarity matrix with Equation (3.4) is $O(Nn + N^2)$: The first factor represents the cost of counting each dataset's tuples and the second factor represents the evaluation of $Size$ for each pair of datasets.

▷**Combining different metrics:** While certain operators may depend on a single property, there exist cases one would want to construct a similarity matrix that combines several ones. In such cases, one can generate algebraic combinations of simpler matrices that reflect more sophisticated similarity expressions. The form of the algebraic expression (e.g., linear combination, matrix multiplication, etc.) that combines the matrices is defined by the user and expresses the importance of each component.

### 3.3.3 Dataset Space Projection

Although the information of a similarity matrix can be directly used by a data engineer, this representation of information is cumbersome for three reasons: (a) The *size* of the matrix increases quadratically with the number of datasets. (b) The matrix provides limited information as it does not represent the relationships at scale, e.g., one can easily identify a dissimilar – to the rest – dataset, but the magnitude of the dissimilarity is not easily comprehensible. (c) Most Machine Learning algorithms require the input metric space to be expressed in a form where the *coordinates* rather than the similarities of the input space points are known. Although a category of nonparametric learning algorithms [Alt92] do support regression based similarities, these algorithms are less sophisticated and are mainly used for simpler learning tasks. In order to address these limitations, our methodology transforms the similarity matrix into a dataset space where the positions of the datasets reflect their similarities: Datasets that are placed closer to each other would be more similar than the more distant ones.

Toward this direction, Multidimensional Scaling (*MDS*) [Gow66] has been used. MDS is a technique used to estimate the coordinates of a set of points given a square matrix that quantifies the dissimilarity between them. According to Classical MDS, the dissimilarity expresses the Euclidean distance between the points, although, in the general case, this dissimilarity can be expressed using any distance function. MDS can also work based on matrices that represent similarity. For clarity and in order to remain aligned to the literature, we will describe MDS based on dissimilarity matrices. Either way, similarity is easily transformed into distance using the following transformation ($s$ being the similarity and $d$ the distance):

$$s = \sqrt{1-d}, d = 1 - s^2 \tag{3.5}$$

where both $s, d \in [0, 1]$. Note that when $s \rightarrow 1$, $d \rightarrow 0$ and when $s \rightarrow 0$, $d \rightarrow 1$. Any pair of transformation equations that respect this property can be used instead, assuming that $d$ is finite.

There exist several methodologies that execute MDS in order to calculate a coordinates matrix. In this work, we utilize *Classical MDS* [Gow66] which expresses the problem as a matrix eigendecomposition problem. In short, MDS entails the execution of eigenanalysis to the dissimilarity matrix of size $N \times N$, producing a list of eigenvalues $[\lambda_1, \lambda_2, \cdots, \lambda_N]$ in descending order and their respective eigenvectors. Each eigenvector represents a dimension of the metric space and its respective eigenvalue represents the variance it captures. The number of eigenvectors to utilize is decided based on the covered variance, expressed by the *Goodness of Fit* (GoF):

$$GoF = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{N} \lambda_i} \tag{3.6}$$

$k$ being the number of eigenvectors to use. A common rule-of-thumb is to set $k$ to a value where $GoF \geq 0.75$ [Jol86].

Although MDS achieves to identify the dimensionality of the final space and the initial dataset positions to it, the difference between the distances obtained by the space and the matrix can be further reduced through a non-linear projection. To this end, after the execution of MDS, we apply Sammon mapping [Sam69], a non-linear space transformation that aims at slightly "moving" the projected datasets in such a manner that their projected distances best approximate the original dissimilarity matrix. Specifically, all the datasets are initially assigned with the coordinates produced through MDS. The distance between the projected and the target distances is expressed by the *Sammon Stress* $E_s$:

$$E_s = \frac{1}{\sum_{i<j} d_{ij}} \sum_{i<j} \frac{(d_{ij} - d^*_{ij})^2}{d_{ij}} \tag{3.7}$$

$d_{ij}$ denoting the distance between the $i$-th and $j$-th element from the distance matrix and $d^*_{ij}$ the respective distance as measured by the produced space. The execution of Sammon mapping entails the use of an iterative optimization methodology such as *Simulated Annealing* (SA) [VLA87], until the $E_s$ value stops declining or a pre-defined iteration threshold is exceeded. This reduction enables the constructed space to better represent the information of the similarity matrix, without increasing its dimensionality.

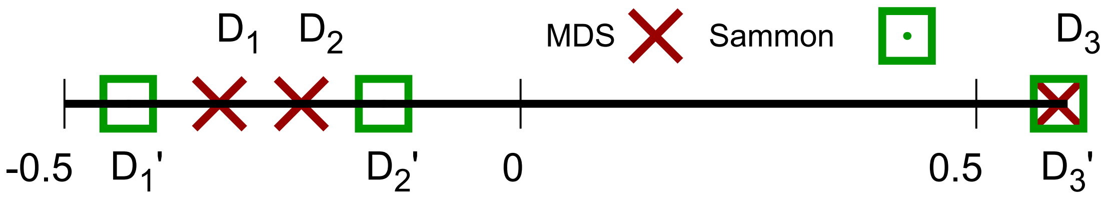


Figure 3.4: Example of MDS and Sammon mapping

Figure 3.4 depicts the constructed 1-d dataset space generated for the datasets depicted in Figure 3.3. After the execution of MDS for $k = 1$, $GoF$ was 0.92, therefore one dimension suffices in order to accurately represent the datasets at scale. After the execution of MDS, we applied Sammon mapping that slightly relocated $D_1$ and $D_2$ to $D'_1$ and $D'_2$ respectively in order to reduce $E_s$ from 0.071 to 0.011. This reduction enables the constructed space to better represent the information of the similarity matrix, without increasing its dimensionality.

### 3.3.4 Modeling

After constructing the dataset space, Neural Networks (NNs) are used for predicting a given operator's outputs. A sample $D_s \subseteq D$ of datasets is obtained, the size of which is determined by the user. An operator $F$ is applied to every dataset in $D_s$ and the output values are provided as a training set to a NN. The trained model is, subsequently, tested using a – disjoint from $D_s$ – test set. NNs were preferred against other Machine Learning methodologies due to their efficiency and their ability to model arbitrary distributions. The employed models comprise 1 hidden layer which is configured according to the rule of thumb that "the optimal size of the hidden layer is between the size of the input and size of the output layers" [Jef05]. Note that although model configuration greatly impacts its accuracy, this topic outside the scope of our work.

### 3.3.5 Optimizations

Having presented the key parts of our methodology, we now present two optimizations that aim at accelerating data analysis without sacrificing the quality of the computations.

#### Approximate Similarity Matrices

The methodology discussed so far entails the calculation of a squared matrix of size $N^2$, $N$ being the number of datasets, and the complexity equals $\mathcal{O}(N^2 x)$, in which $x$ represents the complexity of the employed similarity metric. For an increasing number of datasets, a quadratic complexity becomes prohibitive, as the computational effort required grows rapidly. A way of tackling this challenge is to avoid the calculation of similarities for all the distinct dataset pairs. However, this could lead to information loss, since the non-computed similarities should be replaced by values that approximate them, else this “approximate” similarity matrix may distort the dynamics of the space.

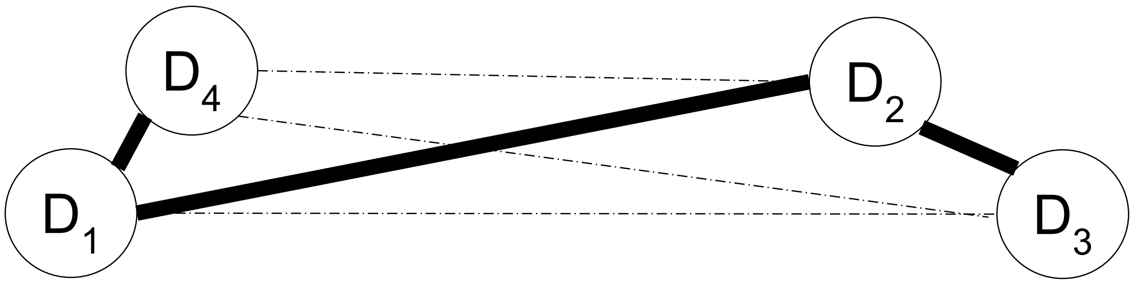


Figure 3.5: Dataset distances

In order to provide a solution, assume the datasets depicted in Figure 3.5 projected to a 2-d dataset space, in which the thin lines represent the distances among them. The distances between the datasets from the left and the right sides are much larger when compared to the distances of the datasets of the same side, e.g., $d_{2,3} < d_{1,2}$. Furthermore, assuming that only $d_{1,2}$ is known, one can say that

$$|d_{1,2} - d_{2,3}| \leq d_{1,3} \leq |d_{1,2} + d_{2,3}|$$

and if $d_{2,3} \ll d_{1,2}$ then $d_{1,3} \approx d_{1,2}$. In other words, in this example one only needs to calculate one of the “large” distances in order to avoid high approximation error. This interesting observation highlights the necessity of prioritizing for large distances when considering which of them should be evaluated. When such a “backbone” of distances is calculated, e.g., the set of thick lines of Figure 3.5, one can easily estimate the distances between the unknown pairs, providing the distances between the closest – to them – known datasets. We generalize this observation in Algorithm 4, in which we provide a simple heuristic to calculate the pairs with the lowest similarities, i.e., the highest distances.

The function receives two arguments: A list of datasets and an integer $t$ that determines how many datasets should be evaluated. At first, two indexes are initialized: $Cls[i] = j$ indicates that $j$ is the most similar dataset to $i$, $Sim[i] = v$ indicates that the similarity between $i$ and $j$ is $v$ and $M$ retains the raw similarity metrics. A random dataset $D_c$ is then picked (line 5), and $t$ iterations occur. At each iteration, the similarities between $D_c$ and the rest of the datasets is calculated through the `similarity` function. The dataset with the least similar closest neighbor is estimated (lines 12-14) and becomes the new $D_c$ dataset. This way, the lowest similarities are eliminated at each iteration since when $D_c$ completes its comparison to the rest of the datasets,

**Algorithm 4** Approximate Similarity Matrix estimation

```
1: function ApproximateSimilarities($[D_1, \cdots, D_]$, $t$)
2:     $Sim = [], Cls = [], M = [][]$
3:     for all $D_i \in [D_1, \cdots, D_n]$ do
4:         $Sim[i] = 0.0, Cls[i] = \emptyset$
5:     $D_c$ =random($[D_1, \cdots, D_n]$)
6:     for $k = 0; k < t; k++$ do
7:         for all $D_i \in [D_1, \cdots, D_n]$ do
8:             $M[c][i]$ =similarity($D_c, D_i$)
9:             if $M[c][i] > Sim[i]$ then
10:                $Sim[i] = M[c][i], Cls[i] = c$
11:        $minSim = 1.0$
12:        for all $D_i \in [D_1, \cdots, D_n]$ do
13:            if $Sim[i] < minSim$ then
14:                $minSim = Sim[i], D_c = D_i$
15: return $Cls$, $M$
```

$Cls[c] = c$ and $Sim[c] = 1$, i.e., the closest visited neighbor dataset is itself. Eventually, after $t$ dataset evaluations occur, $Cls$ and $M$ are returned. The similarity between *any* pair of datasets $D_i$, $D_j$ can be approximated with $M[Cls[i]][Cls[j]]$. The complexity of the `ApproximateSimilarities` function is equal to $\mathcal{O}(S \cdot N \cdot t)$, $S$ representing the complexity of `similarity` function, $N$ being the number of datasets and $t$ being the number of iterations. Note that, in cases where the minimization of the computation time is of high importance, the suggested optimization can be combined with approximate MDS approaches as in [YLMW06], which aim at reducing the quadratic complexity of the subsequent MDS step.

### Online Indexing

In the presented approach we have not taken into account the process of dynamically introducing new datasets, i.e., datasets generated after the calculation of the similarity matrix and the execution of MDS, as the consideration of such cases entails the re-execution of the entire workflow. The nature of the operations applied to the datasets cannot be enforced over newly arrived datasets, since the existence of new datasets would affect the coordinates of all the datasets and – possibly – the dimensionality of the space.

The problem of projecting new datasets in the dataset space, also referred to as the *Online Indexing* problem, is the following: Given a set of datasets $D_1, \cdots, D_N$ along with their coordinates $p_1, \cdots, p_N$ respectively in a $k$-dimensional space, find the coordinates $p_{N+1}$ of a new dataset $D_{N+1}$. Note that the similarities between $D_{N+1}$ and $D_1, \cdots, D_N$ are unknown, but easily computable. Assuming that $d_1, \cdots, d_N$ are the distances between $D_{N+1}$ and $D_1, \cdots, D_N$

respectively – obtained from Equation (3.5) based on the respective similarities – we seek for a vector $p_{N+1} = (x_1, x_2, \cdots, x_k)$ that minimizes Equation (3.7). Note that this time, the coordinates of $D_1, \cdots, D_N$ are fixed and only the coordinates of $D_{N+1}$ need to be updated. Given that, the problem reduces to a typical optimization problem with the objective to find a vector $p_{N+1}$ that minimizes the Sammon stress. Since the problem space is not convex (as more than one local minima may exist), Simulated Annealing (SA) is employed. If the execution time of the Online Indexing process needs to be minimized, one can estimate the distances between $D_{N+1}$ and a subset of $D_1, \cdots, D_N$. This option reduces both the time needed to measure the similarity between the datasets and the number of steps needed by SA in order to converge, because it essentially reduces the constraints of the objective function.

Finally, although SA produces rapid results of high quality when a short portion of datasets is appended in the existing datasets, this cannot be generalized to an ever increasing number of datasets without introducing distortion to the dataset space. This is attributed to the fact the methodology discussed so far tries to approximate the coordinates of new datasets in a given space. If these datasets were available during the Similarity Estimation and MDS phases, they could contribute to a higher space dimensionality, or they could affect the coordinates of the rest of the datasets. To this end, Online Indexing is preferably used when the number of the new datasets is considerably less than $N$.

## 3.4 Accommodating graph datasets

So far, our discussion was conducted under the hypothesis that the input datasets comprise a set of tuples with specific dimensionality, i.e., they follow a tabular format. This was reflected to the selection of the similarity metrics: The statistical *distribution*, the *order* and the *size* are three metrics that are well-defined for sets (or lists, in the ordered case) of tuples. However, the workflow depicted in Figure 3.2 does not pose any limitation to the data format: Different data types can easily be accommodated, assuming that the underlying similarity metric is defined. In order to demonstrate this, we now discuss how our methodology extends to graphs.

### 3.4.1 Operators and Graph properties

One of the main differences between tuple-based and graph datasets is that while the former behave as mathematical *sets* that enumerate a number of elements (tuples), the latter represent specific structures. This structure, that is influenced by the manner through which the graph nodes are connected with the edges, contains information that is useful in different fields of study, including social network information (e.g., graph of friends), dependency tracking on task graphs, structural information (e.g., road or communication networks), etc. Because of this difference in the representation of the information, the previously discussed similarity metrics cannot be directly applied in graphs.

One similarity measure that is extensively used in the literature to measure the similarity between different graphs, is the *degree distribution* of the graph's nodes. We remind here that the degree of a node in an undirected graph equals the number of edges connected to the graph. In the case of directed graphs, we can estimate the in-degree and the out-degree of a node, i.e., the number of incoming/outgoing edges respectively and use both independently. Note that this similarity metric is not only important because it strongly correlates with different graph properties as indicated by different studies, (e.g., [JU08, BdW12, HM11]) but also, it can be efficiently computed. Specifically, one needs to first extract the degrees of all nodes for each graph independently and, then, utilize Equation 3.2 in order to measure the similarity between the distributions of the degrees between the different graphs. Essentially, the previously presented methodology needs no modification other than transforming the original graphs to tuple-based datasets that represent the degree of each of the original graph nodes. The rest of the workflow presented in Figure 3.2 remains as is.

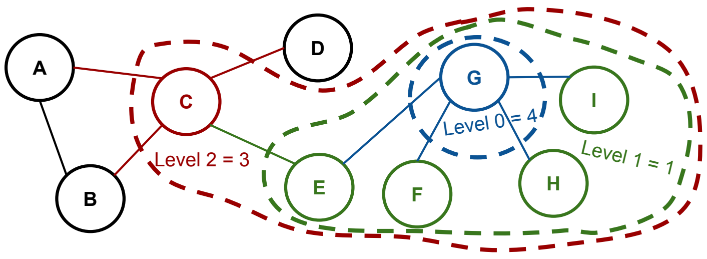
Apart from the similarity metric, the graph representation of the datasets affects the nature of the applicable operators. As discussed in Section 3.2, one key assumption that lies behind our methodology is that the operator type adheres to Equation 3.1, i.e., it receives a dataset as input and produces a real value as its outcome. The same precondition must be met for graph operators as well. As in the tuple-based case, we again focus on analytics operators; In this field of study, these operators are commonly referred to as *topology metrics*. Topology metrics can be loosely classified in three broad categories (based on [JU08, BdW12, HM11]): (a) *distance*, (b) *connectivity* and (c) *spectrum* metrics. In the first class, we find metrics that involve distances between vertices such as diameter, average distance and betweenness centrality. The second class relates to vertex degrees containing metrics such as average degree, degree distribution, clustering coefficient, etc. Finally, the third class comes from the spectral analysis of a graph and contains the computation of eigenvalues, the corresponding eigenvectors or other spectral-related metrics. For our evaluation, we chose representative topology metrics from each category.

### 3.4.2 Similarity metric estimation

From the above discussion, it is evident that if the input graphs are replaced by simple tuple-based datasets where each tuple contains the degree of each graph node, one could directly use the workflow of Figure 3.2. Let us know discuss how this mapping between the graph and the respective node degree dataset takes place.

Take the undirected graph presented in Figure 3.6 and assume that we want to extract the degrees of each node. In order to do so, one should extract the adjacency list, as presented in

Table 3.6b and then estimate the node degrees as presented in the *Degree* column. The complexity of this procedure is $\mathcal{O}(|E|)$, $E$ being the number of edges, as one needs to iterate over all edges and increase the degree of each node that connect to the edge. For the case of directed graphs, two scores should be kept: One of the ingoing and one for the outgoing degrees. In conclusion, with this simple methodology, the input graph of Figure 3.6a, is transformed to the set $\{2, 2, 3, 1, 2, 1, 4, 1, 1\}$ that corresponds to the respective node degrees. One can note that even though many graphs may present similar node degrees without being identical to the graph presented in this example, if the distributions of the node degrees, i.e., the frequency of each degree, between two graphs are similar, these graphs are likely to present a similar structure. In essence, this transformation generates a footprint for each input graph that facilitates efficient similarity extraction between different graphs and is highly correlated with the structure represented by the graph.



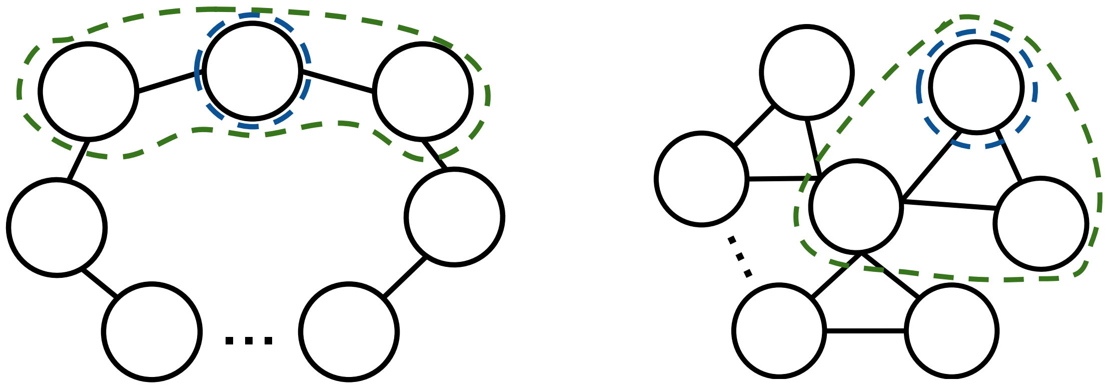
(a) Graph overview

(b) Adjacency list and node degrees

| Node Id | Neighbors | Degree |
|---|---|---|
| $A$ | $B, C$ | 2 |
| $B$ | $A, C$ | 2 |
| $C$ | $A, B, D$ | 3 |
| $D$ | $C$ | 1 |
| $E$ | $C, G$ | 2 |
| $F$ | $G$ | 1 |
| $G$ | $E, F, H, I$ | 4 |
| $H$ | $G$ | 1 |
| $I$ | $G$ | 1 |

Figure 3.6: Node degree estimation in different levels

Nevertheless, the examination of the degree of each graph node only for the immediate neighbors, may not always suffice to generate a representative footprint. As an example, consider the two graphs of Figure 3.7. Assume that each graph comprises 1001 vertices ordered in a symmetrical manner as the one that is presented in the two figures. Each node of Figure 3.7a has a degree of 2 whereas all but 1 node of Figure 3.7b has degree 2 and the last node has a degree of 1000. If we examine the similarity between these two distributions, we conclude that the two graphs present a similarity of $\frac{1000}{1001}$, but their structure is, evidently, very different. In order to overcome this limitation, that is introduced by the fact that we only examine the immediate neighbors of each node, we extend the transformation methodology in order to consider the degree of larger neighborhoods for each node. For example, in the graph of Figure 3.6a, node $G$ has 4 directly attached neighbors, and hence its degree equals 4 (this is referred to as the “Level 0” degree). The neighborhood of the graph that we examine only contains node $G$ and it is represented with blue color. If we extend this neighborhood so that it contains the immediate neighbors, we end of with the nodes contained in the green area and the degree of this region (the center of which is node $G$) equals 1 (the edge $CE$). That is referred to as the “Level 1” degree. If we extend this even further, to the second generation neighbors, the “Level 2” degree equals 3. With this extension, the degree level 0, 1 and 2 degrees for node $G$ are $\{4, 1, 3\}$ respectively.


(a) Ring graph with average degree 2

(b) Graph with triangles with average degree 2

Figure 3.7: Two example graphs with average degree 2

This extension provides more information regarding the way that different neighborhoods of the graph connect to each other, something that is reflected to multi-dimensional degree distributions where the $i$-th dimension represents the Level $i$ degree of each node. Using this mechanism, the level 0 and 1 node degrees the graphs presented in Figure 3.7 would be $\{\{2, 2\}, \cdots\}$ and $\{\{2, 998\}, \cdots, \{999, 0\}\}$ for Figures 3.7a and 3.7b respectively. Observe that even though level 0 node degree may lead to wrong conclusions regarding graph similarity, the extension to level 1 node degrees generates two-dimensional degree tuples that demonstrate the difference in the structure of the two graphs. The complexity of this algorithm is $\mathcal{O}(|V|(\overline{d})^l)$ where $\overline{d}$ is the average branching factor (average degree) and $l$ the level-depth limit. Note that the algorithm’s complexity increases rapidly with the employed level; This is the overhead of having a much more detailed footprint for each input graph.

## 3.5 Experimental Evaluation

In this section we experimentally evaluate the quality and the efficiency of our prototype using a variety of datasets. Our goal is to demonstrate that our approach:

- Projects the datasets into low-dimensional spaces that provide interesting data insights (Section 3.5.1).
- Models the considered operators with high quality and massively accelerates data analysis (Section 3.5.2).

- Generates highly informative spaces through considering multiple data properties (Section 3.5.3).
- Is customizable regarding the level of analysis detail (Section 3.5.4).
- Is capable of reducing the computation burden in order to accelerate data analysis without sacrificing accuracy (Section 3.5.5).
- Supports the introduction of new datasets (Section 3.5.6).
- Can be utilized to model the outcome of operators applied to graph datasets (Section 3.5.7).

**Experimental Setup:** All experiments are conducted on a server with two Intel Xeon E5645 processors running at 2.40GHz, 96G of main memory and 2TB of hard disk, running Ubuntu 14.04.2 LTS with Linux kernel 3.13.05. Our prototype is implemented in Go (v.1.7.6). R (v.3.3.3) was utilized for MDS and NN training.

Table 3.2: Datasets and Operators

| ID | Description | Datasets | Tuples | Operators |
|---|---|---|---|---|
| CLU | Google Cluster Monitoring [clu15] | 4797 | 46 − 2188 | **AVG**, **SUM**, COUNT (**CNT**), DBSCAN (**DBS**), Local Outlier F. (**LOF**), Eigenvalue (**EIG**), Regression (**REG**) |
| HPO | Household Power Consumption [Lic13] | 1442 | 1263 − 1440 | |
| WEA | Weather Station Recordings [noa16] | 552 | 300 − 8766 | |
| NAS | NASDAQ Tech. Stocks [sto17] | 231 | 252 | Holt-Winters (**HOL**) |
| WIK | Wikipedia Page Visits [wik17] | 4503 | 551 | ARIMA (**ARI**) |

**Datasets & Operators:** For the evaluation, five real-world sets of datasets are utilized, provided in Table 3.2. *CLU* provides the monitoring metrics of 4797 physical hosts (each dataset contains metrics from one host) of a data center at Google, running different tasks. Each tuple comprises 14 metrics of one task and different hosts run a varying number of tasks (ranging from 46 − 2188 tasks/host). *HPO* contains 1442 datasets with daily power measurements of a household in Denmark. The measurements usually take place every minute (except for some days where outages were witnessed) and each one comprises 7 features. *WEA* contains weather recordings from 552 different weather stations (w.s.) in 6 countries during 2016. Different w.s. gather measurements in different time intervals, hence, the number of tuples between different datasets ranges from 300 − 8766. All measurements comprise 6 features. For *CLU*, *HPO* and *WEA*, we test the 7 operators presented in the right column of Table 3.2. The Aggregate Functions are applied for one column of each dataset, whereas Linear Regression is applied for *HPO* and attempts to construct a linear model for predicting the active power of a metering zone based on the rest of the metrics. *NAS* contains various measurements of the NASDAQ Technology Sector stocks, for the interval 2016/05/30 – 2017/05/30 (1 dataset represents 1 stock). Finally, *WIK* contains the number of visits for $4503$ different Wikipedia articles for an interval of $551$ days. For *NAS* and *WIK* we apply the two Time-Series forecasting algorithms. Throughout the evaluation, we will refer to each operator using the dataset id and the operator id (indicated in bold in Table 3.2). For example, *HPO-LOF* refers to the Local Outlier Factor operator for the *HPO* dataset.

**Methodology:** To evaluate our methodology's efficiency, we measure the accuracy of the trained ML model that predicts the value of each of the operators. Specifically, we adopt the Normalized Root Mean Squared Error, $NRMSE = \frac{1}{y_{max}-y_{min}}\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y_i})^2}$ and the Median Absolute Percentage Error, $MdAPE = \underset{1\leq i\leq N}{median}\left|\frac{y_i-\hat{y_i}}{y_i}\right|$, where $y_i$ represents the *actual* operator value applied over $D_i$, $\hat{y_i}$ represents its approximated value and $y_{min}, y_{max}$ represent the minimum and maximum operator values respectively. MdAPE was preferred against the Mean Absolute Percentage Error since it is less susceptible to outliers. All the error metrics are estimated for *all* the datasets, i.e., each operator was exhaustively applied over all the available datasets for testing purposes, in order to avoid cases where the model consistently fails to approximate specific dataset space areas. Neural Network (NN) regression models are utilized for modeling. Each model is trained by executing the operator over a portion of the available datasets (this is referred to as the *sampling ratio*). The construction of similarity matrices that combine more than one dataset properties utilizes Equation 3.8:

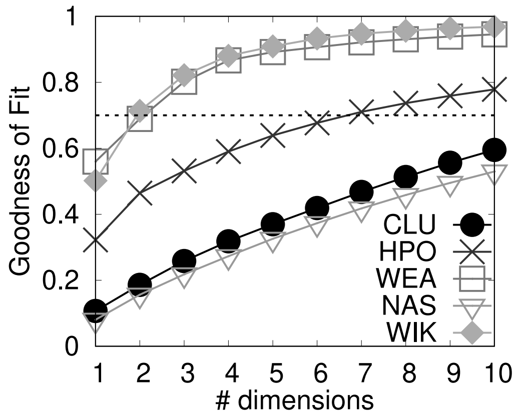
$$Similarity(A,B) = \sum_X (w_X \cdot X(A,B)) \tag{3.8}$$

in which $X \in \{Distribution, Order, Size\}$ and $\sum_X w_X = 1$. Finally, in order to decide on the dimensionality of the produced dataset spaces, Equations 3.6 and 3.7 are used.

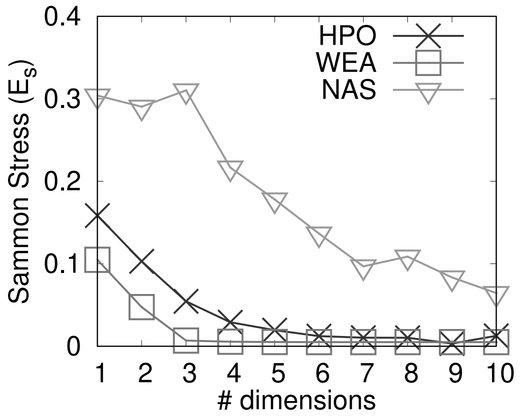
### 3.5.1 Dataset Space Construction

We begin our analysis by generating five dataset spaces, one for each set of datasets. For *CLU*, *HPO* and *WEA* the similarity matrix is solely calculated with the *Distribution* property, utilizing $32$ partitions for the comparison. For *NAS* and *WIK*, the similarity matrix combines the *Distribution* and *Order* properties with equal weights. Figures 3.8a and 3.8b provide the GoF and $E_s$ values, respectively, when obtaining dataset spaces of varying dimensionality whereas Figures 3.8c and 3.8d provide 2-d projections of the constructed *HPO* and *WEA* dataset spaces respectively.

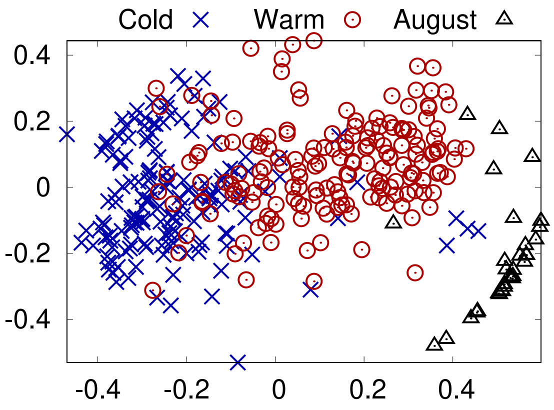
Figures 3.8a and 3.8b provide an estimate of the dimensionality of the space that should be employed in order to retain the distances of the similarity matrix without information loss. As more dimensions are employed, GoF increases and $E_s$ decreases, something that indicates that



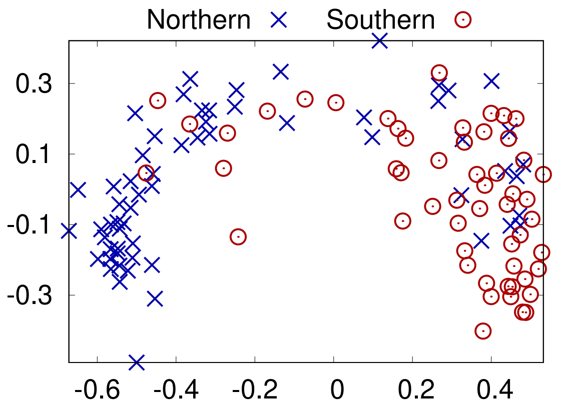
(a) Goodness-of-Fit plot



(b) Sammon Stress $E_s$ plot



(c) *HPO* dataset space



(d) *WEA* dataset space

Figure 3.8: Dataset spaces, $GoF$ and $E_s$ plots

the constructed space retains the calculated dataset distances more accurately. Observe that each set of datasets presents different dimensionality requirements for an approximation of high quality: *WEA* and *WIK* require only 3 dimensions in order to be transformed with $GoF > 0.75$, whereas *HPO* requires 7 dimensions and *NAS* and *CLU* require approximately 15. GoF's values are affected by the relationships between the datasets: The more similar the distances between the datasets, the more dimensions are required in order to generate an accurate representation of the matrix. The Sammon mapping was successfully executed only for 3 of the 5 cases, because *CLU* and *WIK* contained some dataset pairs with distance 0. Since the Sammon mapping requires no points to overlap, it could not be executed for sets that presented this phenomenon. Observe how the Sammon mapping "corrects" the dataset coordinates in order to reduce the number of required dimensions. For example, although *HPO*'s GoF diagram indicates that 7 dimensions are needed, $E_s$ presents a "knee" sooner than that: Since $E_s$ presents similar values for 4 or 5 dimensions, one can use fewer dimensions without losing in accuracy. Finally, the two plots indicate that even for the sets with increased dimensionality requirements, the application of MDS and Sammon mapping can drastically reduce dimensionality as $k \ll N$ for all cases.

Figures 3.8c and 3.8d demonstrate that a visual examination of the dataset spaces provides valuable information to a data analyst, as datasets with similar behavior are projected closer. In the *HPO* case, one can observe that the datasets are positioned according to the day of the year they were collected (only datasets of 2008 are depicted for legibility reasons) and they are organized in three clusters: Datasets from cold and warm days of the year and datasets from August. Note the interesting pattern: The household's energy consumption is much higher during the colder days (due to energy-hungry heating devices), decays during the warmer days and is zero during August. Similarly, this behavior is also exhibited for *WEA* (only depicting Swedish w.s.): The obtained datasets are clustered according to their geographic location and form two groups: The Northern ones (that present lower temperatures) and the Southern ones (with warmer and less humid climate).

Table 3.3: Modeling Accuracy and Execution Speedup of Operators

| Operator | NRMSE | | | | MdAPE | | | | Speedup ($\times$) | | | | Amortized Speedup ($\times$) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **4%** | **8%** | **16%** | **32%** | **4%** | **8%** | **16%** | **32%** | **4%** | **8%** | **16%** | **32%** | **4%** | **8%** | **16%** | **32%** |
| **CLU-AVG** | 0.086 | 0.079 | 0.073 | 0.066 | 0.125 | 0.114 | 0.100 | 0.082 | 3.21 | 2.84 | 2.32 | 1.69 | | | | |
| **CLU-SUM** | 0.085 | 0.077 | 0.070 | 0.063 | 0.182 | 0.158 | 0.136 | 0.114 | 3.21 | 2.84 | 2.32 | 1.69 | | | | |
| **CLU-CNT** | 0.115 | 0.108 | 0.104 | 0.097 | 0.433 | 0.401 | 0.377 | 0.339 | 3.21 | 2.84 | 2.32 | 1.69 | 16.34 | 9.88 | 5.52 | 2.93 |
| **CLU-DBS** | 0.098 | 0.093 | 0.088 | 0.083 | 0.201 | 0.191 | 0.173 | 0.152 | 5.69 | 4.63 | 3.83 | 2.19 | | | | |
| **CLU-LOF** | 0.082 | 0.074 | 0.070 | 0.066 | 0.146 | 0.136 | 0.125 | 0.110 | 12.13 | 8.17 | 4.94 | 2.76 | | | | |
| **CLU-EIG** | 0.069 | 0.063 | 0.058 | 0.053 | 0.089 | 0.079 | 0.071 | 0.060 | 4.27 | 3.65 | 2.83 | 1.95 | | | | |
| **HPO-AVG** | 0.104 | 0.096 | 0.088 | 0.084 | 0.013 | 0.012 | 0.011 | 0.010 | 3.93 | 3.4 | 2.67 | 1.87 | | | | |
| **HPO-SUM** | 0.070 | 0.065 | 0.056 | 0.051 | 0.149 | 0.135 | 0.122 | 0.113 | 3.93 | 3.4 | 2.67 | 1.87 | | | | |
| **HPO-CNT** | 0.098 | 0.079 | 0.069 | 0.061 | 0.115 | 0.104 | 0.092 | 0.084 | 3.93 | 3.4 | 2.67 | 1.87 | | | | |
| **HPO-DBS** | 0.124 | 0.119 | 0.114 | 0.111 | 0.146 | 0.141 | 0.133 | 0.128 | 8.30 | 6.23 | 4.16 | 2.50 | 20.27 | 11.20 | 5.91 | 3.04 |
| **HPO-LOF** | 0.064 | 0.061 | 0.055 | 0.052 | 0.068 | 0.063 | 0.061 | 0.057 | 16.64 | 9.99 | 5.55 | 2.94 | | | | |
| **HPO-EIG** | 0.071 | 0.069 | 0.067 | 0.065 | 0.065 | 0.063 | 0.059 | 0.055 | 7.33 | 5.67 | 3.90 | 2.72 | | | | |
| **HPO-REG** | 0.073 | 0.071 | 0.071 | 0.069 | 0.162 | 0.150 | 0.134 | 0.124 | 11.33 | 7.80 | 4.80 | 2.72 | | | | |
| **WEA-AVG** | 0.089 | 0.074 | 0.068 | 0.059 | 0.035 | 0.025 | 0.020 | 0.018 | 2.68 | 2.42 | 2.03 | 1.53 | | | | |
| **WEA-SUM** | 0.075 | 0.068 | 0.063 | 0.057 | 0.114 | 0.078 | 0.059 | 0.047 | 2.68 | 2.42 | 2.03 | 1.53 | | | | |
| **WEA-CNT** | 0.119 | 0.106 | 0.091 | 0.080 | 0.324 | 0.284 | 0.244 | 0.214 | 2.68 | 2.42 | 2.03 | 1.53 | 18.72 | 10.71 | 5.77 | 3.00 |
| **WEA-DBS** | 0.182 | 0.180 | 0.176 | 0.171 | 0.323 | 0.328 | 0.303 | 0.288 | 6.06 | 4.88 | 3.51 | 2.25 | | | | |
| **WEA-LOF** | 0.126 | 0.123 | 0.115 | 0.110 | 0.118 | 0.113 | 0.107 | 0.093 | 16.71 | 10.02 | 5.56 | 2.94 | | | | |
| **WEA-EIG** | 0.035 | 0.032 | 0.031 | 0.029 | 0.024 | 0.021 | 0.019 | 0.018 | 5.59 | 4.57 | 3.35 | 2.18 | | | | |
| **NAS-HOL** | 0.093 | 0.090 | 0.086 | 0.084 | 0.700 | 0.445 | 0.333 | 0.283 | 0.65 | 0.63 | 0.60 | 0.55 | 3.45 | 3.03 | 2.44 | 1.75 |
| **NAS-ARI** | 0.095 | 0.090 | 0.085 | 0.084 | 0.773 | 0.548 | 0.341 | 0.262 | 2.94 | 2.63 | 2.17 | 1.61 | | | | |
| **WIK-HOL** | 0.018 | 0.018 | 0.018 | 0.018 | 0.812 | 0.686 | 0.582 | 0.353 | 0.17 | 0.16 | 0.16 | 0.16 | 1.42 | 1.34 | 1.21 | 1.01 |
| **WIK-ARI** | 0.019 | 0.019 | 0.019 | 0.019 | 0.595 | 0.488 | 0.324 | 0.237 | 1.27 | 1.20 | 1.10 | 0.93 | | | | |

### 3.5.2 Operator Modeling

For each set of datasets, we construct the similarity matrices based on the properties their operators are affected by, i.e., Distribution and Size for *CLU*, *HPO* and *WEA* (setting $w_{Distribution} = w_{Size} = 0.5$ to Equation 3.8) and *Distribution* and *Order* for *NAS* and *WIK* ($w_{Distribution} = w_{Order} = 0.5$) and, subsequently, execute MDS to obtain the respective dataset spaces. We, then, train Neural Networks utilizing different sampling ratios that vary from 4% to 32%. In Table 3.3, we provide the error metrics and the execution speedup.

**Modeling Quality:** The $NRMSE$ and $MdAPE$ columns of Table 3.3 provide the modeling error of our approach (less is better). Generally, we can observe that increasing sampling ratios lead to decreased modeling errors. Examining each operator class in isolation, the Aggregate Functions, present low modeling errors for all datasets. Interestingly, when only a mere 4% of datasets is examined, one can approximate the operators values with an $MdAPE$ that varies from a minimal $2$ to $15\%$ for most cases (e.g., *HPO-AVG* produced less than 2% for all sampling ratios), with the exception of *CNT* that presents the highest $MdAPE$ for *CLU* and *WEA*. In comparison to the other aggregate operators, *CNT* is the one that presents values closest to $0$, something that contributes to increased $MdAPE$ without actually indicating poor approximation [Mak93].

The Density based operators (*LOF* and *DBS*) also present a robust behavior, measured both in terms of $NRMSE$ and $MdAPE$. *LOF* can be modeled with remarkable accuracy (6.5% $MdAPE$ when 4% of the datasets are considered) and improves with increasing sampling ratios for all datasets. The least accurate operator of this class is *DBS* that presents a modeling error that quickly reduces with increasing sampling ratios. This operator presents an interesting pattern: Some datasets with different distributions (which are evidently located in distant positions in the dataset space) presented similar outcomes, i.e., number of clusters. This unavoidable situation means that there does not exist a monotonic relationship between an operator's scores and their position to the dataset space. This means that the model needs to obtain more samples in order to increase its accuracy, hence the error degradation with increasing sampling ratios.

*REG* and *EIG* also present robust behavior for all the examined datasets. *EIG* in particular, presented $MdAPE$ less than 10% for all the examined datasets. This means that one can approximate the value of the most important eigenvalue of each dataset with a minimal error through actually running the operator for a minimal subset of the available datasets. On the contrary, the Time-Series operators exhibit a seemingly abnormal behavior where they present low $NRMSE$ and abnoy high $MdAPE$ that declines fast with increasing sampling ratios. As in the *CNT* operator case, the fact that both *HOL* and *ARI* produce values close to $0$ leads to increased $MdAPE$ values. The fact that these values decline fast with the sampling ratio means that while more knowledge is obtained, the approximated scores increase and, hence, $MdAPE$ decreases.

In a nutshell, the demonstrated errors indicate that our methodology successfully approximates all the considered operators. The fundamental idea behind our approach, that dataset spaces constructed in such a manner so as to reflect the relationships between them, facilitates the training of Machine Learning models that accurately approximate the behavior of the operators.

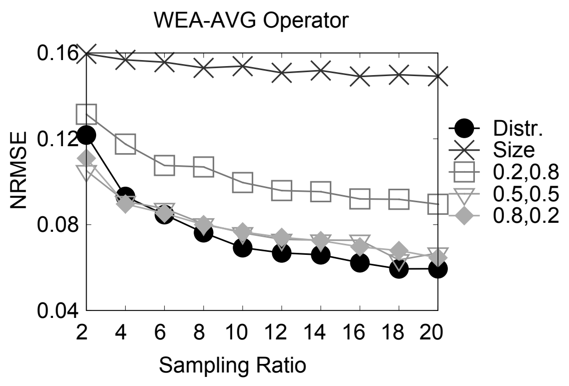
**Speedup:** The *Speedup* and *Amortized Speedup* columns of Table 3.3 provide an estimate of the time needed to approximate each operator in comparison to exhaustively executing them for all datasets (more is better). Specifically, the speedup equals $\frac{T_{op}^{(i)}}{SR \times T_{op}^{(i)} + T_{SM} + T_{MDS} + T_{ML}}$, where $T_{op}^{(i)}$ is the execution time of the $i$-th operator for all datasets, $SR$ is the sampling ratio, $T_{SM}$ is the time needed to construct the similarity matrix, $T_{MDS}$ the time of executing Multidimensional Scaling and Sammon mapping and $T_{ML}$ is the time needed to train the Neural Network. Considering that $T_{SM}$, $T_{MDS}$ are only paid once for each set of datasets, we also calculate the Amortized Speedup where $T_{op}^{(i)}$ is replaced with $\sum_i T_{op}^{(i)}$, i.e., the execution time for all operators for each case.

The examination of the *Speedup* column indicates that the sampling ratio is inversely proportional to the achieved speedup. Lower sampling ratios indicate that an operator is executed to less datasets and, hence, the achieved speedup is greater. The exact value of the speedup is directly related to the complexity of the operator. When the employed operator is complex and its execution time is large, as in the *LOF* cases, the achieved speedup approximates the optimal one, which is $\frac{1}{SR}$. For example, *HPO-LOF* presents a speedup of $16.64\times$ when 4% of the datasets are considered. On the contrary, when the operator is less complex, the achieved speedup is limited (e.g., *CLU-AVG* presents a speedup of $3.21$ when 4% of the datasets are considered), because the construction time of the similarity matrix and the MDS execution is not counterbalanced by the avoidance of operator executions.

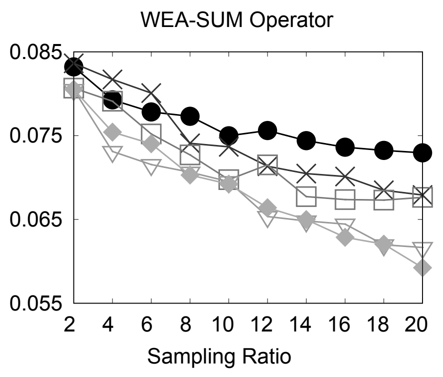
Nevertheless, when the similarity matrix construction and MDS execution is amortized to more than one operators, the achieved speedup largely increases. This is visible for the first 3 sets of datasets where the achieved speedup closely approximates $\frac{1}{SR}$. For example, the amortized speedup of *HPO* is $20.27$ for a sampling ratio of 4%. This highlights the power of the suggested methodology. As more operators emerge, they utilize the previously computed dataset space and the amount of avoided computation increases linearly with their number.

However, if the cost of constructing a dataset space is high, the achieved speedup may require many more operators in order to counterbalance the offline cost. The *NAS* and *WIK* sets require more computation time for the construction of their similarity matrices since the *Order* property entail the evaluation of Equation 3.3 which is more expensive than the previous cases. This is the reason behind the lower speedup values encountered to *NAS* and *WIK*. Nevertheless, an increasing number of operators will, eventually, lead them to save an increasing amount of execution time and, hence, accelerate the analysis for these cases as well. Evidently, the observed speedup is strongly affected by two factors: (a) the complexity of the operators and (b) the complexity of the similarity expression. Avoiding the execution of complex operators leads to higher speedups as soon as the similarity expression is efficient. For example, *CLU-LOF* presents a remarkable speedup of $16.64$ both because the operator's execution time is increased and due to the low complexity of the similarity expression. Therefore, it is crucial to obtain similarity expressions that are both efficient in terms of expressiveness and complexity.

### 3.5.3 Combining Similarity Metrics



(a) Modeling accuracy for WEA-AVG



(b) Modeling accuracy for WEA-SUM

Figure 3.9: Combining data properties

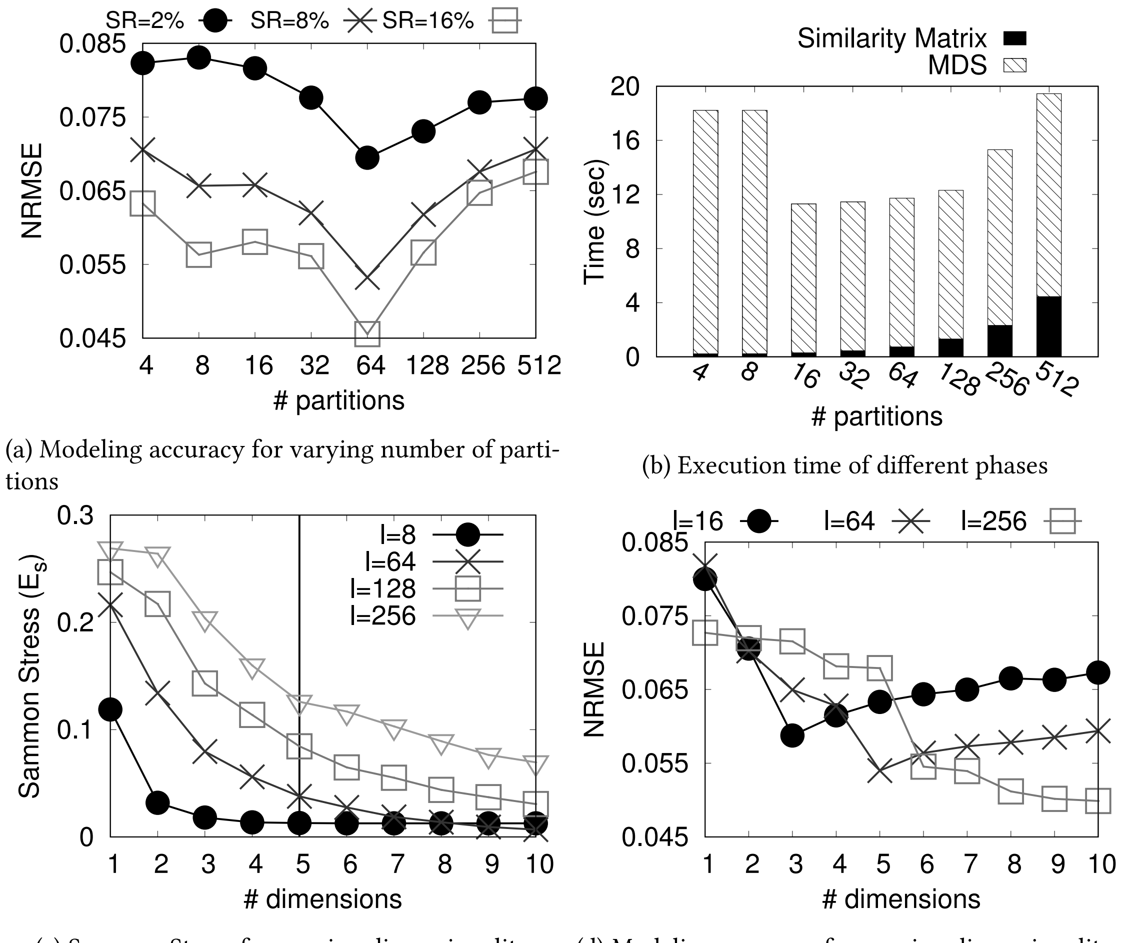
We now evaluate the impact of combining multiple similarity metrics for the *WEA* case to our scheme. Each weather station collects measurements in different intervals and, thus, the size of the datasets varies between 300 – 9000 tuples. We construct two similarity matrices based on (a) the Distribution similarity property and (b) the Size similarity property. Based on those, three more similarity matrices are constructed where ($w_{Distr.}$,$w_{Size}$) equals to $(0.2, 0.8)$, $(0.5, 0.5)$ and $(0.8, 0.2)$. All matrices are then transformed to 5-d dataset spaces and, subsequently, model *WEA-AVG* and *WEA-SUM* for each space, utilizing sampling ratios between 2% – 20%. The modeling part is executed 20 times and Figure 3.9 depict the results.

We remind here that *AVG* is only affected by the *Distribution* of the datasets whereas *SUM* is affected both by *Distribution* and *Size*. Figure 3.9a, showcases that the most informative space, i.e., the one with the least error, is constructed using only the Distribution property and the sole consideration of Size creates a space with no information as regardless of the increasing sampling ratio, the error is not reduced. However, when constructing spaces that combine both parameters, the size seems to be ignored. Interestingly, when the respective weights are equal or when favoring the important property, e.g., as in the (0.8, 0.2) case, the model is practically unaffected by the consideration of the Size. On the contrary, when an operator is affected by both properties, as in the *WEA-SUM* case, the spaces born from the combination of the primitive matrices are far more informative than the primitive ones and lead the models to increase their accuracy which is, in turn, increasing with the sampling ratio. This practically means that when a space is constructed using a combination of dataset properties, the simpler operators tend to ignore the non-interesting – to them – properties and do not present degraded modeling accuracy. On the contrary, the operators that do require the property combinations to be reflected to the space, are modeled much more efficiently than considering one property at a time. Finally, in cases where the users are knowledgable about the applied operator, the adjustment of the weights can result in a dataset space that best projects the property of interest, e.g., Distribution in the *WEA-AVG* case.

### 3.5.4 Distribution Similarity Granularity

We now evaluate the impact of the number of partitions ($l$) during the examination of the similarity of *Distribution.* Based on *HPO*, we calculate 8 different similarity matrices, using different numbers of partitions. Each of them is, subsequently, transformed to 5-d dataset spaces and, finally, a NN model for *HPO-AVG* is trained, using three different sampling ratios (2%, 8% and 16%). Recall that *HPO-AVG* is only affected by the *Distribution* of the datasets. The modeling part is repeated 20 times and in Figure 3.10a we provide the median modeling error. Figure 3.10b depicts the execution time of the similarity matrix construction and MDS and Figure 3.10c depicts $E_s$ when different dimensionality is employed for varying $l$.

Figure 3.10a presents an interesting finding: Although an increasing $l$ (between 4 – 64) seems to decrease the modeling error, as more partitions imply that the similarity comparison is more detailed, increasing $l$ beyond a point seems to increase the error. In fact, for higher sampling ratios, it is preferable to utilize a er $l$ value rather than a higher one, as for the 16%, the error for $l = 4$ is lower than the error in the $l = 256$ case. This finding is explained when examining Figure 3.10c: The utilization of more partitions increases the level of detail for the similarity estimation and, hence, an increasing number of dimensions is necessary in order to accurately transform the similarities to a dataset space. Since all cases from 3.10a comprise 5 dimensions, the similarity matrices with more partitions are less accurately projected and, hence, less accurately modeled. Therefore, there exists a clear dependency between $l$ and $k$: More partitions require space of higher dimensionality. Furthermore, the execution time presents another interesting pattern: The time needed to construct the matrix increases linearly with $l$ but MDS seems to require more time when fewer partitions are employed. Fewer partitions leave more room for the “optimal” dataset placement in 5 dimensions and SA needs more time to converge to this point. However, constructing similarity matrices with increased number of partitions and, hence,



(a) Modeling accuracy for varying number of partitions

(b) Execution time of different phases

(c) Sammon Stress for varying dimensionality

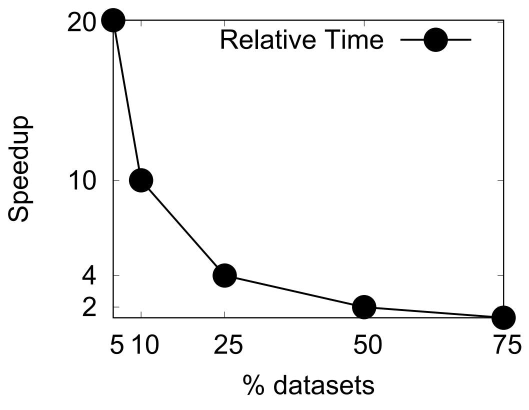
(d) Modeling accuracy for varying dimensionality

Figure 3.10: Distribution Similarity Granularity

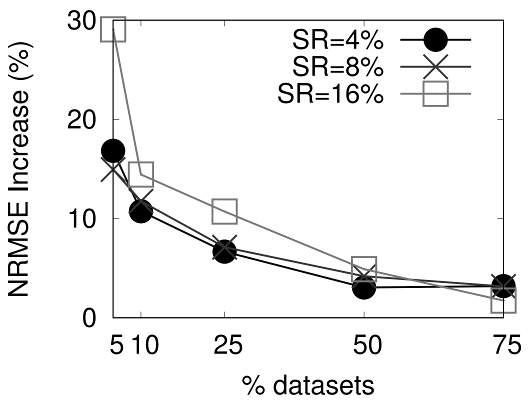
dimensions, does not always benefit the analysis. To evaluate this, in Figure 3.10d we provide the median error of NN models trained considering only 8% of the available datasets, using spaces of different dimensionality (horizontal axis), estimated using three different $l$ values. For higher $l$ values, more dimensions need to be utilized to decrease the modeling accuracy. However, the utilization of more dimensions than actually needed (e.g., when $l = 64$ there is no need to use more than 5 dimensions) leads to complex models and the modeling accuracy degrades. There exists an opportune area for selecting the appropriate dimensionality.

### 3.5.5 Approximate Similarity Matrices

We now evaluate the optimization introduced in Section 3.3.5. To this end, based on the *HPO* case (which comprises the most datasets), we construct similarity matrices based solely on the dataset distribution property, estimate an approximate similarity matrix, in which only $t$ datasets



(a) Achieved speedup vs % datasets



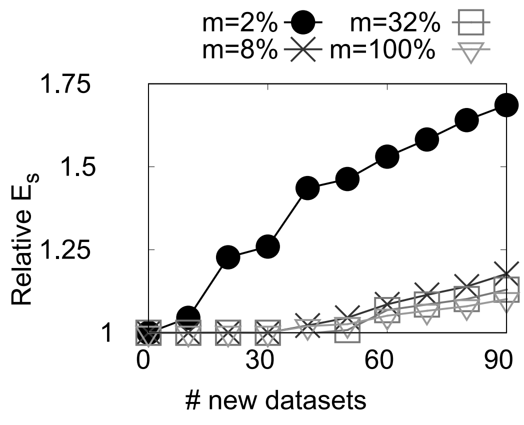
(b) Relative modeling error vs % datasets

Figure 3.11: Approximate Similarity Matrix Evaluation

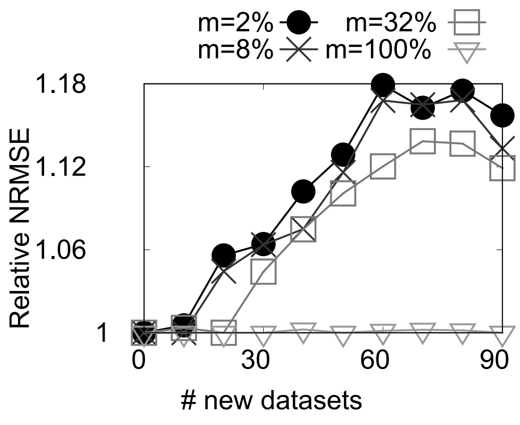
are fully calculated (expressed as a percentage of the total number of datasets). In Figure 3.11a, we provide the relative construction time of the similarity matrix for varying $t$ values, i.e., the ratio of the time needed to construct the matrix for the $t = 100\%$ case, divided by the time for each $t$ value. Our optimization linearly reduces the computation time, according to the number of datasets which are evaluated. For example, when 5% of the datasets are considered, the similarity matrix construction time is $20\times$ faster. Moreover, in order to evaluate the impact to the modeling accuracy, dataset spaces are constructed based on the similarity matrices and SVM models are trained using the respective spaces for the *HPO2* operator. In Figure 3.11b we provide the NRMSE increase of each model for varying $t$ values for three sampling rates. The NRMSE increase is defined as: $\frac{NRMSE_t - NRMSE_{100\%}}{NRMSE_t}$. The drop in accuracy becomes increasingly important when higher sampling rates are employed. For example, when SR=16% and $t = 5\%$, the NRMSE is about 30% higher. However, the error degrades quickly with increasing $t$ values, and even when $t = 10\%$, the introduced error does not exceed 14% (when $SR = 16\%$) compared to the full similarity matrix but, simultaneously, the construction is accelerated by a factor of $10\times$. Therefore, our optimization is able to significantly speedup the construction, introducing a relatively small increase in error.

### 3.5.6 Online Indexing

Finally, we evaluate the optimization introduced in Section 3.3.5. Based on *HPO* data from 2008 (366 datasets), we construct the dataset space using the Distribution Similarity metric. We then "insert" datasets for the next 3 months of 2009, i.e., introduce 90 new datasets. For each new dataset, the similarity with $m$ of the existing datasets is measured and SA is executed to identify the best coordinates for the new entries. Parameter $m$ is expressed as a portion of the already



(a) Relative Sammon Stress vs # of new datasets



(b) Relative Modeling Error vs # of new datasets

Figure 3.12: Online Indexing Evaluation

calculated datasets. Using the Sammon Stress in order to quantify the space distortion and the NRSME, in order to quantify the modeling accuracy (trained with a sampling rate of 16%), we compare the cases where the new datasets are Online Indexed for varying $m$ against the case where MDS is executed from scratch for the old and the new datasets. Figure 3.12 provides our findings expressed in relative terms, i.e., both $E_s$ and $NRMSE$ are normalized with the respective values for the case where MDS is executed from scratch. When a small number of datasets is introduced (i.e., 10), our optimization achieves both minimal $E_s$ values and minimal modeling error. This renders our approach most suitable for cases where the insertion of only a few datasets is required. When the number of new datasets increases, $E_s$ rapidly increases for two reasons: First, the new datasets have a stronger impact and the introduced errors propagate to the new entries. Second, while new datasets arrive, the dynamics of the space change. This means the dimensionality of the space would differ if all datasets were available from the beginning. Comparing against extremely few datasets generates higher errors, hence, the rapid $E_s$ increase for $m = 2\%$.

The above behavior is also observed for the modeling error, especially when $m$ receives low values. In this case, the coordinates of the new datasets become increasingly inaccurate and this severely impairs accuracy. Interestingly, when $m = 100\%$, the modeling accuracy follows the accuracy achieved when the workflow is executed from scratch, even when 90 new datasets are inserted. However, even with a considerable $m = 32\%$, NRMSE increases after 30 new datasets are introduced. In conclusion, this optimization is capable of dynamically introducing new datasets by executing a marginal number of similarity comparisons (2%) with a tolerable modeling error increase of 6%, provided that the number of new datasets does not exceed 10% of the existing ones. When this percentage increases, one should first use an increasing number of datasets for the comparisons; after a certain point, execution of the workflow from scratch is suggested.

### 3.5.7 Extension to graph data

**Datasets:** For the evaluation of graph analytics operators, we consider both real and synthetic datasets. The real datasets comprise a set of ego graphs from Twitter (*TW*) which consists of 973 user "circles" as well as a dataset containing 733 snapshots of the graph that is formed by considering the Autonomous Systems (*AS*) that comprise the Internet as nodes and adding links between those systems that communicate to each other. Both datasets are taken from the Stanford Large Network Dataset Collection [LK14].

We also experiment with a dataset of synthetic graphs (referred to as the *BA* dataset) generated using the SNAP library [LS16]. We use the `GenPrefAttach` generator to create random scale-free graphs with power-law degree distributions using the Barabasi-Albert model [BA99]. The degree distribution of the synthetic graphs, according to this model, can be given by the formula: $P(k) \sim k^{-\gamma}$, where $\gamma = 3$. We keep the vertex count of the graphs constant to 4K. We introduce randomness to this dataset by having the *initial outdegree* of each vertex be a uniformly random number in the range $[1, 32]$. The Barabasi-Albert model constructs a graph by adding one vertex at a time. The *initial outdegree* of a vertex is the maximum number of vertices it connects to, the moment it is added to the graph. The graphs of the dataset are simple and undirected. Further details about the datasets can be found in Table 3.4.

**Similarity Measures:** As a similarity measure, we compare the degree distributions of different graphs for varying levels, as discussed in 3.3.2. To investigate their strengths and limitations, we compare them against two measures functioning as our baselines. The first is a sophisticated similarity measure not based on degree but rather on distance distributions (from which the degree distribution can be deduced). *D-measure* [SCDG+17] is based on the concept of network node dispersion (NND) which is a measure of the heterogeneity of a graph in terms of connectivity distances. From a computational perspective, *D-measure* is based on the all-pairs shortest paths algorithm, which can be implemented in $O(|E| + |V|log(|V|))$ using Fibonacci heaps. It is a state-of-the-art graph similarity measure with very good experimental results for both real and synthetic graphs. It is considered efficient and since it incorporates additional information to the degree distribution, it is suitable to reason about how sufficient the measures we propose are.

Our second baseline comes from the extensively researched area of graph kernels. Kernel methods for comparing graphs were first introduced in [GFW03]. Many kernels have been since proposed to address the problem of similarity in structured data [GDG+18]. In our evaluation, we incorporate the *Random Walk Kernel* [GFW03] which intuitively performs random walks on

Table 3.4: Datasets overview

| **Name** | **Size (N)** | $\overline{\lvert V\rvert}$ | $\overline{\lvert E\rvert}$ | **Range** $\lvert V\rvert$ | **Range** $\lvert E\rvert$ |
|---|---|---|---|---|---|
| **TW** | 973 | 132 | 1,841 | min: 6<br>max: 248 | min: 9<br>max: 12,387 |
| **AS** | 733 | 4,183 | 8,540 | min: 103<br>max: 6,474 | min: 248<br>max: 13,895 |
| **BA** | 1,000 | 4,000 | 66,865 | 4,000 | min: 3,999<br>max: 127,472 |

a pair of graphs and counts the number of matching walks as a measure of their similarity. For the purposes of our evaluation, we opted for the geometric *Random Walk Kernel* (*rw-kernel*) as a widely used representative of this class of similarity measures. The complexity of random walk kernels is in the order of $O(|V|^6)$, however faster implementations with speedups up to $O(|V|^2)$ exist [VSKB10]. In order to avoid the *halting* phenomenon due to the kernel's decay factor ($\lambda^k$) we set $\lambda = 0.1$ and the number of steps $k \leq 4$, values that are considered to be reasonable for the general case [SB15].

**Graph Operators:** As stated in Section 3.3.2, graph operators can be clustered in three categories: *distance*, *connectivity* and *spectrum* operators. As representatives of the distance class, we choose betweenness (**bc**), edge betweenness (**ebc**) and closeness centralities (**cc**) ([NG04, BE05]), three metrics that express how central a vertex or edge is in a graph. The first two consider the number of shortest paths passing from a vertex or edge while the third is based on the distance between a vertex and all other vertices. From the spectrum class, we choose spectral radius (**sr**) and eigenvector centrality (**ec**). The first is defined as the largest eigenvalue of the adjacency matrix of the graph. As a metric, it is associated with the robustness of a network against the spreading of a virus [JKMvD06]. The second is another measure that expresses vertex centrality [Bon87]. It is based on the eigenvectors of the adjacency matrix. Finally, as a connectivity related metric we consider PageRank (**pr**), a centrality measure used for ranking web pages based on popularity [BP98]. All measures, except spectral radius, are centrality measures expressed at vertex level (edge level in the case of edge betweenness). Since we wish all our measures to be expressed at graph level, we will be using a method attributed to Freeman [Fre77] to make that generalization. This is a general approach that can be applied to any centrality [BE05], and measures the average difference in centrality between the most central point and all others: $c(G) = \frac{\sum_{i,j \in V} c(j)^* - c(i)}{|V|-1}$ $c(G)$ being the measure at graph level, $c(i)$ the centrality value of the $i$-th vertex of $G$ and $c(j)^*$ the largest centrality value for all $i \in V$.

Table 3.5: Modeling Errors and Execution Speedup for Different Sampling Rates

| **Dataset** | **Metric** | **MdAPE (%)** | | | **nRMSE** | | | **Speedup ×** | | | **Amortized Speedup ×** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $p$=**5%** | $p$=**10%** | $p$=**20%** | $p$=**5%** | $p$=**10%** | $p$=**20%** | $p$=**5%** | $p$=**10%** | $p$=**20%** | $p$=**5%** | $p$=**10%** | $p$=**20%** |
| | **sr** | 1.3 | 1.1 | 0.9 | 0.05 | 0.03 | 0.02 | 6.4 | 3.8 | 3.3 | | | |
| | **ec** | 0.1 | 0.1 | 0.0 | 0.01 | 0.00 | 0.00 | 5.7 | 4.5 | 3.1 | | | |
| | **bc** | 1.4 | 1.2 | 1.1 | 0.04 | 0.03 | 0.03 | 15.7 | 8.8 | 4.7 | | | |
| **AS** | **ebc** | 3.1 | 2.7 | 2.4 | 0.04 | 0.04 | 0.04 | 17.3 | 9.3 | 4.8 | 18.0 | 9.5 | 4.9 |
| | **cc** | 0.4 | 0.4 | 0.3 | 0.01 | 0.01 | 0.01 | 14.0 | 8.2 | 4.5 | | | |
| | **pr** | 0.9 | 0.8 | 0.7 | 0.05 | 0.04 | 0.03 | 5.7 | 4.4 | 3.1 | | | |
| | **sr** | 16.3 | 15.3 | 14.7 | 0.10 | 0.10 | 0.10 | 13.3 | 8.0 | 4.4 | | | |
| | **ec** | 8.0 | 7.7 | 7.7 | 0.14 | 0.14 | 0.13 | 13.1 | 7.9 | 4.4 | | | |
| | **bc** | 17.8 | 17.5 | 16.8 | 0.16 | 0.15 | 0.14 | 13.0 | 7.8 | 4.4 | | | |
| **TW** | **ebc** | 29.5 | 29.8 | 28.6 | 0.12 | 0.12 | 0.12 | 13.5 | 8.0 | 4.4 | 14.8 | 8.5 | 4.6 |
| | **cc** | 3.3 | 3.0 | 2.9 | 0.10 | 0.10 | 0.09 | 13.0 | 7.9 | 4.4 | | | |
| | **pr** | 9.2 | 7.7 | 7.2 | 0.07 | 0.06 | 0.05 | 13.2 | 7.9 | 4.4 | | | |
| | **sr** | 3.3 | 1.8 | 0.9 | 0.04 | 0.03 | 0.03 | 5.6 | 4.4 | 3.0 | | | |
| | **ec** | 0.4 | 0.3 | 0.3 | 0.01 | 0.01 | 0.01 | 3.7 | 3.1 | 2.4 | | | |
| | **bc** | 10.3 | 10.1 | 9.6 | 0.10 | 0.05 | 0.02 | 12.6 | 7.7 | 4.4 | | | |
| **BA** | **ebc** | 10.9 | 9.3 | 8.5 | 0.10 | 0.09 | 0.01 | 13.6 | 8.1 | 4.5 | 16.3 | 9.0 | 4.7 |
| | **cc** | 2.4 | 2.2 | 2.1 | 0.04 | 0.04 | 0.03 | 9.9 | 6.6 | 4.0 | | | |
| | **pr** | 6.7 | 6.1 | 5.9 | 0.06 | 0.05 | 0.05 | 3.6 | 3.0 | 2.3 | | | |

**Modeling Accuracy:** To evaluate the accuracy of our approximations, we calculate *MdAPE* and *nRMSE* for a randomized 20% of our dataset: For a dataset of 1,000 graphs, 200 will be chosen at random for which the error metrics will be calculated. We vary the sampling ratio $p$, i.e., the number of graphs for which we actually execute the operator, divided by the total number of graphs in the dataset. The results are displayed in Table 3.5. Each row represents a combination of a dataset and a graph operator with the corresponding error values for different values of $p$ between 5% and 20%.

The results in Table 3.5 showcase that our method is capable of modeling different classes of graph operators with very good accuracy. Although our approach employs a degree distribution-based similarity measure, we observe that the generated similarity matrix is expressive enough to allow the accurate modeling of distance- and spectrum-related metrics as well, achieving errors well below 10% for most cases. In *AS* graphs, the $MdAPE$ error is less than 3.2% for all the considered operators when only a mere 5% of the available graphs is examined. Operators such as closeness or eigenvector centralities display low $MdAPE$ errors in the range of $< 8\%$ for all datasets. Through the use of more expressive or combined similarity measures, our method can improve on these results, as we show later in this Section. We also note that the approximation accuracy increases with the sampling ratio. This is expressed by the decrease of both $MdAPE$ and $nRMSE$ when we increase the size of our training set. These results verify that modeling such graph operators is not only possible, but it can also produce highly accurate models with marginal errors.

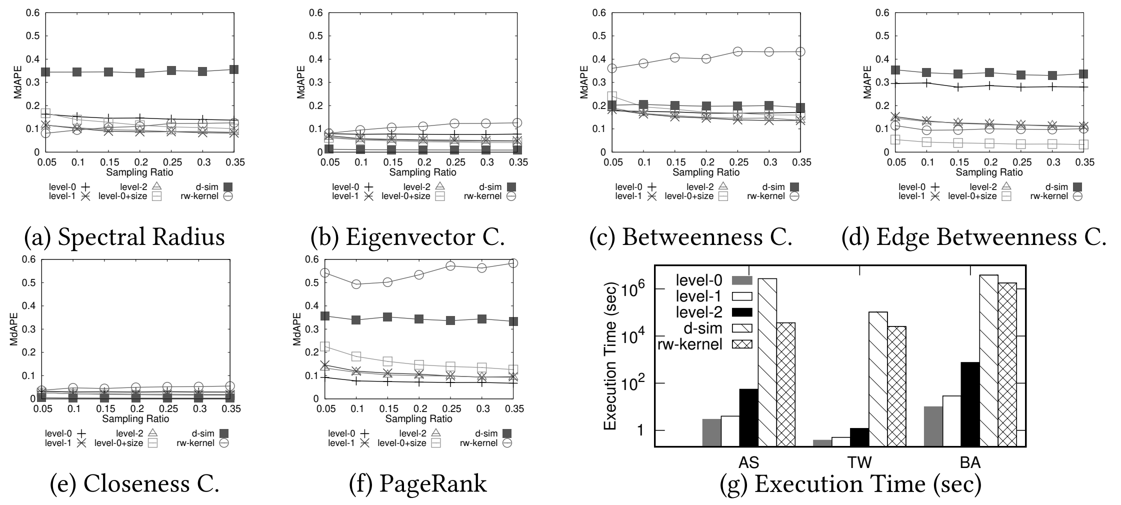
Specifically, in the case of the *AS* dataset, we observe that all the operators are modeled more accurately than in any other real or synthetic dataset. This can be attributed to the topology of the *AS* graphs. These graphs display a linear relationship between vertex and edge counts. Their clustering coefficient displays very little variance, suggesting that as the graphs grow in size they keep the same topological structure. This gradual, uniform evolution of the *AS* graphs leads to easier modeling of the values of a given graph topology metric.

On the other hand, our approach has better accuracy for degree- than distance-related metrics in the cases of the *TW* and *BA* datasets. The similarity measure we use is based on the degree distribution that is only indirectly related to vertex distances. This can be seen, for example, in the case of *BA* if we compare the modeling error for the betweenness centrality (bc) and PageRank (pr) measures. Overall, we see that eigenvector and closeness centralities are the two most accurately approximated metrics across all datasets. After we find PageRank, spectral radius, betweenness and edge betweenness centralities. Willing to further examine the connection between modeling accuracy and the type of similarity measure used, we have experimented with different similarity measures, leading to the inclusion of *D-measure* and *rw-kernel* in our evaluation. This has also lead to the development of the degree-level similarity measures and the combination of similarity matrices in the cases of degree distribution similarity matrix and vertex count similarity matrix.

**Execution Speedup:** Next, we evaluate the gains our method can provide in execution time when compared to the running time of a graph operator being executed for all the graphs of each dataset. Similarity matrix computation is a time-consuming step that is executed once for each dataset. Yet, an advantage of our scheme is that it can be reused for different graph operators. Consequently, time costs can be amortized over different operators. In order to provide a better insight, we calculate two types of speedups: One that considers the similarity matrix construction from scratch for each operator separately (provided in the *Speedup* column of Table 3.5) and one that expresses the average speedup for all six metrics for each dataset, where the similarity matrix has been constructed only once (provided in the *Amortized Speedup* column of Table 3.5). For example, in the case of the *AS* dataset and for the spectral radius metric, our approach is $6.4\times$ faster when using 5% sampling ratio than the computation of the spectral radius for all the graphs of *AS*. Additionally, if we utilize the same matrix for all six operators, this increases the speedup to $18\times$.

The observed results highlight that our methodology is not only capable of providing models of high quality, but also does so in a time-efficient manner. A closer examination of the Speedup columns shows that our method is particularly efficient for complex metrics that require more computation time (as in the *ebc* and *cc* cases for all datasets). The upper bound of the theoretically anticipated speedup equals $\frac{1}{p}$, i.e., in the $p = 5\%$ case each operator runs on 20 times fewer graphs than the exhaustive modeling, without taking into account the time required for the similarity matrix and the training of the kNN model. Interestingly, the *Amortized Speedup* column indicates that when the procedure of constructing the similarity matrix is amortized to the six operators under consideration, the achieved speedup is very close to the theoretical one. This is indeed the case for the *AS* and *BA* datasets that comprise the largest graphs, in terms of number of vertices: For all $p$ values, the amortized speedup closely approximates $\frac{1}{p}$. In the case of the *TW* dataset which consists of much smaller graphs and, hence, the time dedicated to the similarity matrix estimation is relatively larger than the previous cases, we observe that the achieved speedup is also sizable. In any case, the capability of reusing the similarity matrix, which is calculated on a per-dataset rather than on a per-operator basis, enables our approach to scale and be more efficient as the number and complexity of graph operators increases.

**Comparing Similarity Measures:** The results of the similarity measure comparisons, in the case of the *TW* dataset, are displayed in Figure 3.13, where $MdAPE$ is used to express the modeling error. We compare six similarity measures: The *degree distribution + levels* measure (for levels equal from 0 to 2), a combination of *level-0* degree distribution with vertex count (denoted by *level-0 + size*) (combined using Equation 3.8 with equal weights), *D-measure* and the *Random Walk Kernel* based similarity measure (denoted by rw-kernel). The results indicate the



(a) Spectral Radius (b) Eigenvector C. (c) Betweenness C. (d) Edge Betweenness C.

(e) Closeness C. (f) PageRank (g) Execution Time (sec)

Figure 3.13: Similarity Metrics Comparison for *TW* Dataset

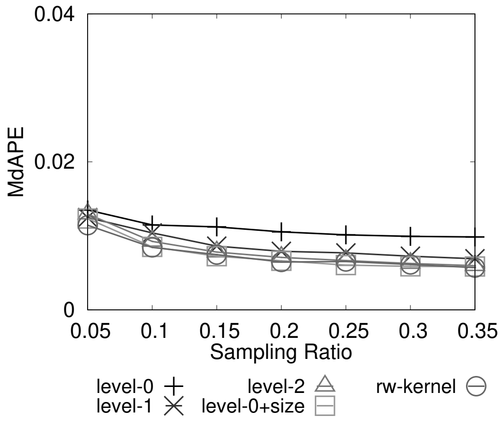
impact that the choice of similarity measure has on modeling accuracy. A more suitable to the modeled operator and detailed similarity measure is more sensitive to topology differences and can lead to better operator modeling.

In all figures, with the exception of PageRank, we observe that the *degree distribution + levels* similarity measure, for a number of levels, can model an operator more accurately than the simple degree distribution-based, effectively reducing the errors reported in Table 3.5. Indeed, the addition of more levels to the degree distribution incorporates more information about the connectivity of each vertex. This additional topological insights contribute positively to better estimate the similarity of two graphs. For instance, this allows the MdAPE error to drop from 29.5% to about 15% when utilizing a *level-2* similarity for edge betweenness centrality and $p = 5\%$.

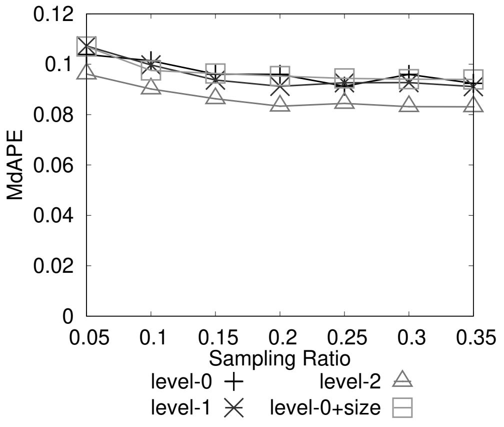
Examining the modeling quality, we observe that it increases but only up to a certain point, in relation to the topology of the graphs in the dataset. For example, since *TW* comprises of ego graphs, all the degrees of level $> 2$ are zero, since there exist no vertices with distance greater than 2; therefore, employing more levels does not contribute any additional information about the topology of the graphs when computing their similarity.

Finally, we observe that, in specific cases, such as PageRank (Figure 3.13f), enhancing the degree distribution with degrees of more levels introduces information that is interpreted as noise during modeling. PageRank is better modeled with the simple degree distribution as a similarity measure. As such, we argue that for a given dataset and graph operator, experimentation is required to find the number of levels that give the best tradeoff between accuracy and execution time.

We next concentrate on the effect of the combination of degree distribution with vertex count in the modeling accuracy. We note that the vertex count contributes positively in the modeling



(a) *AS* dataset



(b) *BA* dataset

Figure 3.14: Similarity Metric Comparison for Betweenness C.

of distance-related metrics while having a neutral or negative impact on degree- and spectrum-related metrics. This is attributed to the existence of, at least, a mild correlation, between vertex count and *bc*, *ebc* and *cc* [JU08]. For our least accurately approximated task, edge betweenness centrality, employing the combination of measures results in a more than $6\times$ decrease in error.

For *D-measure*, our experiments show that, for distance-related metrics it performs at least as good as the *degree distribution + levels* similarity measures for a given level, with the notable exception of the PageRank case. On the other hand, the degree distribution can be sufficiently accurate for degree- or spectrum-related metrics. As *D-measure* is based on distance distributions between vertices, having good accuracy for distance-related measures is something to be expected. However, *degree distribution + levels* measures exhibit comparable accuracy for distance-related metrics as well. A good example of the effectiveness of *D-measure* is shown in the case of closeness centrality that involves all-pairs node distance information directly incorporated in *D-measure* as we have seen in Section 3.5.7. In Figure 3.13e we observe that by adding levels we get better results, vertex count contributes into even better modeling but *D-measure* gives better approximations. Yet, our methods' errors are already very small (less than 3%) in this case. Considering the *rw-kernel* similarity measure, we observe that it performs poorly for most of the modeled operators. Although its modeling accuracy is comperable to the *degree distribution + levels* similarity measures for some operators, we find that for a certain level or in combination with vertex count a degree distribution-based measure has better accuracy. Notably, *rw-kernel* has low accuracy for degree and distance related operators while performing comperably in the case of spectrum operators.

Identifying betweenness centrality as one of the hardest operators to model accurately, we present $MdAPE$ approximation errors for *AS* and *BA* in Figures 3.14a, 3.14b. These Figures do not include *D-measure*, since it was not possible to compute it because of its running time. We note that the approximation error is below 12% and that the *degree distribution + levels* measures further improve on it for both datasets. Compared to *TW* (Fig. 3.13), we observe that the *level-2* similarity measure provides better results for *AS* and *BA* but not *TW*, something we attribute to the fact that *TW* consists of ego graphs with their *level-2* degree being equal to zero. Finally, it is expected that *level-0 + size* for *BA* to be no different than plain *level-0* since all the graphs in *BA* have the same vertex count by construction.

Although the above similarity measures are comparable in modeling accuracy, they are not in execution time. A comparison in computation time for different levels of the *degree distribution + levels* similarity measure is presented in Figure 3.13g. In the case of *D-measure*, the actual execution time is presented for the *TW* dataset, since it was prohibitively slow to compute it for the other two datasets. For the remaining two datasets, we have computed *D-measure* on a random number of pairs of graphs and then projected the mean computation time to the number of comparisons performed by our method for each dataset.

Our results show that the overhead from *level-0* to *level-1* is comparable for all the datasets. However, that is not the case for *level-2*. The higher the level, the more influential the degree of the vertices becomes in the execution time. Specifically, while we find *level-0* to be $3.2\times$ faster than *level-2* for *TW*, we observe that in the case of *AS* and *BA* it is $19\times$ and $76\times$ faster. The computation of the *D-measure* and the *rw-kernel*, on the other hand, are orders of magnitude slower, i.e., we find *level-0* to be about 385K times faster than *D-measure* for the *TW* dataset, while it is 273K and 933K times faster for the *BA* and *AS* datasets, respectively. Given the difference in modeling quality between the presented similarity functions, we observe a clear tradeoff between quality of results and execution time in the context of our method.

## 3.6 Discussion

Let us now discuss in more detail some aspects of the proposed methodology. First, the motivation behind this methodology is to accelerate exploratory data analysis, in order to provide insights on the properties of each dataset that make them more suitable for specific analytics tasks. Although accuracy is not our primary concern, in the sense that our method does not provide theoretical guarantees on the approximated operator values, in practice we saw that the tested operators are modeled accurately enough for a plethora of real-world datasets. On the same time, a data analyst can obtain an estimate of an operator's outcome applied to different datasets many times faster than applying it for all of them in an exhaustive manner. This is the true power of this work: As new operators emerge the amount of exhaustive operator executions is massively increased and, hence, the achieved speedup is inversely proportional to the desired modeling accuracy (higher accuracy implies more operator executions, hence lower speedup).

This property of constructing operator-agnostic dataset spaces, in the sense that the coordinates of each dataset into the space are only influenced by dataset similarity and not any operator, render the core part of our work reusable for different operators.

Second, even though our analysis was initially based on tuple-based data, we already demonstrated that an extension to graph datasets is not only possible, but it also produces results of high quality. In fact, the key idea behind this work, i.e., that similar datasets should have a similar impact to an operator's outcome, is not solely tailored for tuple-based and graph datasets. The key aspect behind the adoption of a new dataset format such as text, image, etc., is the definition of a similarity function that manages to quantify the key properties that affect most the respective operator. There is a profound relationship between the similarity function and the applied operator. If the similarity metric achieves to point out these data characteristics that mostly influence the operator, then a smart Machine Learning model will be able to capture this relationship.

Although this seems to violate the "operator-agnostic" property that was discussed before, as there must be a relationship between the operator and the similarity metric (thus, the space), note that this operator-to-similarity metric relationship is not one-to-one. In the tuple-based case, we identified 3 key similarity metrics that have very high impact to a variety of operators, either directly (e.g., Aggregate Functions are instantly affected by distribution) or indirectly (e.g., density and spectrum operators are both affected by distribution). A similar observation can also be done for graph datasets. We argue that although the number of applicable operators to a dataset is huge, the properties that different operators are affected by is not equivalently large, but only a few properties (hence, similarity functions) suffice. Domain knowledge and experience is required for selecting the appropriate ones. However, when the appropriate properties are selected, the same methodology can be applied without modifications.

Finally, even though our work was exclusively focused on static datasets, it should not be overlooked that modern data sources are constantly evolving both through the insertion of new datasets and through updating the existing ones. The first case, is largely addressed by the algorithm introduced in Section 3.3.5. Using this approach, we can augment existing dataset spaces through introducing up to 10% of new datasets without causing severe distortions to them. On the contrary, updating existing datasets through, e.g., adding/removing tuples or nodes/edges, is not addressed by this work. In order to efficiently support this capability, we should be able to run both the similarity metrics and MDS in an incremental fashion. This extension can find interesting applications to high-velocity and latency-sensitive operators applied on evolving graphs, time-series, etc., and it provides an interesting foundation for future work.

CHAPTER 4

# Related Work

This thesis presents two methodologies aiming at modeling the behavior of an operator from two different perspectives. The first methodology is focused on modeling its performance when executed with different resource configurations and application-level parameters. The second one focuses on modeling the operator's outcome when executed over different datasets. Therefore, the research presented in this thesis spans multiple diverse fields of related work. In the following sections we present research works that relate to the methodologies presented in this thesis, where we also examine how our work exploits or differentiates from them.

## 4.1 Performance Modeling of Big Data Applications

Performance modeling is a vividly researched area. The challenge of accurately predicting the performance of a Big Data application emerged since the very beginning of the Big Data era, where the newly proposed systems need to modeled appropriately in order to be utilized in the most cost efficient manner. Although performance modeling of Big Data applications can be viewed as a subcategory of the performance modeling field of research, the deployment of these systems presents two peculiarities that strongly influence the modeling approaches presented in the literature. First, their distributed nature radically increase their configuration space, since different application modules may span to a varying number of machines that may theoretically grow beyond limits. Second, in order to fully exploit this design, Big Data applications typically run on Cloud infrastructures that add extra complexity to measuring the impact of the input resources to the output performance.

The distinct approaches that attempt to partially or fully tackle this challenge can be categorized in four categories. First, *cloud benchmarks* attempt to quantify the impact of virtualization to an application's performance. Second, *simulation* methodologies attempt to theoretically model the relationship between the input (i.e., the resources and application level parameters) and the output (i.e., the performance metric) of the application. Third, *emulation* techniques are also based on simulation but also try to utilize actual performance metrics for application parts by running the application in a smaller scale. Finally, *black-box* approaches approach the problem from the opposite direction: They deploy the Big Data application for some configurations, obtain the respective performance metrics for these and use statistical inference and Machine Learning in order to construct the performance model. Let us know present each category separately:

**Cloud benchmarking:** The tools and systems that belong in this category mainly attempt to model the underlying resources rather than the behavior of the application itself. Nevertheless, they are mentioned here since they provide invaluable insight to understanding an application's behavior when this is executed over a cloud platform.

HiBench [HHD$^+$11] is one of the first cloud benchmarking suites that exclusively focuses on Big Data applications. The first versions of this benchmark consisted of a variety of Big Data operators implemented on Hadoop whereas in the later versions Spark jobs were added as well. It supports both batch and streaming operators. It is one of the most popular benchmarking tools as it is frequently used not only for measuring the performance of the Cloud but also for measuring the performance of components of the Hadoop stack.

Cloudsuite [FAK$^+$12] is another popular cloud benchmarking suite that is not restricted to typical Big Data applications, but also contains a variety of applications that are commonly encountered to cloud deployments. This suite focuses on providing certain benchmarks with dedicated resource requirements (e.g., purely network-bound as in the Video Streaming case or memory-bound as in the memcached case) in order to benchmark the hardware configuration of the underlying host. Special efforts have been put in making this suite as easily usable as possible, since each benchmark application is shipped in a Docker container that can be directly pulled and executed without requiring manual installation.

PerfKitBenchmarker [per] is an initiative backed by numerous companies and academic institutions in order to create a standard benchmarking suite that covers all modern needs of benchmarking. It contains a variety of applications that are utilized as memory caches (e.g., Redis), document stores (e.g., MongoDB), distributed NoSQL databases (e.g., Cassandra), etc.

These benchmark suites can facilitate understanding the performance of the virtualized hardware of the cloud infrastructure, but also, they can be used as a means of comparison between different platforms. CloudCMP [LYKZ10] is one of the first works that tackle the challenge of measuring the performance offerings between different providers also considering the efficiency of the resources. The proposed methodology targets to compare different providers in terms of all type of resources, i.e., compute, storage and network and it remains generic enough in order to be applicable to different providers.

The importance of measuring cloud performance has given birth to industrial solutions such as CloudHarmony [cloc], Cloud Spectator [clod] and Load Impact [loa] in order to collect and process statistics regarding the performance of the cloud platforms. Through them, these companies can provide insightful comparisons that aim at providing suggestions to a cloud user regarding which cloud provider should be preferred for different workflows.

**Simulation:** Simulation approaches attempt to do “white-box” modeling of the application under examination: They construct analytical models that explain how different application components interact with each other and what is the expected performance output of each input. The inputs, the output and the intermediate stages of the model are defined by the application, usually in terms of mathematical function. This model is then used by simulators in order to predict the anticipated performance for a given model input.

CloudSim [CRB+11] is a toolkit that focuses on modeling the performance of the cloud environment when simulated workloads are executed. The focus of this tool is to evaluate resource provisioning policies according to the application that run on the Cloud. Even though the focus is mainly put on the infrastructure side, rather than the application side, CloudSim can also act as a simulation environment where the user can provide their workflow and measure the Cloud’s efficiency for it.

CDOSim [FFH12] is an approach that targets to model the Cloud Deployment Options (CDOs) and simulate the cost and performance of an application. It relies on CloudSim and its main objective is to model the possible configuration options the application can be deployed with, estimate their performance and cost and, finally, through simulating the workload to the cloud, returns the best possible CDOs to the user.

CloudAnalyst [WCB10] is another CloudSim based tool that focuses on simulating the behavior of large scale distributed applications deployed on cloud infrastructures. It accepts descriptions of application workloads that contain information regarding the geographic location of the service’s users and the location of data centers, their size and latency requirements. It is particularly useful for modeling latency-sensitive applications with deployments that span all over the globe.

WebProphet [LZZ+10] is a performance modeling system that specializes in web services. The main idea behind this work lies on identifying the dependencies of a given web service. Since delays in requests at any given service will be propagated to the depending components, WebProphet generates a service dependency graph and, through simulation, statistically predicts the latency of the web service.

Simulation based approaches have been extensively used in different fields of study as they present some characteristics that make them more preferable than other modeling techniques. They are easy to build, as they do not require any interaction with the application. They achieve high accuracy for applications with simpler structure and they are usually agnostic of the underlying implementation, something that makes them portable. Nevertheless, they also present downsides that reduce their applicability. First of all, the generation of such a model require high expertise and deep understanding of the application's functionality. For more complex cases, the generated models will either become extremely complicated or they will suffer from *latent variables* that hinder their accuracy. Finally, although models of these type can provide high-level overview of the application's design, they do not consider deployment details (e.g., the storage type, network connection, etc.) that may have a great impact to the final performance.

**Emulation:** Methodologies based on emulation resemble the simulation-based approaches, with the difference that they also attempt to run real benchmarks for small parts of the application, gather traces based on these benchmarks and "replay" them at higher scale in order to predict application performance. Their target is to create more complete application models that rely both on white-box, analytical models but also consider infrastructure-related information through micro-benchmarking.

CloudProphet [LZK+11] is a system that bases its functionality on the principle of "trace-and-replay" and it tackles the problem of predicting the performance of an application that is under migration to the cloud. Instead of deploying the application to the target cloud environment and measuring its performance, CloudProphet follows a different approach: Traces are collected when the application is stressed in a local environment and these traces are, then, replayed to the cloud in order to obtain an estimate of the application's performance. This is particularly tested for storage resources, where instead of actually deploying the entire application, CloudProphet tracks the disk access pattern in a local environment and then attempts to replay it in the cloud, measuring the deviation and predicting the total performance.

Similarly to CloudProphet, //Trace [MWS+07] is a system that aims at modeling the I/O behavior of a given application. Specifically, using traces from given workloads to various filesystems, //Trace measures the response times of each filesystem call. In parallel, the application model that contains its structure and module interdependencies, is utilized and the gathered traces are applied to it through the "causality" engine, that is responsible to replay the traces in order to extract a performance model for the final application.

The work in [MTK+15] also aims at modeling the performance of Big Data applications through studying their I/O behavior. Initially, using a set of well-known benchmarks an initial profile is obtained for the performance of the storage medium in cloud environments. In order to tackle heterogeneity, different storage types are examined, spanning from traditional mechanical HDDs, to fast SSDs and distributed filesystems. Next, the known application model is interpolated with these traces and an estimate of the final performance is provided.

The approach presented in [HPE$^+$06] solves a slightly different problem: Given a number of platforms and a set of applications, this work aims at selecting an appropriate platform for each application, based on the type of the workloads of the latter. The main idea of this work lies on executing a set of pre-defined micro-benchmarks to each platform and obtaining the performance characteristics for each one. Later, when a new application needs to be deployed to one of these platforms, its similarity to each of the examined benchmarks is calculated, using different approaches. In essence, this begins from a set of micro-benchmarks and constructs a metric space that is formed of dimensions that represent microarchitecture-independent characteristics and its points represent the application under examination. According to their features, the respective platform is selected.

In conclusion, emulation-based approaches represent an attempt to augment known application models with infrastructure-related information. Even though they work well for applications with accurate models and require minimal deployment overhead in order to be constructed, they suffer from the limitations of the pure simulation-based models: The modeling part requires experience and its accuracy can be jeopardized by latent models variables or poorly approximated application behavior.

**Black-box:** Finally, and in contrast to the previous modeling categories, the "black-box" approaches take a purely mathematical view to the modeling problem. The performance is viewed as a function and the problem is formulated as a function approximation problem. The input space of the function represents the available configuration setups and the output space of the function represents the application performance. The application is deployed for some of the available configurations and the relationship between the inputs and the output is obtained either using statistical inference or machine learning techniques. Although Machine Learning can be perceived as a subset of statistical inference, the main difference between them is that purely statistical methods make assumptions regarding the function's behavior (e.g., linear, polynomial, etc.) in contrast to Machine Learning that makes no assumptions on it.

The work in [DPIK18] aims at modeling the performance of distributed applications deployed to cloud infrastructures in order to predict the resource allocation that is required so that the application performs with given SLO requirements. The work focuses on a Video Streaming application that presents two components: a video encoder and a cache component. After running the service for different setups, the authors utilize Support Vector Regression and Polynomial Regression in order to predict the application's throughput and latency for different configuration combinations.

Marathe et al. in [MAJ+17] attempt to overcome one of the problems that are presented with the adoption of the "black-box" paradigm: The selection of the input parameters. As statistical inference assumes that the input space comprises a set of uncorrelated dimensions, this principle is violated for more complex applications since their input dimension may present correlations that are easily identifiable. In order to overcome this limitation, this work presents a modeling attempts that relies o deep-learning. The main idea is that the modeling algorithm per se identifies correlations between dimensions and prune the input space. Moreover, in order to avoid an excessive amount of deployments, this work combines observations gathered at smaller scale, i.e., with pruned number of input dimensions, with fewer observations gathered at higher scale.

Kundu et al. in [KRG+12] demonstrate the efficiency of Neural Networks and Support Vector Machines for modeling the performance of cloud based applications. This work emphasizes on the problem of parameter selection, considering the number of cores, the available memory and the latency of the virtualized I/O device. After an extensive experimental evaluation, the authors demonstrate that the adoption of the two aforementioned Machine Learning learners overcomes traditional regression approaches that produce results of lower quality. Interestingly, their work also proves capable of modeling not only the average performance values, but also the 90th percentile, showcasing that the induced learners provide a robust behavior.

Goncalves et al. in [GCMS15] present a performance inference methodology that aims at improving capacity planning. The main idea of this work is the following: Instead of blindly deploying the application for different configurations, heuristics can be applied in order to reduce the number of deployment that need to be tested in order to extract an accurate profile. Assuming an ordering of application workflows that demonstrates the resource requirements, i.e., $W_1 < W_2$ indicates that $W_2$ requires more resources than $W_1$ in order to meet some given SLO objectives, it is safe to consider that if configuration $C_1$ leeds $W_1$ to failure, then it shall lead $W_2$ to failure as well. The authors showcase that the generalization of this idea leads to pruning of the configuration space of up to 80%.

PANIC [GTPK15] is another work that constructs performance models using Machine Learning and addresses the problem of selecting a set of representative points for training the learner. This approach favors the points that belong to the most steep regions of the Deployment Space, based on the idea that these regions characterize most appropriately the entire performance function. This work is the predecessor of the work presented in Section 2: The idea of favoring regions of the configuration space where application performance is harder to model is common to both works. However, PANIC's feature to solely focus on exploiting the obtained knowledge, overlooking space exploration leads the methodology presented in this thesis to outperform PANIC.

In conclusion, methodologies that belong to the "black-box" category seem to improve all these characteristics that reduce the applicability of simulation and emulation approaches. First of all, they require no application knowledge (hence, "black-boxes") since the structure of the model is dynamically generated during training. Second, their accuracy is only proportional to the number of tested configuration, something that make them perfect for modeling more complex applications. Third, they do not suffer from the problem of latent variables since the model takes everything into consideration: The application structure, the infrastructure, etc. On the contrary, the number of configurations that need to be tested grows rapidly with the application complexity and radically increase the deployment time and cost. Moreover, the extracted profiles are not portable to other infrastructures (since they encapsulate all infrastructure-related information). Nevertheless, the advent of the Machine Learning era and the research improvements it introduced has made "black-box" modeling compelling in comparison to the other methods.

## 4.2 Data-driven Modeling of Analytics Operators

The problem of modeling the outcome of an operator when applied to different datasets has been extensively researched and expressed in different forms. The advent of the Big Data era and the tremendous increase in the computation power and storage capacity that the cloud paradigm has delivered, has gradually shifted the attention of both academia and industry from performance to data efficiency. From this perspective, the success of a newly deployed service is no longer solely affected by the performance measured in terms of traditional computing, i.e., throughput, latency, etc., but is also affected by the quality of the results it produces. This quality is determined by the data it can access. Hence, the problem of selecting these datasets that maximize the utility for a given operator is extremely important for the success of any application.

The diverse applications can be categorized in three categories. *Data Integration* approaches seek for a statistical summarization of the underlying data. Approaches of this category aim at exporting summaries, metadata and any other information that can help data scientists to quickly obtain an initial idea of the datasets. The concept of *Dataspaces*, in particular, can be viewed as a subfield of Data Integration, where the proposed systems aim at solving problems that come up because of the difference in storage locations, indexing, schemas, etc., of different datasets. The second category is *Data Exploration.* Although different works from this category may present radically different characteristics, they share one thing in common: Data Exploration seeks for a subset of a given dataset in order to maximize a given utility metric. Finally, *Feature Selection* techniques aim at finding features of a given dataset that exhibit a desired behavior. Let us now provide some representative research works for each category:

**Data Integration & Dataspaces:** The idea of combining data from different sources in order to increase their utility is the key idea behind *Data Integration.* Since diverse datasets may differ in origin, schema, size, storage location, etc., research works of this category focus on generating an initial statistical view of the underlying datasets and possible encoding this knowledge in a set of metadata. The concept of *Dataspaces* and *Data Lakes* is based on the same principle: Data

Lake Systems also attempt to solve the engineering problem that come up from having a massive number of datasets living in different locations.

Lenzerini et al. in [Len02] outline the requirements for the implementation of data integration systems and provides the theoretical models to express the basic operations over distinct datasets. Different aspects of data integration systems are considered, such as data modeling, reasoning and query answering.

Ground [HSG+17] is a data context service that focuses on the creation of the appropriate metadata that informs the data scientist on the possible use of each dataset. The *context* of the data retains information regarding the representation of the dataset, details about how data was created and versioning history, in order to keep track of data updates.

CrossCat [MSJ+16] focuses on analyzing high dimensional data. It relies on mixture modeling and Bayesian structure learning. It evaluates each data column separately, constructs different views of the data and uses a separate non-parametric mixture to model each view. Although this work attempts to obtain better knowledge for existing datasets, it does not take into consideration the relationships between different datasets.

Abedjan et al. in [AGN15] present different approaches towards data profiling that aim at automatically extracting metadata for given datasets. This metadata can, in turn, be utilized for clustering and categorizing them according to their usage and utility for different operators. Our work is also based on this idea and extends it: Using a unique (i.e., not operator-dependent) knowledge basis, our methodology can infer the outcome of diverse operators when applied to datasets that belong to the same clusters.

In [FHM05], the authors discuss the necessity of adopting a new abstraction mechanism that supports indexing of different datasets to a data engineer through a unified view. The properties that differentiate the distinct datasets, though, should not be eliminated: Different origins, schemas, storage locations, etc., should be respected.

Goods [HKN+16] is the implementation of such a dataspace system, used by Google to index 26B datasets. Goods supports an indexing mechanism that extracts the schema of the unstructured data and 'guesses' the types of specific columns.

To recap, Data Integration is an interesting field that aims to produce meta-knowledge on top of existing datasets in order to provide better insights to data scientists for their content. Our approach, presented in Section 3, also has the same motivation. Nevertheless, our work exceeds the scope of providing static knowledge for the datasets through bridging the gap between the datasets and the operators.

**Data Exploration:** In contrast to Data Integration that seeks for labeling datasets and extracting metadata out of them that summarize their utility, Data Exploration approaches have a different focus. Their objective is to navigate through a massive dataset and isolate a set of tuples that are most suitable for a specific purpose.

Joglekar et al. in [JGMP16] present a "drill-down" mechanism which is an interactive approach that aims at finding a set of "interesting tuples" that best summarize a given dataset. The users can provide custom rules according to the properties that they want to explore and the mechanism will, eventually, return the tuples that best fit the given criteria. Although this work differs from our work since the latter focuses on modeling the dataset space, both works rely on the identification of the factors that best summarize and diversify a dataset.

Similarly, DBExplorer [SCJ16] attempts to provide an "in context" summary of a database, in the sense that a user may possess previous knowledge and want to explore a specific part of a database which is relevant to this knowledge. DBExplorer is the implementation of Conditional Attribute Dependency (CAD) View which showcases the dependencies between specific attribute values according to the user-defined context. Again, the objective of this work is to explore a specific portion of a given database rather than comparing different datasets.

Cafarella et al. in [CHW$^+$08], millions of different databases are compared, collected from a set of raw HTML tables crawled through the Google search engine. The authors compare the schemas of the different databases and apply attribute correlation statistics in order to find similarities. Although the motivation behind this work is different from our work, the necessity of identifying the similarity between different datasets is highlighted.

MacroBase [BGM$^+$17] aims at prioritizing attention to fast data streams. The incoming tuples are classified to inliers and outliers. It, then, attempts to summarize the outliers' properties and provide explanations to the users. This work examines streaming data on the hypothesis that "interest" is gathered around outlying points. In cases where the "interest" changes, one needs to structurally modify the default pipeline, something cumbersome when one needs to analyze the same stream from different viewpoints. On the contrary, our work focuses on data themselves and applies different operators (i.e., the equivalent of MacroBase's concept of "interest") without requiring explicit re-execution of parts of our workflow.

Data Canopy [WWDI17] demonstrates the power of statistical analysis over unknown datasets in order to infer knowledge. The main idea of this work lies on generating a set of *basic aggregates*, which can be synthesized in order to infer knowledge without constantly accessing the raw data. Data Canopy, similarly to our work, highlights the necessity of statistically analyzing the datasets once and utilizing the analysis for consequent tasks. Yet, in this work, we are interested in predicting complex analytics operator performance considering more complex statistical properties for the datasets (such as distribution) than simple aggregates.

AIDE [DPD16] is a database exploration system using Active Learning. It implements an iterative methodology, in which it requests feedback from the users whether the returned tuple is of interest to them. While the users provide more responses, AIDE isolates the interesting areas of the database and returns them to the user in the form of SQL queries. This work is driven by the assumption that the user is unaware of the database and only provides yes/no responses.

On the contrary, our work requires no feedback from the user and the “interesting” datasets are determined by operator performance.

To summarize, Data Exploration is an interesting field with many works that share similar objectives with our work. Nevertheless, our approach differs from traditional Data Exploration approaches in the sense that the output of our methodology, i.e., the Dataset Space, can be reused as is in order to model the output of different operators. Moreover, all the listed approaches are applicable ta tuple-based data; On the contrary, our work is demonstrated to be applicable to graph-based datasets with the utilization of a suitable similarity metric.

**Feature selection:** Finally, feature selection methodologies tackle a similar problem to this work but from a different angle. The main idea behind instance selection is the extraction of a set of features that maximize a given utility metric. In contrast to the approaches listed above, approaches of this category mostly focus on a smart space exploration in order to isolate these dataset features or tuples that best server a particular purpose.

Brainwash [AAB$^+$13] is a system that aims at reducing the *Explore-Extract-Evaluate* loop happening in modern systems that implement or rely on Machine Learning trained classifiers. Brainwash supports the execution of a set of statistical operators in the provided dataset in order to facilitate exploration. Based on the statistical importance of different features, the system proposes a set of features (extraction) and, finally, evaluate the utility of its results through using lazy code execution.

Zhang et al. in [ZKR16] study the problem of computation materialization in feature selection techniques. This interesting work attempts to adopt materialization methodologies that are widely used in the database realm in order to avoid re-computation in feature selection workloads. Interestingly, their work indicates that the adoption of simple, well-known materialization techniques can lead to tremendous speedups in feature selection workflows.

Zombie [AC16] is a system that conducts input selection for feature engineering. The main idea of this work is to focus on the appropriate tuples of a given dataset in order to accelerate feature selection for machine learning tasks. Specifically, Zombie manages to accelerate the “inner-loop” of the feature evaluation loop, i.e., the selection of a subset of the existing features, through applying heuristic techniques that achieve to generate approximate subsets of features with the desired characteristics.

Fu et al. in [FZL13] present an interesting survey that summarizes the works in the area of instance selection for Active Learning. Active Learning [Set10] is utilized for training Machine Learning classifiers when the extraction of labeled training sets is expensive. In this setup, a classifier is trained in an *online* fashion and the labels of the training points are either unknown or hard to be obtained. The objective of these approaches is to provide efficient heuristics to select a subset of points that minimize the classifier’s uncertainty under given cost constraints.

Although feature selection approaches present many commonalities with our work, in the sense that both seek for entities that maximize data utility, their objectives and outcomes are slightly different as feature selection is primarily used as a preprocessing step to Machine Learning algorithms whereas our work is mostly used during exploratory analysis.

CHAPTER 5

# Conclusions and Future Directions

In this thesis, we revisited the problem of modeling the behavior of a Big Data operator from two different angles. First, we addressed the problem of modeling its performance when deployed with different deployment configurations and application level parameters and, second, we proposed a content-based methodology that models its output when it is applied over different datasets. Even though our contributions approach the problem from different perspectives, their combination provide an holistic view of the problem of modeling the behavior of a modern operator.

Our first contribution relates to modeling the operator's performance for different resource configurations. Even though this is a widely studied problem, the proposed research approaches do not take into consideration the ever increasing complexity on the application structure, requiring an increasing number of samples that make profiling cumbersome and cost inefficient. To this end, our research effort was focused on providing a methodology that identifies representative configurations that can lead the model to maximize its accuracy under a given number of sample configurations.

The proposed divide-and-conquer approach proved to be particularly effective for this problem. Considering that by virtue of their design, distributed systems tend to produce performance functions that are easily approximated by linear or piecewise linear functions, the decomposition of the space into more and smaller regions facilitated modeling and allowed focusing on each one separately. Moreover, the consideration of both exploration and exploitation when distributing the deployment budget to the generated subregions allowed us to avoid over fitting to areas with abnormalities without prohibiting us to “zoom-in” as required in order to partition the space in higher detail. Finally, the adoption of oblique Decision Trees increased the expressiveness of the final model with the cost of requiring more computation in order to estimate good split lines. This expressiveness translates to modeling accuracy gains when the deployment budget is relatively small. However, an interesting side-effect of the adoption of oblique split lines is that performance function patterns such as discontinuities, maxima, etc., that involve more than one input dimensions on the same time, are identified faster.

The second contribution of this thesis relates to modeling the operator’s output when it is applied for different datasets. The absence of any ordering relationship between the input datasets makes the problem particularly challenging since traditional statistical modeling methodologies cannot be used. Driven by the observation that similar datasets make an operator present similar similar outputs, our research was focused on creating a dataset metric space that reflects the inter-relationships between them, measured in the light of three fundamental properties, i.e., statistical distribution, dataset size and ordering of tuples.

The suggested scheme achieved to construct highly informative, low dimensional dataset spaces that provide useful insights even with a plain visual examination. The heart of our methodology lies on the selection of appropriate similarity metrics. Even though the information represented by the dataset space is entirely operator agnostic, the properties that need to be examined for extracting dataset similarity must affect the operator under examination in order to provide meaningful models. Interestingly, in our work we isolated three properties that affect 5 different categories of operators. We argue that the data properties of interest are much fewer than the possible operators that can be applied to a datasets. Hence, isolating and combining a handful of key properties that affect many operators can massively accelerate data data analysis. Moreover, the consideration of different types of datasets, such as graphs, highlights the applicability of our scheme for different use cases.

Finally, this work provides a strong foundation for future research. Both modeling methodologies were developed under the hypothesis that the operator has batch characteristics, i.e., it operates with given static data and its runtime behavior receives the same inputs as in the training phase. Nevertheless, this might not be the case: Constantly evolving datasets and new data sources would require an operator with streaming characteristics. This parameter creates a new dimension for modeling: Since deployment reproducibility is now compromised, since the performance measurements for each deployment now have a temporal dimension as well, how could we update the performance modeling methodology to predict the performance for streaming operators? Moreover, for the second part of this thesis, the dynamic introduction of new datasets is mostly tackled with the optimization described in Section 3.3.5. However, how could the data modeling scheme be adjusted in order to accommodate updates to the existing datasets without having to repeat the entire workflow from scratch? This interesting variation could find interesting applications for data with a temporal dimension such as logs, sensor data, etc.

# Publications

## Journals

- I. Giannakopoulos, I. Konstantinou, D. Tsoumakos and N. Koziris: Cloud application deployment with transient failure recovery. Journal of Cloud Computing. Advances, Systems and Applications, 2018, 7:11

## Conferences

- I. Giannakopoulos, D. Tsoumakos and N. Koziris: A Content-Based Approach for Modeling Analytics Operators. In Proceedings of the 2018 Conference on Information and Knowledge Management (CIKM 2018), October 22-26, 2018, Lingotto, Turin, Italy.

- F. Loukidis-Andreou, I. Giannakopoulos, K. Doka, N. Koziris: Docker-Sec: A Fully Automated Container Security Enhancement Mechanism. In Proceedings of the 38th IEEE International Conference on Distributed Computing Systems (ICDCS 2018), Demo track, 2-5 July, 2018, Vienna, Austria.

- I. Giannakopoulos, D. Tsoumakos and N. Koziris: Towards an Adaptive, Fully Automated Performance Modeling Methodology for Cloud Applications. In Proceedings of the IEEE International Conference on Cloud Engineering (IC2E 2018), April 17-20, Orlando, Florida, USA.

- I. Giannakopoulos, D. Tsoumakos and N. Koziris: A Decision Tree Based Approach Towards Adaptive Modeling of Big Data Applications. In Proceedings of the 2017 IEEE International Conference on Big Data (BigData 2017), December 11-14, 2017, Boston, MA, USA.

- I. Giannakopoulos, K. Papazafeiropoulos, K. Doka and N. Koziris: Isolation in Docker through Layer Encryption. In Proceedings of the 37th IEEE International Conference on Distributed Computing Systems (ICDCS 2017), Demo track, 5-8 June, 2017, Atlanta, GA, USA.

- I. Giannakopoulos, I. Konstantinou, D. Tsoumakos and N. Koziris: AURA: Recovering from Transient Failures in Cloud Deployments (demo paper) . In Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2017), May 14-17, 2017, Madrid, Spain.

- N. Korasidis, I. Giannakopoulos, K. Doka, D. Tsoumakos and N. Koziris: Fair, Fast and Frugal Large-Scale Matchmaking for VM Placement. In Proceedings of the 2nd International Workshop on Algorithmic Aspects of Cloud Computing (ALGOCLOUD 2016), in conjunction with the ALGO 2016 Conference, August 22, 2016, Aarhus, Denmark.

- I. Giannakopoulos, I. Konstantinou, D. Tsoumakos and N. Koziris: Recovering from Cloud Application Deployment Failures through Re-execution. In Proceedings of the 2nd International Workshop on Algorithmic Aspects of Cloud Computing (ALGOCLOUD 2016), in conjunction with the ALGO 2016 Conference, August 22, 2016, Aarhus, Denmark.

- I. Giannakopoulos, P. Karras, D. Tsoumakos, K. Doka and N. Koziris: An Equitable Solution to the Stable Marriage Problem. In Proceedings of the 27th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2015), 9-11 November, Vietri sul Mare, Italy.

- I. Mytilinis, I. Giannakopoulos, I. Konstantinou, K. Doka, D. Tsitsigkos, M. Terrovitis, L. Giampouras and N. Koziris: MoDisSENSE: A Distributed Spatio-Temporal and Textual Processing Platform for Social Networking Services. In Proceedings of the 2015 ACM SIGMOD/PODS International Conference on Management of Data (Demo Track), May 31-June 4, 2015, Melbourne, Australia.

- I. Giannakopoulos, D. Tsoumakos, N. Papailiou and N. Koziris: PANIC: Modeling Application Performance over Virtualized Resources. In Proceedings of the 2015 IEEE International Conference on Cloud Engineering (IC2E 2015), 9-13 March, Tempe, AZ, USA.

- I. Mytilinis, I. Giannakopoulos, I. Konstantinou, K. Doka and N. Koziris: MoDisSENSE: A Distributed Platform for Social Networking Services over Mobile Devices. In Proceedings of the 2014 IEEE International Conference on Big Data (BigData 2014), 27-30 October, Washington DC, USA.

- I. Giannakopoulos, N. Papailiou, C. Mantas, I. Konstantinou, D. Tsoumakos and N. Koziris: CELAR: Automated Application Elasticity Platform. In Proceedings of the 2014 IEEE International Conference on Big Data (BigData 2014), 27-30 October, Washington DC, USA.

# Bibliography

[10v] The 10 Vs of Big Data. https://tdwi.org/articles/2017/02/08/10-vs-of-big-data.aspx.

[42v] The 42 V's of Big Data and Data Science. https://www.kdnuggets.com/2017/04/42-vs-big-data-data-science.html.

[AAB+13] Michael R Anderson, Dolan Antenucci, Victor Bittorf, Matthew Burgess, Michael J Cafarella, Arun Kumar, Feng Niu, Yongjoo Park, Christopher Ré, and Ce Zhang. Brainwash: A data system for feature engineering. In *CIDR*, 2013.

[AC16] Michael R Anderson and Michael Cafarella. Input selection for fast feature engineering. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 577–588. IEEE, 2016.

[AFG+10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of Cloud Computing. *Communications of the ACM*, 53(4):50–58, 2010.

[AFGo09] Michael Armbrust, Armando Fox, Rean Griffith, and Joseph others. Above the clouds: A berkeley view of cloud computing. Technical report, Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[AGN15] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. Profiling relational data: a survey. *The VLDB Journal—The International Journal on Very Large Data Bases*, 24(4):557–581, 2015.

[AI13] Nrusimham Ammu and Mohd Irfanuddin. Big data challenges. *International Journal of Advanced Trends in Computer Science and Engineering*, 2(1):613–615, 2013.

[Alt92] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[ans] Ansible. https://www.ansible.com/. Accessed December 11th, 2017.

[ARB12] A-F Antonescu, Philip Robinson, and Torsten Braun. Dynamic Topology Orchestration for Distributed Cloud-Based Applications. In *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on*, pages 116–123. IEEE, 2012.

[auf] AUFS. http://aufs.sourceforge.net/. Accessed December 11th, 2017.

[aur] AURA Source Code. https://github.com/giagiannis/aura/. Accessed December 11th, 2017.

[AV07] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[awsa] AWS CloudFormation Documentation. https://docs.aws.amazon.com/AWSCloudFormation latest/UserGuide/troubleshooting.html. Accessed March 31st, 2018.

[awsb] AWS Elastic Load Balancing Documentation. https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/ts-elb-error-api-response.html. Accessed March 31st, 2018.

[awsc] AWS Incident. https://goo.gl/f959fl. Accessed December 11th, 2017.

[awsd] AWS Maintenance. https://aws.amazon.com/maintenance-help/. Accessed December 11th, 2017.

[B$^+$08] Dhruba Borthakur et al. Hdfs architecture guide. *Hadoop Apache Project*, 53, 2008.

[BA99] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[BC01] Sergio Bermejo and Joan Cabestany. Oriented principal component analysis for large margin classifiers. *Neural Networks*, 14(10):1447–1461, 2001.

[BdW12] Gergana Bounova and Olivier de Weck. Overview of metrics and their correlation patterns for multiple-metric topology analysis on heterogeneous graph ensembles. *Phys. Rev. E*, 85:016117, 2012.

[BE05] Ulrik Brandes and Thomas Erlebach. *Network Analysis: Methodological Foundations (Lecture Notes in Computer Science).* Springer-Verlag New York, Inc., 2005.

[bea] AWS Elastic BeanStalk. http://aws.amazon.com/elasticbeanstalk/. Accessed December 11th, 2017.

[Ber14] David Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, 2014.

[BFSO84] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees.* CRC press, 1984.

[BGM$^+$17] Peter Bailis, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. Macrobase: Prioritizing attention in fast data. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 541–556. ACM, 2017.

[BGRS17] Peter Bailis, Edward Gan, Kexin Rong, and Sahaana Suri. Prioritizing attention in fast data: Principles and promise. *CIDR*, 2017.

[BGTK18] Tasos Bakogiannis, Giannis Giannakopoulos, Dimitrios Tsoumakos, and Nectarios Koziris. A Similarity-based Approach to Modeling Graph Operators. *arXiv preprint arXiv:1802.05536*, 2018.

[BJRL15] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control.* John Wiley & Sons, 2015.

[BKNS00] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *ACM sigmod record.* ACM, 2000.

[blo14] The Big Problem Is Medium Data. http://goo.gl/5nYrrz, 2014.

[Bon87] Phillip Bonacich. Power and centrality: A family of measures. *American Journal of Sociology*, 92(5):1170–1182, 1987.

[Bow81] Adrian Bowyer. Computing dirichlet tessellations. *The computer journal*, 24(2):162–166, 1981.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[BWZ15] Len Bass, Ingo Weber, and Liming Zhu. *DevOps: A Software Architect's Perspective.* Addison-Wesley Professional, 2015.

[BY13] Ricardo A Baeza-Yates. Big Data or Right Data? In *AMW*, 2013.

[cfe] CFEngine. https://cfengine.com/. Accessed March 31st, 2018.

[Cha78] Chris Chatfield. The holt-winters forecasting procedure. *Applied Statistics*, pages 264–279, 1978.

[che] Chef. https://www.chef.io/chef/. Accessed December 11th, 2017.

[CHW$^+$08] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: Exploring the Power of Tables on the Web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008.

[cloa] AWS CloudFormation. http://aws.amazon.com/cloudformation/. Accessed December 11th, 2017.

[clob] CloudFoundry. https://www.cloudfoundry.org/. Accessed December 11th, 2017.

[cloc] CloudHarmony. https://cloudharmony.com/.

[clod] CloudSpectator. http://cloudspectator.com/.

[clu15] Google Cluster Monitoring Dataset. https://github.com/google/cluster-data/, 2015. Online; accessed Feb 2018.

[CMS16] M Cunha, NC Mendonça, and A Sampaio. Cloud Crawler: a declarative performance evaluation environment for infrastructure-as-a-service clouds. *Concurrency and Computation: Practice and Experience*, 2016.

[CRB$^+$11] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1), 2011.

[CRM00] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Real-time tracking of non-rigid objects using mean shift. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 142–149. IEEE, 2000.

[CST+10] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010.

[DL06] Clara Dismuke and Richard Lindrooth. Ordinary least squares. *Methods and Designs for Outcomes Research*, 93, 2006.

[DPD16] Kyriaki Dimitriadou, Olga Papaemmanouil, and Yanlei Diao. Aide: An active learning-based approach for interactive data exploration. *IEEE Transactions on Knowledge and Data Engineering*, 28(11):2842–2856, 2016.

[DPIK18] Sevil Dräxler, Manuel Peuster, Marvin Illian, and Holger Karl. Towards predicting resource demands and performance of distributed cloud services. *KuVS-Fachgespräch Fog Computing 2018*, page 9, 2018.

[DVJ+15] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, et al. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015.

[EK+96] Martin Ester, Hans-Peter Kriegel, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, 1996.

[ela] AWS Elastic Load Balancing. http://aws.amazon.com/elasticloadbalancing/. Accessed December 11th, 2017.

[FAK+12] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '12, New York, NY, USA, 2012. ACM.

[FFH12] Florian Fittkau, Sören Frey, and Wilhelm Hasselbring. CDOSim: Simulating cloud deployment options for software migration support. In *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2012 IEEE 6th International Workshop on the*. IEEE, 2012.

[FHM05] Michael Franklin, Alon Halevy, and David Maier. From Databases to Dataspaces: a new Abstraction for information management. *ACM Sigmod Record*, 34(4):27–33, 2005.

[Fre77] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[FS99] Yoav Freund and Robert E Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

[FZL13] Yifan Fu, Xingquan Zhu, and Bin Li. A survey on instance selection for active learning. *Knowledge and information systems*, 35(2):249–283, 2013.

[G$^+$15] NIST Big Data Public Working Group et al. Nist big data interoperability framework. *Special Publication*, pages 1500–6, 2015.

[GCMS15] Marcelo Gonçalves, Matheus Cunha, Nabor C Mendonca, and Américo Sampaio. Performance Inference: A Novel Approach for Planning the Capacity of IaaS Cloud Applications. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. IEEE, 2015.

[GDG$^+$18] Swarnendu Ghosh, Nibaran Das, Teresa Gonçalves, Paulo Quaresma, and Mahantapas Kundu. The journey of graph kernels through two decades. *Computer Science Review*, 27:88–111, 2018.

[Gei93] Seymour Geisser. *Predictive inference*, volume 55. CRC press, 1993.

[GFW03] Thomas Gärtner, Peter A. Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003, Proceedings*, pages 129–143, 2003.

[GKTK16] Ioannis Giannakopoulos, Ioannis Konstantinou, Dimitrios Tsoumakos, and Nectarios Koziris. Recovering from cloud application deployment failures through re-execution. In *International Workshop of Algorithmic Aspects of Cloud Computing*, pages 117–130. Springer, 2016.

[GKTK17] Ioannis Giannakopoulos, Ioannis Konstantinou, Dimitrios Tsoumakos, and Nectarios Koziris. Aura: Recovering from transient failures in cloud deployments. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 762–765. IEEE Press, 2017.

[goo] Google App Engine Incident. https://goo.gl/ICI0Mo.

[Gow66] John C Gower. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, pages 325–338, 1966.

[GPM$^+$14] Ioannis Giannakopoulos, Nikolaos Papailiou, Christos Mantas, Ioannis Konstantinou, Dimitrios Tsoumakos, and Nectarios Koziris. Celar: automated application elasticity platform. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 23–25. IEEE, 2014.

[GTK17] Ioannis Giannakopoulos, Dimitrios Tsoumakos, and Nectarios Koziris. A Decision Tree Based Approach Towards Adaptive Profiling of Distributed Applications (Extended Version). *arXiv preprint arXiv:1704.02855*, 2017.

[GTK18a] Ioannis Giannakopoulos, Dimitrios Tsoumakos, and Nectarios Koziris. A Content-Based Approach for Modeling Analytics Operators. In *International Conference on Information and Knowledge Management (CIKM) 2018*. ACM, 2018.

[GTK18b] Ioannis Giannakopoulos, Dimitrios Tsoumakos, and Nectarios Koziris. Towards an Adaptive, Fully Automated Performance Modeling Methodology for Cloud Applications. In *Cloud Engineering (IC2E), 2018 IEEE International Conference on*, pages 148–158. IEEE, 2018.

[GTPK15] Ioannis Giannakopoulos, Dimitrios Tsoumakos, Nikolaos Papailiou, and Nectarios Koziris. PANIC: Modeling Application Performance over Virtualized Resources. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on*. IEEE, 2015.

[GVCL14] Marisol García-Valls, Tommaso Cucinotta, and Chenyang Lu. Challenges in real-time virtualization and predictable cloud computing. *Journal of Systems Architecture*, 60(9), 2014.

[had] Hadoop AURA application description. https://github.com/giagiannis/aura/ tree/-master/example/hadoop. Accessed December 11th, 2017.

[heaa] Openstack Heat. https://wiki.openstack.org/wiki/Heat.

[heab] Openstack Heat Signaling and Coordination. https://wiki.openstack.org/wiki/Heat/Signaling-And-Coordination. Accessed March 31st, 2018.

[her] Heroku. https://www.heroku.com/. Accessed December 11th, 2017.

[HFH$^+$09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1), 2009.

[HHD+11] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *New Frontiers in Information and Software as Services*. Springer, 2011.

[HHD16] Oliver Hanappi, Waldemar Hummer, and Schahram Dustdar. Asserting reliable convergence for configuration management scripts. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 328–343. ACM, 2016.

[HKN+16] Alon Halevy, Flip Korn, Natalya F Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. Goods: Organizing Google's Datasets. In *Proceedings of the 2016 International Conference on Management of Data*, pages 795–806. ACM, 2016.

[HKS93] David Heath, Simon Kasif, and Steven Salzberg. Induction of oblique decision trees. In *IJCAI*, 1993.

[HM11] Javier Martin Hernández and Piet Van Mieghem. Classification of graph metrics. pages 1–20, 2011.

[HP94] John S Heidemann and Gerald J Popek. File-system development with stackable layers. *ACM Transactions on Computer Systems (TOCS)*, 12(1):58–89, 1994.

[HPE+06] Kenneth Hoste, Aashish Phansalkar, Lieven Eeckhout, Andy Georges, Lizy K John, and Koen De Bosschere. Performance prediction based on inherent program similarity. In *Proceedings of the 15th international conference on Parallel architectures and compilation techniques*. ACM, 2006.

[HROE13] Waldemar Hummer, Florian Rosenberg, Fábio Oliveira, and Tamar Eilam. Testing idempotence for infrastructure as code. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 368–388. Springer, 2013.

[HSG+17] Joseph M Hellerstein, Vikram Sreekanti, Joseph E Gonzalez, James Dalton, Akon Dey, Nag, et al. Ground: A data context service. In *CIDR*, 2017.

[JD11a] Gideon Juve and Ewa Deelman. Automating application deployment in infrastructure clouds. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 658–665. IEEE, 2011.

[JD11b] Gideon Juve and Ewa Deelman. Wrangler: Virtual cluster provisioning for the cloud. In *Proceedings of the 20th international symposium on High performance distributed computing*, pages 277–278. ACM, 2011.

[Jef05] TH Jeff. Introduction to neural networks with java, heaton research, 2005.

[JGMP16] Manas Joglekar, Hector Garcia-Molina, and Aditya Parameswaran. Interactive data exploration with smart drill-down. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 906–917. IEEE, 2016.

[JKMvD06] A. Jamakovic, R. E. Kooij, P. Van Mieghem, and E. R. van Dam. Robustness of networks against viruses: the role of the spectral radius. In *2006 Symposium on Communications and Vehicular Technology*, pages 35–38, 2006.

[Jol86] Ian T Jolliffe. Principal component analysis and factor analysis. In *Principal component analysis*, pages 115–128. Springer, 1986.

[JU08] Almerima Jamakovic and Steve Uhlig. On the relationships between topological measures in real-world networks. *NHM*, 3(2):345–359, 2008.

[juj] Juju. https://juju.ubuntu.com/. Accessed December 11th, 2017.

[Ken48] Maurice George Kendall. Rank correlation methods. 1948.

[KJM$^+$14] Rakesh Kumar, Kanishk Jain, Hitesh Maharwal, Neha Jain, and Anjali Dadhich. Apache cloudstack: Open source infrastructure as a service cloud computing platform. *Proceedings of the International Journal of advancement in Engineering technology, Management and Applied Science*, pages 111–116, 2014.

[Kni66] William R Knight. A computer method for calculating kendall's tau with ungrouped data. *Journal of the American Statistical Association*, 61(314):436–439, 1966.

[KP14] Kailash C Kapur and Michael Pecht. *Reliability engineering*. John Wiley & Sons, 2014.

[KRDZ10] Sajib Kundu, Raju Rangaswami, Kaushik Dutta, and Ming Zhao. Application performance modeling in a virtualized environment. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*. IEEE, 2010.

[KRG$^+$12] Sajib Kundu, Raju Rangaswami, Ajay Gulati, Ming Zhao, and Kaushik Dutta. Modeling virtualized applications using machine learning techniques. In *ACM SIGPLAN Notices*, volume 47. ACM, 2012.

[Kru58] William H Kruskal. Ordinal measures of association. *Journal of the American Statistical Association*, 53(284):814–861, 1958.

[KT15] Yasuharu Katsuno and Hitomi Takahashi. An automated parallel approach for rapid deployment of composite application servers. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, pages 126–134. IEEE, 2015.

[LBG80] Yoseph Linde, Andres Buzo, and Robert Gray. An algorithm for vector quantizer design. *IEEE Transactions on communications*, 28(1):84–95, 1980.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[LC94] David D Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the eleventh international conference on machine learning*, 1994.

[Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM, 2002.

[Lic13] Lichman. Uci machine learning repository, 2013.

[LJ12] Alexandros Labrinidis and Hosagrahar V Jagadish. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, 5(12):2032–2033, 2012.

[LK14] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[LMVdMF11] Changbin Liu, Yun Mao, Jacobus Van der Merwe, and Mary Fernandez. Cloud resource orchestration: A data-centric approach. In *Proceedings of the biennial Conference on Innovative Data Systems Research (CIDR)*, pages 1–8, 2011.

[loa] LoadImpact. https://loadimpact.com/.

[Loh12] Steve Lohr. The age of big data. *New York Times*, 11(2012), 2012.

[LS16] Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.

[LYKZ10] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. CloudCmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010.

[LZK+11] Ang Li, Xuanran Zong, Srikanth Kandula, Xiaowei Yang, and Ming Zhang. CloudProphet: towards application performance prediction in cloud. In *ACM SIGCOMM Computer Communication Review*, volume 41. ACM, 2011.

[LZZ+10] Zhichun Li, Ming Zhang, Zhaosheng Zhu, Yan Chen, Albert G Greenberg, and Yi-Min Wang. WebProphet: Automating Performance Prediction for Web Services. In *NSDI*, volume 10, 2010.

[MAJ+17] Aniruddha Marathe, Rushil Anirudh, Nikhil Jain, Abhinav Bhatele, Jayaraman Thiagarajan, Bhavya Kailkhura, Jae-Seung Yeom, Barry Rountree, and Todd Gamblin. Performance modeling under resource constraints using deep transfer learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 31. ACM, 2017.

[Mak93] Spyros Makridakis. Accuracy measures: theoretical and practical concerns. *International Journal of Forecasting*, 9(4):527–529, 1993.

[MHH+10] David R Miller, Shon Harris, Allen Harper, Stephen VanDyke, and Chris Blask. *Security Information and Event Management (SIEM) Implementation (Network Pro Library).* McGraw Hill, 2010.

[MSJ+16] Vikash Mansinghka, Patrick Shafto, Eric Jonas, Cap Petschulat, Max Gasner, and Joshua B Tenenbaum. Crosscat: a fully bayesian nonparametric method for analyzing heterogeneous, high dimensional data. *The Journal of Machine Learning Research*, 17(1):4760–4808, 2016.

[MTK+15] Ioannis Mytilinis, Dimitrios Tsoumakos, Verena Kantere, Anastassios Nanos, and Nectarios Koziris. I/O Performance Modeling for Big Data Applications over Cloud Infrastructures. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on.* IEEE, 2015.

[MWS+07] Michael P Mesnier, Matthew Wachs, Raja R Simbasivan, Julio Lopez, James Hendricks, Gregory R Ganger, and David R O'hallaron. //Trace: parallel trace replay with approximate causal events. USENIX, 2007.

[NG04] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, 2004.

[noa16] National Centers for Environmental Information. https://www1.ncdc.noaa.gov/pub/data/noaa/, 2016. Online; accessed May 2017.

[per] PerfKitBenchmarker. https://goo.gl/b4Xcij.

[Pet16] Peter Schlampp. Spark takes on the big security threats. http://www.ibmbigdatahub.com/blog/spark-takes-big-security-threats, 2016.

[PJ13] Rahul Potharaju and Navendu Jain. When the network crumbles: An empirical study of cloud network failures and their impact on services. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 15. ACM, 2013.

[PKB03] Philip M Papadopoulos, Mason J Katz, and Greg Bruno. Npaci rocks: Tools and techniques for easily deploying manageable linux clusters. *Concurrency and Computation: Practice and Experience*, 15(7-8):707–725, 2003.

[PRS09] Pankesh Patel, Ajith H Ranabahu, and Amit P Sheth. Service level agreement in cloud computing. 2009.

[pup] Puppet. https://puppet.com/. Accessed December 11th, 2017.

[Qui86] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1), 1986.

[Qui96] J Ross Quinlan. Bagging, boosting, and c4. 5. In *AAAI/IAAI, Vol. 1*, 1996.

[RBM13] Ohad Rodeh, Josef Bacik, and Chris Mason. Btrfs: The linux b-tree filesystem. *ACM Transactions on Storage (TOS)*, 9(3):9, 2013.

[rig] RightScale 2017 State of the Cloud Report. https://www.rightscale.com/lp/2017-state-of-the-cloud-report.

[Rok10] Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2), 2010.

[sah] Openstack Sahara. https://wiki.openstack.org/wiki/Sahara. Accessed December 11th, 2017.

[sal] SaltStack. https://saltstack.com/. Accessed March 31st, 2018.

[Sam69] John W Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on computers*, 100(5):401–409, 1969.

[SB98] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 1998.

[SB15] Mahito Sugiyama and Karsten M. Borgwardt. Halting in random walk kernels. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1639–1647, 2015.

[SCDG$^+$17] Tiago A Schieber, Laura Carpi, Albert Díaz-Guilera, Panos M Pardalos, Cristina Masoller, and Martín G Ravetti. Quantification of network structural dissimilarities. *Nature communications*, 8:13928, 2017.

[SCJ16] Manish Singh, Michael J. Cafarella, and H. V. Jagadish. DBExplorer: Exploratory Search in Databases. In *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016.*, pages 89–100, 2016.

[Set10] Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66), 2010.

[sou18] Data Profiler source code. https://github.com/giagiannis/data-profiler, 2018.

[sto17] Google Finance API. https://www.google.com/finance/historical, 2017.

[T$^+$13] Alexandru Adrian Tole et al. Big data challenges. *Database systems journal*, 4(3):31–40, 2013.

[tab17] Medium Data is the New Sweet Spot. https://goo.gl/mnxnEx, 2017.

[TÖ09] Yingying Tao and M Tamer Özsu. Efficient decision tree construction for mining time-varying data streams. In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*. IBM Corp., 2009.

[vag] Vagrant. https://www.vagrantup.com/. Accessed December 11th, 2017.

[vcl] VMware vCloud Automation Center Documentation Center. http://goo.gl/YkKNic. Accessed December 11th, 2017.

[VLA87] Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated Annealing: Theory and Applications*. Springer, 1987.

[VSKB10] S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.

[WCB10] Bhathiya Wickremasinghe, Rodrigo N Calheiros, and Rajkumar Buyya. Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. IEEE, 2010.

[WF15] Felix Wortmann and Kristina Flüchter. Internet of things. *Business & Information Systems Engineering*, 57(3):221–224, 2015.

[Whi12] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[wik17] Web Traffic Time Series Forecasting. https://www.kaggle.com/c/web-traffic-time-series-forecasting/data, 2017. Online; accessed Feb 2018.

[wor] Wordpress AURA application description. https://github.com/giagiannis/aura/tree/master/example/wordpress. Accessed December 11th, 2017.

[WWDI17] Abdul Wasay, Xinding Wei, Niv Dayan, and Stratos Idreos. Data canopy: Accelerating exploratory statistical analysis. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 557–572. ACM, 2017.

[YLMW06] Tynia Yang, Jinze Liu, Leonard McMillan, and Wei Wang. A Fast Approximation to Multidimensional Scaling. In *IEEE workshop on Computation Intensive Methods for Computer Vision*, 2006.

[YMTU14] Yoji Yamato, Masahito Muroi, Kentaro Tanaka, and Mitsutomo Uchimura. Development of template management technology for easy deployment of virtual resources on openstack. *Journal of Cloud Computing*, 3(1):7, 2014.

[ZCWF14] Ennan Zhai, Ruichuan Chen, David Isaac Wolinsky, and Bryan Ford. Heading off correlated failures through independence-as-a-service. In *OSDI*, pages 317–334, 2014.

[ZKR16] Ce Zhang, Arun Kumar, and Christopher Ré. Materialization optimizations for feature selection workloads. *ACM Transactions on Database Systems (TODS)*, 41(1):2, 2016.

APPENDIX A

# Cloud Application Deployment with Transient Failure Recovery

## A.1 Introduction

The advent of the Cloud computing [AFG+10] era has been a key enabler for the migration of many applications from traditional, on-premise servers to public clouds, in order to fully exploit the advantages of the latter: Seemingly infinite resources, billed in a pay-as-you-go manner allow the cloud users to not scale their applications in a cost-effective way. The ability to dynamically allocate and utilize new virtualized resources has liberated the cloud users from the burden of managing the physical infrastructure, since the cloud provider itself is responsible for maintenance and any administrative tasks. The dynamic nature of the resources, though, gave birth to new requirements: Applications need to be deployed and utilize the resources in an automated manner, without requiring human intervention. This concept is the cornerstone behind *automation*, i.e., the ability to run complex tasks that entail resource provisioning and software configuration in a fully automated manner. Automation is an integral part of the Cloud, that helped traditional system administration principles to evolve in order to consider dynamic infrastructures and virtualized resources [BWZ15, KP14].

Especially during the application deployment phase, automation is essential in order to guarantee that different software components such as the cloud's software stack, the Virtual Machines

(VMs), external services, etc., will cooperate in a synchronous manner so as to successfully deploy a given application to the cloud. This challenging task has been both an active research field [HHD16, HROE13, KT15, JD11a, PKB03, DVJ$^+$15, YMTU14, ARB12] and the objective of many production systems, operated by modern cloud providers [heaa, sah, juj, cloa]. These approaches differ in various aspects: Some of them specialize to specific applications (e.g., Openstack Sahara [sah] focuses on deploying data processing systems to Openstack) whereas others [JD11a, JD11b] support an application description language and allow the user to define the application structure. Moreover, some tools operate on specific providers (e.g., [juj]), whereas other retain a cloud-agnostic profile and abstract the provider specific details from the application description (e.g., [vag]). In spite of their differences, these systems share the same objective: They expect an application description along with any other runtime configuration option and they are responsible to allocate new resources, provision them and configure a new application instance in a fully automated manner.

However, achieving a fully automated application deployment assumes that all the components participating to it (e.g., cloud compute, storage and metadata services, Virtual Machines (VMs), external services, etc.) function in a coordinated and failure-free manner. Even if coordination between different modules is achievable, failures are not rare: Network glitches, request timeouts due to delays in VM bootstrapping, etc., are commonly encountered. Moreover, even the most popular cloud providers, often encounter severe infrastructure failures [awsc, goo] that, according to their severity, may lead to service failures for considerable amount of time (spanning from seconds up to several minutes). A key characteristic of this type of errors is their *transient* nature: They appear for a short time period and vanish without any obvious, from the application's viewpoint, reason. In most cases, the appearance of such errors may be tolerable. For example, during a network glitch, the application may lose some requests, but after the glitch disappears the requests can be easily repeated and the application will be completely functional again. However, during the *sensitive* application deployment phase, such glitches may lead an entire deployment to failure, requiring human intervention either to terminate the allocated VM instances or to manually fix the failed deployment parts.

To tackle this challenge, a crucial observation can be made: When such a *transient* failure occurs during the deployment phase, in many cases, one only needs to repeat the deployment scripts that failed in order to overcome it. For example, take the case of installing new software packages to a newly allocated VM: If one wants to download a software package (e.g., through `apt`) and a network glitch occurs, all one has to do in order to overcome this error is to repeat the download command until the glitch vanishes. Evidently, the transient nature implies that the cloud user does not have control over it. Hence, an optimistic error-recovery policy would aim at repeating the script execution until the error disappears.

Given this crucial observation, in this work we propose a cloud application deployment methodology that aims at identifying the parts of an application deployment that failed due to a transient failure and repeat them until the deployment is successfully accomplished. Specifically, building upon previous work [GKTK16, GKTK17], we propose a deployment model, perfectly suited for distributed applications with multiple software components deployed to different VMs. The application description is serialized to a Directed Acyclic Graph (DAG) that describes the dependencies between different application modules. The application deployment effectively leads to the traversal of the dependency DAG in a specific order, in order to satisfy the module dependencies. In case of *transient* errors, our methodology first examines the DAG and isolates the part of it that failed and, then, executes the failed scripts until the transient errors vanish. In order to ensure that the re-executed deployment parts (i.e., *deployment scripts*) always have the same effects (i.e., they are *idempotent*), we adopt a lightweight filesystem snapshot mechanism that ensures that each configuration script can be re-executed as many times as required, until the error vanishes.

The contributions of this work can be, thus, summarized as follows:

- We propose a powerful deployment model, which is capable of efficiently expressing the configuration actions that need to take place between different application modules, in the form of a Directed Acyclic Graph.
- Using this model, we formulate the application deployment problem as a DAG traversal problem and suggest an efficient algorithm for identifying the scripts that failed, along with their respective dependencies.
- We suggest a lightweight filesystem snapshot mechanism in order to ensure that the configuration scripts are idempotent and, hence, can be re-executed as many times as needed.
- We offer AURA, an open-source Python prototype of the proposed methodology [aur], which is capable of issuing application deployments to Openstack instances.
- Finally, we provide illustrative examples of our deployment model for various popular real-world applications and thoroughly evaluate the performance of our prototype through deploying them in a private Openstack cluster.

Our evaluation demonstrated that the proposed approach is capable of deploying diverse applications with different structures in cloud environments that exhibit high error probabilities (reaching up to 0.8), while inserting minimal performance overhead for ensuring the deployment script idempotency. Moreover, the efficiency of the proposed approach is showcased to be increasing with the application structure complexity, as more complex applications are more susceptible to transient failures since they occupy more resources.

## A.2 Related Work

Since the emergence of the cloud computing era, the challenge of fully automated software configuration and resource provisioning has attracted a lot of interest, both by the academia and the industry. The importance of automation was diagnosed early and, thus, many solutions have been proposed and are currently offered to the cloud users. We now briefly discuss the distinct approaches and outline their properties.

**Industrial Systems:** There exist several tools and systems offered by modern cloud providers to their users that aim at providing fully automated application deployment. In the Openstack ecosystem, Heat [heaa] is an orchestration system which aims at managing an application throughout its lifecycle. The users submit application descriptions (named HOT, i.e., Heat Orchestration Template), where they define: The application modules, the resources each module will occupy, the dependencies between different modules (if any) and any other runtime parameter (e.g., name of the SSH key to use, flavor id, etc.). In order to maximize reusability, Heat templates can be parameterized, abstracting the installation-specific details (e.g., image/flavor IDs, key names, etc.) from the application description. Sahara [sah] is a different deployment tool for Openstack that specializes in provisioning Data Processing clusters, i.e., Hadoop and Spark clusters. Sahara uses Heat as the deployment engine and differs from it as it enables the users to provide Hadoop-specific deployment parameters (e.g., HDFS replication size, number of slaves, etc.) and applies them to the cluster. The AWS counterpart of Openstack Heat is the AWS CloudFormation [cloa]. Similar to Heat, CloudFormation receives templates that describe what resources will be utilized and by which components. The cloud user can then define a set of parameters (keys, image ids, etc.) and launch the deployment. AWS Elastic Beanstalk [bea] specializes in deploying web applications to the AWS cloud, hiding the infrastructure internals and automating the provisioning of load balancers, application-level monitoring, etc.

The aforementioned systems are implemented and offered as components of the cloud software stacks they operate on. However, there exist many systems that operate *outside* the target cloud and are capable of deploying to different providers. Vagrant [vag] is an open source tool for building and maintaining portable virtual software development environments. It is mainly used during the development phase of a system which will be deployed to a cloud provider and its main objective is to simplify the software configuration process. Juju [juj] is another open source system, developed and maintained by Canonical that works on top of Ubuntu images. According to Juju, each application comprise of building blocks named *Charms*. When deploying an application, each Charm is deployed independently and, finally, the different Charms are unified through passing parameters to each other. Finally, CloudFoundry [clob] and Heroku [her] are two systems that focus on providing a more platform-oriented view of the cloud, i.e., Platform-as-a-Service (PaaS) semantics on top of Infrastructure-as-a-Service (IaaS) clouds. The difference between them and the previous solutions is that both CloudFoundry and Heroku deploy applications to different providers but their objective is to provide access to the platform level, rather than the VM level, decoupling this way the underlying virtualized hardware from the deployed application.

All the aforementioned systems generate a dynamic landscape of deployment tools, with diverse characteristics and different strengths. Nevertheless, none of these tools considers the dynamic and, frequently, error-prone nature of the cloud, as none of them provides a mechanism of overcoming transient errors in a fully automated manner, as they require human intervention either to resume the deployment or trigger a new one. For example, CloudFormation's official documentation suggests to manually continue rolling back an update when a Resource is not stabilized due to an exceeded timeout period [awsa]. A similar suggestion is also made for the Elastic Load Balancing [ela] component, extensively utilized both by CloudFormation and Beanstalk instances: In case where a delay in certificate propagation is encountered, the users are advised to wait for some minutes and retry to setup a new load balancer [awsb].

**Research Approaches:** The complexity on the structure of modern applications comprising different software modules has, inevitably, complicated their deployment to cloud infrastructures and has been the center of research from different works. NPACI Rocks [PKB03] attempts to automate the provisioning of high-performance computing clusters in order to simplify software installation and version tracking. Although this approach was suggested prior to the wide adoption of the cloud and focuses on the HPC world, it is one of the first works that discusses the problem of resource provisioning and software configuration at a big scale and proposes a tool set in order to automate common tasks. Wrangler [JD11a] is a generic deployment system that receives application descriptions in XML format and deploys them to different cloud providers (Amazon EC2, OpenNebula and Eucalyptus). Each description comprises different *plugins*, i.e., deployment scripts, executed on different VMs in order to install a particular software component, e.g., a monitoring plugin, a database plugin, etc. Antonescu et al. in [ARB12] propose a novel specification language and architecture for dynamically managing distributed software and cloud infrastructure. The application is now expressed as a set of services each of which is adjusted (dynamically started and stopped) according to the achieved SLAs and the user-defined constraints. Katsuno et al. in [KT15] study the problem of deployment parallelization in multi-module applications. Specifically, the authors attempt to detect the dependencies between different modules through the extension of the Chef [che] configuration tool and execute the deployment scripts in a parallel fashion. Finally, Pegasus [DVJ$^+$15] is another deployment system that specializes to deploying scientific workloads. All the discussed systems, attempt to either achieve complete automation or speedup application deployment, ignoring the frequently unstable nature of the cloud resources, in contrast to our work that aims at overcoming transient cloud failures.

Interestingly, the problem of failure overcoming has been the center of interest for many research works lately. Liu et al. in [LMVdMF11] provide an interesting formulation where an application deployment is viewed as a database transaction and the target is to implement the necessary mechanisms to achieve the ACID properties of the latter. To this end, the authors assume that each deployment script must be accompanied by an *undo* script that reverts the changes of the former. This way, in case of errors, any unwanted side effects are nullified. Note that, although this is an interesting formulation, the hypothesis that each script is accompanied by another script that executes undo actions is rather strong and, in many cases, impossible. Yamato et al in [YMTU14] diagnosed some insufficiencies of the state of the art Heat and CloudFormation deployment tools and proposed a methodology through which Heat Templates can be shared among users, extracted from existing deployments and trigger resource updates. The authors of this work also diagnosed the problem of partial deployments due to transient errors and describe a rollback mechanism in order to delete resource generated due to failed deployments. Although deletion of stale resources is a positive step, there exists much room for an automated failure overcoming mechanism. Hummer et al. in [HROE13] highlight the necessity of achieving *idempotency* in production systems, for cases where a new software version is released. In cases where one needs to return to a previous, stable version, it is crucial for the deployment system to rapidly do the rollback and converge to a healthy system state. To this end, this paper focuses on the theoretical model that guarantees theoretical convergence to a healthy deployment state. Rather than wandering to the – possibly enormous – state space, our approach adopts a lightweight filesystem snapshot mechanism that guarantees fast convergence. Finally, the work in [HHD16] shares a similar solution to the previous one and extends the previous work using a different configuration management language.

**Generic Deployment Systems:** Although the prevalence of the Cloud paradigm accentuated the importance of fully automated software configuration, the problem is long discussed prior to the wide adoption of the Cloud and many solutions that operate on the resources per se (either virtualized or not) have been proposed. These solutions do not consider resource allocation. Nevertheless, the problems of synchronization and dependency resolution are also addressed by them and resemble the problem addressed by our work.

CFEngine [cfe] is one of the first systems that was proposed to deal with automated software configuration. It introduced the idea of *convergent operators*: Each operation should have a fixed-point nature: Instead of explaining what needs to be done, a user explains the *desired state*. CFEngine is, then, responsible to perform the necessary actions to reach it. Chef [che] is a popular software configuration system, adopted by Facebook. The deployment scripts (*recipes*) are written in a Ruby-based domain-specific language. Multiple recipes are organized in *cookbooks* which describe the desired state of a set of resources. Puppet [pup] is another widely used software configuration system used by Twitter and Mozilla. Similarly to CFEngine, Puppet adopts a declarative syntax and users describe the resources and the *state* that should be reached when the deployment is over. It is released in two flavors: Enterprise Puppet, that supports coordination between different hosts and Open Source Puppet that comes with a limited number of features.

Unlike the rest of the systems proposed so far, Ansible [ans] follows an *agentless* architecture. Instead of running daemon services inside the machines, Ansible utilizes plain SSH connections in order to run the deployment scripts, configure software, run services, etc. Although its architecture supports coordination between different machines, Ansible users need to carefully design the deployment scripts (i.e., *playbooks*) in order to achieve idempotency. Finally, Salt [sal] (or SaltStack Platform) is a software configuration and remote execution platform written in Python. Similar to Ansible, Salt also uses an agentless architecture; recent Salt versions also support daemon processes inside the target machines in order to accelerate deployment. It is highly modular as a user can customize the behavior of a deployment workflow through extending the default functionality using Python.

Although these tools are extensively utilized in modern data centers and cloud infrastructures, none of them offers a built-in support of coordination between software modules that span to multiple hosts, as our work. This is supported either on premium versions of them or through adopting community-based plugins. Moreover, although these tools attempt to repeat the execution of configuration scripts in order to reach convergence, none of them guarantees idempotence to the extent our work does, i.e., full idempotence to file system related resources. As all of these tools do support the execution of possibly non-idempotent calls (e.g., through the `execute-` calls in Chef) that cannot be undone or leave the burden of writing idempotent scripts to the users (Ansible), our work decouples idempotency from the ability of undoing actions and achieves a greater level of decoupling scripts from their side effects. Finally, since our work is based on executing simple bash scripts in the correct order to different machines, it is much easier to exploit existing deployment scripts written in bash in order to compile new application descriptions and eschew the learning curve of a new language that reflects the target deployment states.

## A.3 Application Deployment

In this section, a thorough description of the proposed methodology is provided. We, first, provide the architecture of the system we implemented. Next, we discuss an efficient deployment model, through which one can describe the deployment of any arbitrary application. We, then, examine how this model is utilized for the identification and recovery from transient cloud failures and, finally, we examine the mechanism through which our prototype guarantees the idempotency of each deployment part.

### A.3.1 Architecture

We begin our discussion through introducing the architecture of the system that implements the deployment methodology this work proposes. Our prototype is named *AURA*[1] and Figure A.1 depicts its architecture. AURA comprises two components: The *AURA Master* and the *AURA Executor(s)*. AURA Master is deployed to a dedicated host (VM or physical host) and consists of a set of submodules that coordinate the deployment process and orchestrate error recovery. Specifically, the *Web UI* and *REST API* submodules export AURA's functionality to its users. Through them, an authenticated AURA user can submit new application descriptions, launch new deployments, query a deployment's state, obtain monitoring metrics and terminate a deployment. The *Provisioner* is responsible to contact the underlying Openstack cluster in order to allocate the necessary resources. The *Scheduler* is the core component that coordinates the deployment process and orchestrates failure overcoming. The application descriptions, the deployment instances, the user configuration options and any other relative information is persisted to a database inside the AURA Master host and it is accessible by the Scheduler.

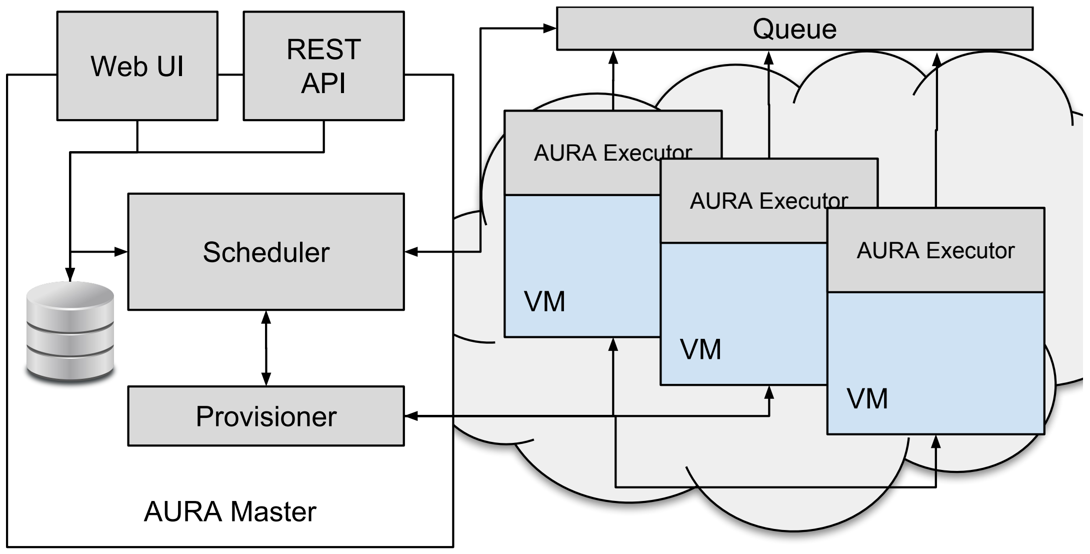


Figure A.1: AURA Architecture

AURA Executors are lightweight processes, running inside the VMs which are allocated to host the application and they are responsible to execute the configuration scripts of the respective modules they are deployed to. The Executors communicate with the AURA Master through a *Queue* that acts as the communication channel between the different components. In case of a transient error, the Scheduler identifies which deployment parts need to be replayed and transmits messages to the Executors that, in turn, cancel any unwanted side effect a previous

[1]According to Greek mythology, Aura was the goddess of breeze, commonly encountered to cloudy environments.

script execution left. Note that, although the Queue is depicted as an external module in order to increase the figure's readability, it also belongs to AURA Master, in the sense that it is statically deployed in the same AURA Master host and Executors from different deployments utilize the same queue to exchange messages.

Before analyzing AURA's functionality is detail, let us provide the key assumptions that drove AURA's design. First of all, the errors that emerge have a *transient* nature. This means that they are only present for a short period of time and vanish without requiring any manual intervention. Second, we assume that the communication channel between the Executors and the Master is *reliable*, i.e., in case where an Executor sends a message to the Queue, this message always reaches its destination. Third, we assume that the AURA Master is always *available*, i.e., it may not fail. Given the above, we now proceed with describing AURA's deployment model and mechanisms through which it achieves error recovery and deployment script idempotence.

### A.3.2 Deployment Model

Assume a typical three-tier application consisting of the following modules: A Web Server (rendering and serving Web Pages), an Application Server (implementing the business logic) and a Database Server (that persists the application's data). For simplicity's sake, we assume that each module runs in a dedicated server and the application will be deployed in a cloud provider. If the application deployment occurred manually, one should create three VMs and connect (e.g., via SSH) to them in order to execute scripts that take care of the configuration of the software modules. In many cases, the scripts need input that is not available prior to the resource allocation. For example, the Application Server needs to know the IP address and the credentials of the Database Server in order to be able to establish a connection to it and function properly. In such cases, the administrator should manually provide this *dynamic* information.

The automation of the deployment and configuration process requires the transmission of such dynamic information in a fully automated manner. For example, upon the completion of the configuration of the Database Server, a message can be sent to the Application Server containing the IP address and the credentials of the former, through a *communication channel*. Such a channel can be implemented via a simple queueing mechanism. Each module publishes information needed by the rest of the modules and subscribes to queues, consuming messages produced by other modules. The deployment scripts are executed up to a point where they expect input from another module. At these points, they block until the expected input is received, i.e., a message is sent from another module and it is successfully transmitted to the blocked module. The message transmission is equivalent to posting a new message into the queue (from the sender's perspective) and consuming this message from the queue (from the receiver's perspective). In cases that no input is received (after the expiration of a fixed time threshold), the error recovery mechanism is triggered.

Albeit the content of the messages exchanged between two modules depends on the executed deployment scripts and the type of the modules (e.g., passwords, SSH keys, static information regarding the environment a module is deployed to, etc.), each message belongs to one of the following three categories: *Static*, *Dynamic* and *ACK* messages. *Static* messages are considered the ones that contain information that does not change when a deployment script is re-executed (e.g., VM's IP address, amount of memory, number of cores, etc.). *Dynamic* messages contain information that change any time a deployment script is re-executed. For example, if a script running on a Database Server generates a random password and transmits it to the Web Server, in case of a script re-execution a new password will be generated and a new message will be sent (containing a different password). Finally, *ACK* messages assist synchronization of the deployment between different software modules and are used to enforce a script execution in a particular order as they do not contain any useful information.

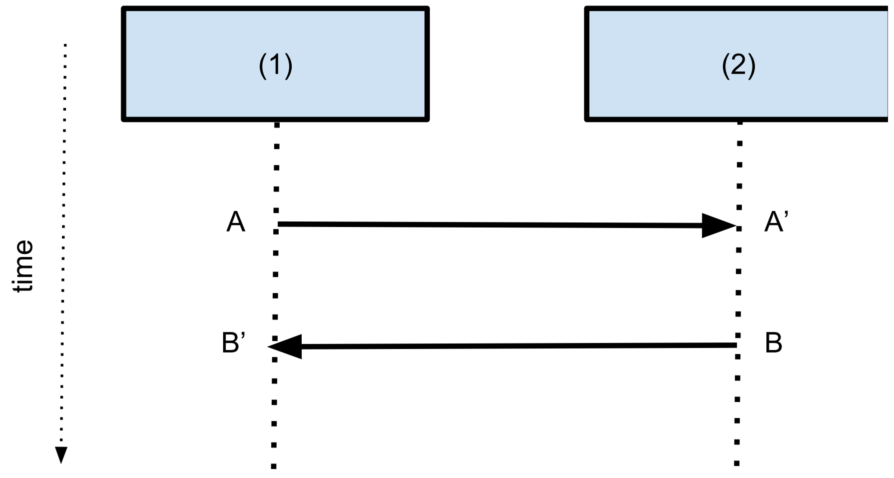


Figure A.2: Message exchange between different software modules and circular dependencies

To provide a better illustration of the deployment process, consider Figure A.2. In this Figure, a deployment process between two software modules named (1) and (2) is depicted. The vertical direction represents the elapsed time and the horizontal arrows represent message exchange[2]. At first, both (1) and (2) begin the deployment process until points $A$ and $A'$ are reached respectively. When (1) reaches $A$, it sends a message to (2) and proceeds. On the other side, (2) blocks

[2]Note that message transmission might not be instant (as implied by the Figure) since consumption of a specific message might occur much later than the message post, but the arrows are depicted perpendicular to the time axis for simplicity.

at point $A'$ until the message sent from $(1)$ arrives and, upon arrival, consumes it and continues with the rest of the script. If the message does not arrive, then the recovery procedure is triggered, as described in the next section.

The functionality of the above message exchange mechanism resembles the functionality of a UNIX pipe: Message receiving is blocking for the receiver end, whereas the sender module posts something to the channel and proceeds instantly. In some cases, though, blocking message transmissions may be desired. For example, take the case of two modules negotiating about a value, e.g., a randomly generated password assigned to the root account of the Database Server. Assume that module $(1)$ (the Application Server) decides on the value and sends it to module $(2)$ (the Database Server). Module $(1)$ must ensure that the password is set, prior to trying to connect to the database. To this end, $(2)$ can send an acknowledgment as depicted between points $B$ and $B'$. In this context, the message exchange protocol can also function as a synchronization mechanism. This scheme represents a dependency graph between the application's modules, since each incoming horizontal edge (e.g., the one entering at point $A$') declares that the execution of a configuration script depends on the correct execution of another.

Note that, schemes like the one depicted in Figure A.2 present a circular dependency, since both modules $(1)$ and $(2)$ depend on each other, but on different time steps. This feature enhances the expressiveness of the proposed deployment model, as one can easily describe extremely complex operations between different modules, which depend on each other in a circular manner. Various state-of-the-art deployment mechanisms do not handle this circularity (e.g., Heat [heaa], CloudFormation [cloa]) since they lack the concept of time during the configuration process and, hence, forbid their users to declare dependencies that create loops. In our view, this circularity naturally occurs to a wealth of modern cloud applications, that comprise many modules deployed to different VMs and, hence, the prohibition of such loops leads to application descriptions that rely on "hacks" to work. On the other side, the allowance of circular dependencies leaves room for application descriptions that contain deadlocks, i.e., pathological cases where each module waits for another and the deployment blocks forever. This problem is outside the scope of our work; it is the user's responsibility to generate correct, deadlock-free application descriptions that, if no failures occur, reach to termination.

### A.3.3 Error Recovery

We now describe the mechanism through which the deployment errors are identified. During the deployment process, a module instance may send a message to another module. This message may contain information needed by the latter module in order to proceed with its deployment, or it could just be a synchronization point. In any case, the second module blocks the deployment execution until the message arrives, whereas the first module sends its message and proceeds with the execution of its deployment script. In the case of an error, a module will not be able to send a message and the receiver will remain blocked. To this end, we set a timeout period that, if exceeded, the waiting module sends a message to the AURA Master informing it that a message has been missed and a possible error might have occurred. From that point on, the error recovery mechanism takes control, evaluates whether an error has, indeed, occurred or the running scripts need more time to finish, i.e., the script is still running. In the former case, the error is evaluated, as we will shortly describe, and the necessary actions are performed in order for the deployment to be restored. The selection of an appropriate timeout period is not trivial. While smaller thresholds may trigger an unnecessary large number of health checks in cases where the deployment scripts need much time to complete, larger ones make our approach reacting slower to errors. The default timeout period used by AURA equals to 1 minute. Nevertheless, if the users are knowledgeable about the real deployment time of their application, they can further optimize it. In the experimental evaluation, we provide a rule of thumb that facilitates with this choice. Finally, when a module's configuration successfully terminates, it reports its terminal state to AURA Master; the entire deployment is considered successful when all software modules have successfully been deployed.

Upon the identification of a (possible) failure, the error recovery mechanism is triggered. The error recovery process aims at overcoming the transient cloud failure encountered during the deployment and is based on the repetition of the execution of scripts/actions which led the system to failure. In order to identify the parts of the deployment that need to be re-executed, AURA needs to be aware of the dependencies that exist between different deployment scripts. These dependencies can be easily resolved, when considering that the executions of the scripts and the message exchange between different modules (as presented in Figure A.2) constitute a *graph* (referred as the *deployment graph* henceforth) that express the order with which different actions must take place. For example, consider the graph presented in Figure A.5, that presents the deployment graph of a Wordpress application comprising a Web Server and a Database Server. The graph nodes represent the states of the deployment, the solid edges represent a script execution and the dotted edges represent a message exchange. The examination of the deployment graph indicates that the execution of, e.g., the `web-server/3` script depends on the execution of two other scripts: `web-server/2` and `db-server/3` (that sends an *ACK* message). In the general case, if one wants to identify the dependencies of any script, one needs to traverse the graph in reverse edge direction.

Having defined the methodology used to identify the dependencies between different deployment scripts, we can now formally describe the Error Recovery procedure, as presented in Algorithm 5. The algorithm accepts two parameters: A deployment graph $T$ (as the one depicted in Figure A.5) and the id of a failed node $n$. The algorithm traverses $T$ in a Breadth-First order beginning from the failed node and traversing the graph in opposite edge direction. The goal

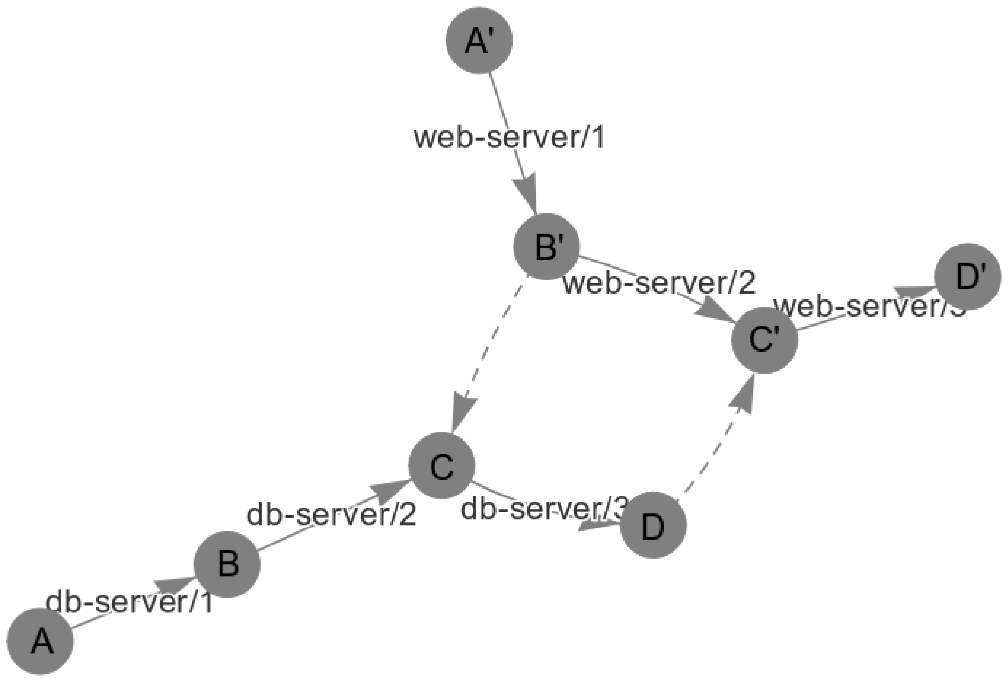


Figure A.3: Wordpress deployment graph with 1 Web Server and 1 Database Server

of the algorithm is to identify a list of nodes that have been successfully reached by the deployment. Starting from the node that triggered the Error Recovery, the algorithm retrieves all the node's dependencies (`depends` function). For example, if the failed node is $C'$, the `depends` function returns the nodes $B'$ and $D$. The `failed` function checks the respective Executor's state and returns *true* if the Executor reports that the intermediate state has not been reached. For example, if $D$ has not been reached then `failed`($D$) returns *true* and $D$ is placed to the `failed` list in order to be later examined. If $B'$ is successfully reached, then $B'$ is placed in the `healthy` list and its dependencies are not examined. This procedure is iterated until all failed nodes are identified. Using this *healthy* list, the AURA Master contacts the respective Executor for each node and instructs them to resume their execution from their last successfully reached healthy state. In the previous example, if we assume that $C$ was successful, then the *db-server* Executor is instructed to repeat `db-server/3`.

It should not be overlooked, that when certain parts of the deployment graph are re-executed, the replayed scripts may produce new or require to consume older messages previously transmitted by other modules. In case where a replayed script needs to consume older messages previously sent by other modules, AURA keeps track of all the messages and forwards the ones required by a replayed script. What happens, though, when a replayed script produces new messages? In these cases, AURA's behavior is affected by the type of the message: If the new message is of type *Static* or *ACK*, the message is sent by the replayed script and ignored by AURA's Queue

**Algorithm 5** Error Recovery Algorithm

```
Require: deployment graph $T$, failed node $n$
Ensure: list of healthy nodes $healthy$
1: $failed \leftarrow \{n\}$
2: $healthy \leftarrow \emptyset$
3: while $failed \neq \emptyset$ do
4:   $v = \text{pop}(failed)$
5:   for $t \in \text{depends}(T, v)$ do
6:     if failed($t$) then
7:       $failed \leftarrow failed \cup \{t\}$
8:     else
9:       $healthy \leftarrow healthy \cup \{t\}$
   return $healthy$
```

module. Since both message types are not affected by a script re-execution, there is no need to propagate the new messages to other, healthy deployment parts. However, when a *Dynamic* message is sent, the recipient module need to repeat the configuration since the receiving information changed. For example, if a Database Server script re-generated a new password and sent it to a Web Server, the latter needs to be reconfigured in order to reflect the updates. It is, now, evident, why AURA implements different message types and how these affect error-recovery. Through this mechanism, AURA's user has the liberty to define the message types of the deployment scripts and affect the algorithm's behavior.

Finally, before proceeding to the examination of the idempotency enforcement mechanism, we should not overlook that since different deployment parts run in parallel, *race conditions* between different modules may be encountered. For example, take the case where one module broadcasts a health-check request, and while the master runs Algorithm 5, another module broadcasts a new request. Since parallel algorithm executions may lead to contradicting actions (i.e., different *healthy* sets), we need to ensure that the algorithm executions are properly synchronized. To this end, each health-check request is placed to a FIFO queue and examined one at a time: The first health-check request is examined by the master, which runs Algorithm 5 and informs the failed modules about the necessary actions. Subsequently, the next health-check request is examined, Algorithm 5 runs again, but now the previously failed modules are in an "Executing" state. If the new health-check request was issued for another module (i.e., not one of the modules that failed in the previous Algorithm execution), then this module is resumed. In any other case, the failing module is already in an "Executing" state and, hence, the health-check request is ignored. In essence, the serialization of both the health-check requests and the algorithm's executions, ensure that the actions taking place for resuming the failed deployment parts leave the deployment in consistent states and eschew race conditions.

### A.3.4 Idempotency

The idea of script re-execution when a deployment failure occurs, is only effective when two important preconditions are met: (a) the failures are transient and (b) if a script is executed multiple times it always leads the deployment into the same state, i.e., it is *idempotent*. In this, section we discuss these preconditions and describe how are these enforced through our approach.

First, a failure is called "transient" when caused by a cloud service and was encountered for a short period of time. Network glitches, routers unavailability due to a host reboot and network packet loss are typical examples of such failures caused by the cloud platform but, in most cases, they are only observed for a short period of time (seconds or a few minutes) and when disappeared the infrastructure is completely functional. Various works study those types of failures (e.g., [PJ13]) and attribute them to the complexity of the cloud software and hardware stacks which presents strong correlations between seemingly independent cloud components [ZCWF14]. Although most cloud providers protect their customers from such failures through their SLAs [PRS09], they openly discuss them and provide instruction for failures caused by sporadic host maintenance tasks [awsd] and various other reasons [vcl]. Since the cloud environment is so dynamic and the automation demanded by the cloud orchestration tools requires the coordination of different parties, script re-execution is suggested on the basis that such transient failures will eventually disappear and the script will become effective.

However, in the general case, if a script is executed multiple times it will not always have the same effect. For example, take the case of a simple script that reads a specific file from the filesystem and deletes it. The script can only be executed exactly once: The second time it will be executed it will fail, since the file is no longer available. This failure is caused by the side effects (the file deletion) of the script execution, which lead the deployment to a state in which the same execution cannot be repeated.

In order to overcome this challenge, we employ a lightweight filesystem snapshot mechanism, that aims at persisting the filesystem state *prior* to each script execution and, in case of failure, revert to it, in order to cancel any changes committed by the failed script. Filesystem snapshotting is a widely researched topic in the Operating Systems field [RBM13, HP94], and it is commonly used for backups, versioning, etc. The mechanics behind the snapshot implementation differs among various filesystems: A straightforward implementation requires exhaustively copying the entire VM filesystem to a secure place (and copying it back during the *revert* action), whereas more sufficient approaches rely on layering or copy-on-write techniques. The huge overhead of copying the entire filesystem lead us to favor the latter techniques. To this end, we implemented two different snapshot mechanisms, using two different widely popular filesystems: AUFS [auf], that relies on layering and BTRFS [RBM13] that relies on copy-on-write.

*AUFS* is one of the most popular layered filesystems that rely on union mounting. The main idea behind union mount is that a set of directories are mounted with a specific order and the final filesystem is the union of the files contained to each distinct directory. Whenever one wants to *read* a file, the uppermost layers are scanned first and, if the file is not found, the search is continued to the bottom layers. Each layer may be mounted with different permissions; usually, the topmost layer is mounted with read-write access and the rest of the layers are mounted with read-only access. When a *write* operation takes place, the updates are only written to the topmost, read-write layer, keeping the rest of the layers intact. The same happens when deleting a file: Instead of physically removing the file from its layer, a new *special file* is generated, demonstrating that the target file is not available. This mechanism inherently supports filesystem snapshots: Prior to each script execution, a new *layer* is mounted on top of the others. If the script fails, then the newly allocated layer is removed and, hence, all the scripts' side-effects are canceled. In different case, new layers are appended on top of the existing ones.

*BTRFS*, on the other hand, is a much newer and promising filesystem that relies on copy-on-write for snapshotting. It supports the concept of *volume*, which acts as an endpoint used by the OS to mount it to the root filesystem. Snapshotting occurs on a per-volume basis: One can create a snapshot, which represents a state of a volume. When something changes (e.g., new files are created/updated/deleted), BTRFS only creates the inodes that are updated, minimizing the number of filesystem structures which are updated. The main difference between BTRFS and AUFS is that the latter works on a file level, i.e., when a file is updated, the entire file is replicated to the topmost layer, whereas the former works on a block level, i.e., only the filesystem blocks which are affected are replicated. On the other side, AUFS is more generic, since it can be implemented on top of other filesystems, whereas BTRFS is more restrictive since the application data must be persisted to a BTRFS partition. In the experimental evaluation, we continue the discussion for their differences and evaluate their impact to the efficiency of our approach.

Finally, we should note that the discussion around idempotency, is only limited to filesystem related resources because the configuration process of most applications usually relates to modifying configuration files. We shall not overlook, though, that modern applications may demand idempotency to other resources, as well: The memory state of a module, for example, cannot be reset when a script need to be re-executed. This is an interesting extension of our approach which will be addressed in the future.

## A.4 Implementation Aspects

Upon presenting AURA's architecture and describing its main functionality, we now wish to extend the discussion to specific implementation aspects of the proposed system. Our analysis comprises two dimensions. We first introduce two optimizations that boost AURA's applicability and, subsequently, discuss the technical means through which the proposed system approaches the concepts of *Portability* and *Reusability*.

### A.4.1 Optimizations

We now provide some optimizations that were implemented to enhance AURA's applicability. So far, we have made the assumption that there exists a one-to-one mapping between application modules and VMs, i.e., each module (e.g., (1) of Figure A.2) is deployed to a dedicated VM. Bearing in mind, though, that modern distributed applications, inherently deployed to cloud environments, rely on deploying the same software modules to more than one VMs (e.g., HDFS clusters [B+08] comprise many datanodes), this assumption is rather restrictive. To this end, AURA's implementation supports augmenting application description with an extra *deployment parameter* that refers to each module's `multiplicity`. This means that prior to deploying, a user can define a (strictly positive) module multiplicity and this, practically, leads to replicating the same software module description as many times as required in different VMs. In this case, the Queue module modifies message exchange accordingly: During message transmission, messages from VMs that host the same software modules are merged in a predefined order (ascending Executor UUID) and offered to the recipient(s), whereas during message reception, the Queue replicates the message as many times as needed. Figure A.4 graphically represents this process, being a variation of Figure A.2, where (2) is deployed to two VMs. Note that, the fact that the Queue replicates and merges messages, abstracts the concept of multiplicity from application descriptions and maximize their reusability.

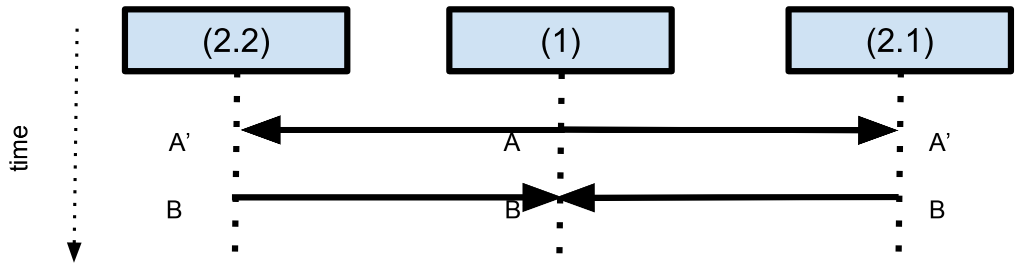


Figure A.4: Message exchange between modules of different multiplicity

Furthermore, apart from the initial deployment phase, the suggested workflow can be extremely helpful during an application's lifetime, for different occasions. For example, take a two-tier application comprising a Web and a Database Server and assume that an administrator wants to run a maintenance task, e.g., update the Database Server software. This task can easily be described in the form of a DAG, where one would, first, gracefully shutdown the Web Server (closing any stale DB connections), then shutdown the DBMS, update it and, finally, start the services in reverse order. Note that, this chain of events can be much more complex and may span to more than two modules (e.g., assume that there exist more than one Web Servers). In order to support these actions, AURA supports the definition of a second *deployment parameter* which defines the `address` of a VM: If set, the Provisioner submodule does not allocate a new VM. In this case, the (already deployed) AURA Executor, receives a new set of scripts that need to be executed according to the proposed methodology. It must be emphasized that this is a first step towards dynamic application *scaling*, i.e., resource allocation and software configuration in order for an application to utilize the new resources, and it is a very interesting future direction that will be explored.

### A.4.2 Portability and Reusability

Two dimensions that highly affected AURA's design choices and have not been examined this far, are *Portability* and *Reusability*. Portability is related to AURA's ability to be able to deploy new application instances in different environments. Reusability is the ability to utilize existing deployment scripts in order to compose new application descriptions compliant with AURA. Although these aspects are not considered *Functional Requirements* in the narrow sense, they are both highly desirable and, if achieved, they can drastically increase AURA's utility.

AURA's modular architecture, as depicted in Figure A.1, allows different modules (e.g., Provisioner, Scheduler, etc.) to communicate in a transparent way, without relying on the design of each other. For example, when the Scheduler module issues a new request to the Provisioner module to allocate new VMs, the latter communicates with Openstack without exposing Openstack-specific semantics to the former. This means that if one wants to utilize AURA in a different cloud environment that adheres to a different API (e.g., AWS, CloudStack [KJM+14]) or utilizes a different virtualization technology (e.g., Kubernetes [Ber14] utilizing Linux Containers), one could provide a custom Provisioner module that contacts the respective cloud platform without altering anything else to AURA's functionality. The only requirement for a custom Provisioner module is to respect the `CloudOrchestrator` API, as seen from AURA's source code [aur], and implement the `create_vm` and `delete_vms` functions. In a similar manner, the user may also override AURA's default behavior regarding VM creation that in case where a VM allocation request fails, the whole deployment is aborted. One could easily change this behavior through adding a loop that eagerly spawns new VM allocation requests in case where the underlying Openstack calls fail.

Finally, we should note that the design choice of supporting DAGs for AURA's application descriptions was favored against other options (e.g., describing the state of the different VMs) for the following reason: The users can easily utilize existing deployment scripts, written in `bash` or any other language, in order to compose more complex application descriptions. One can easily transform a set of existing deployment scripts to a DAG-based description, as the one depicted in Figure A.3, following two steps:

1. "Break" a larger deployment script in smaller parts, identifying which ones should be executed on which VMs and what input is anticipated for each module,
2. Generate a `description.json` file that encodes the order of execution inside a module and the input/output messages each script anticipates.

The syntax of the description file is simple, and contains the bare minimum information required to generate the deployment graph. Listing A.1 provides an example description file for the Wordpress application. Initially, the users define the application modules of their application (e.g., `db-server` and `web-server`). For each module, the user provides a list of scripts that contain its sequence number (`seq` parameter) that indicates its order of execution, the path of the deployment script (`file` parameter) that may be any executable file, and (if applicable) a list of script names in which the current script depends to or the scripts to which the current script will send messages to (i.e., `input` and `output` parameters respectively).

Listing A.1: Wordpress description file

```
{
  "name": "Wordpress",
  "description": "Simple Wordpress installation",
  "modules": [{
    "name": "db-server",
    "scripts": [{
      "seq": 1,
      "file": "db_server/install.sh"
    },{
      "seq": 2,
      "file": "db_server/configure.sh"
    },{
      "seq": 3,
      "file": "db_server/create_user.sh",
      "input" : ["web-server/1"],
      "output": ["web-server/3"]
    }]
  },{
    "name": "web-server",
    "scripts": [{
      "seq": 1,
      "file": "web_server/send_ip.sh",
      "output":["db-server/3"]
    },{
      "seq": 2,
      "file": "web_server/install.sh"
    },{
      "seq": 3,
      "file": "web_server/configure.sh",
      "input": ["db-server/3"]
    }]
  }]
}
```

Note that the actual deployment scripts (such as `web_server/send_ip.sh`) need not be modified in order to publish messages to the AURA Queue. Instead, the AURA Executors that run the deployment scripts, collect their output (everything a script has produced in its standard output) and send it through the AURA Queue to their recipients. On the other end, the Executor that runs a script awaiting for a message, serializes this output and stores it to a temporary file. The path of this temporary file is, then, given as the first argument to the deployment script that opens it, parses it and utilizes its content as it wishes. For example, the script `web_server/send_ip.sh` echoes the VM's IP address. The Web Server Executor collects it and sends it to the Database Server Executor through the Queue module. When the script `db_server/create_user.sh` need to be executed, the Database Server Executor collects the IP address and places it to a file; the script is then launched with this path as the first argument. Subsequently, the script reads the IP address and utilizes it to grant access to Wordpress' database so that the Web Server can access it.

It should be stressed that the choice of not exporting the Queue's semantics to the deployment scripts, liberate the users from writing "AURA-compliant" application descriptions and allow them to write simple scripts that produce and consume information in a traditional way, i.e., through files. In our view, this option greatly simplifies the application description generation process and makes AURA more user-friendly and maximize the reusability of existing deployment scripts.

## A.5 Experimental Evaluation

We now provide a thorough experimental evaluation of the proposed approach that attempts to quantify AURA's efficiency and suitability for deploying real-world application in unstable environments.

**Experimental Setup:** All experiments were conducted on a private Openstack cluster installation, that comprises 8 nodes, each of which has 2 $\times$ Intel Xeon E5-2630 v4 (2.20GHz) and 256G RAM (totaling 320 HW threads and 2TB of RAM). The nodes are connected with 10G network interfaces and the VM block devices are stored over a CEPH cluster that consists of 8 OSDs (4 $\times$ 3TB 3.5" HDDs on RAID-5 setup each) and 98TB storage. The cluster runs the latest stable Openstack version (Pike) and the hosts run Ubuntu 16.04 with Linux kernel 4.4.0-97.

**Applications:** In order to evaluate the efficiency of our approach, we opted for popular, real-world applications, commonly encountered to cloud environments:

- *Wordpress* is a popular Content Management System, used to run and manage Web Applications for different purposes. It requires two components: A Web Server, which renders the user interface and a Database Server, which persists the application's data. Therefore, the application description comprise these modules: One module that installs the Apache Web Server along with any other dependency and one module that hosts MariaDB, i.e., the application's database backend. Each module is installed to a dedicated VM and the Web Server may be replicated to more than one VMs, i.e., the Web Server's multiplicity may be greater than one.

- *Hadoop* [Whi12] is a popular data processing system, commonly deployed to cloud infrastructures in order to persist and process Big Data. It also comprises two modules: A Master node that acts as the coordinator of the cluster and the Slave node(s) that are responsible both to persist the cluster's data and to run tasks from MapReduce jobs. In each deployment, there only exists one Hadoop Master node and a set of Slave nodes, determined by the module's multiplicity. We should emphasize that Hadoop is a typical application deployed to the Cloud; for this reason, the most popular cloud providers offer tools to their users that automate the provisioning of Hadoop clusters and also consider their elastic scaling (e.g., Amazon EMR[3]).

Both application descriptions contain: (a) the appropriate scripts in order to install the necessary software components (e.g., Web/Database servers, Hadoop daemons, etc.) along with any other software (e.g., Java, PHP, etc.) or other requirements (e.g., SSH keys, setting up the hosts file, etc.) and (b) the deployment DAG that describes the order of the script execution. For brevity, we omit a detailed description of each configuration script[4]; Figures A.5 and A.6 depict the structure of the deployment graphs for Wordpress (that consists of 1 Web Server and 1 Database Server) and Hadoop (that consists of 1 Hadoop Master and 2 Hadoop Slaves) respectively.

The graph nodes represent the states of the modules, the solid edges represent script executions and the dotted edges represent message exchanges between different modules/VMs. Each script execution is labeled in the form `<module>/<sequence>`. When the module multiplicity is greater than 1, the module name in the label also contains the VM's serial number as a

[3]https://aws.amazon.com/emr/

[4]More details, though, can be found online at [had] for Hadoop and [wor] for Wordpress.

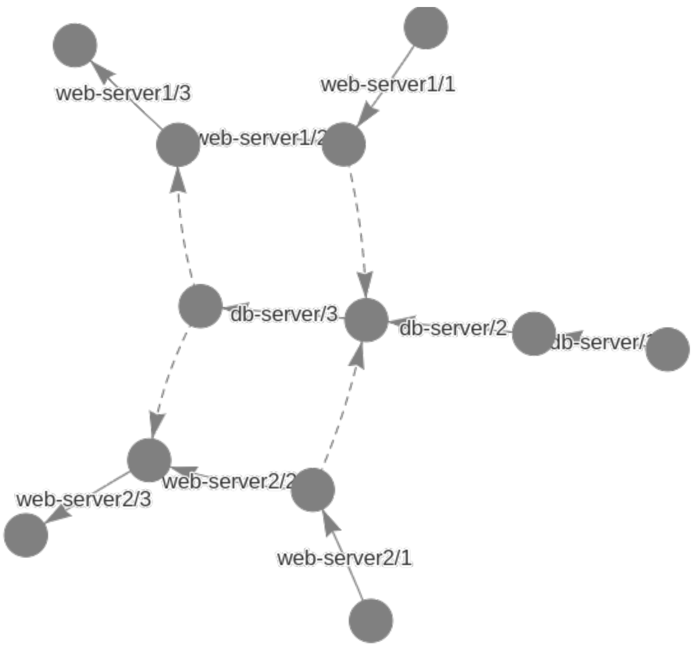


Figure A.5: Wordpress deployment graph with 2 Web Servers and 1 Database Server

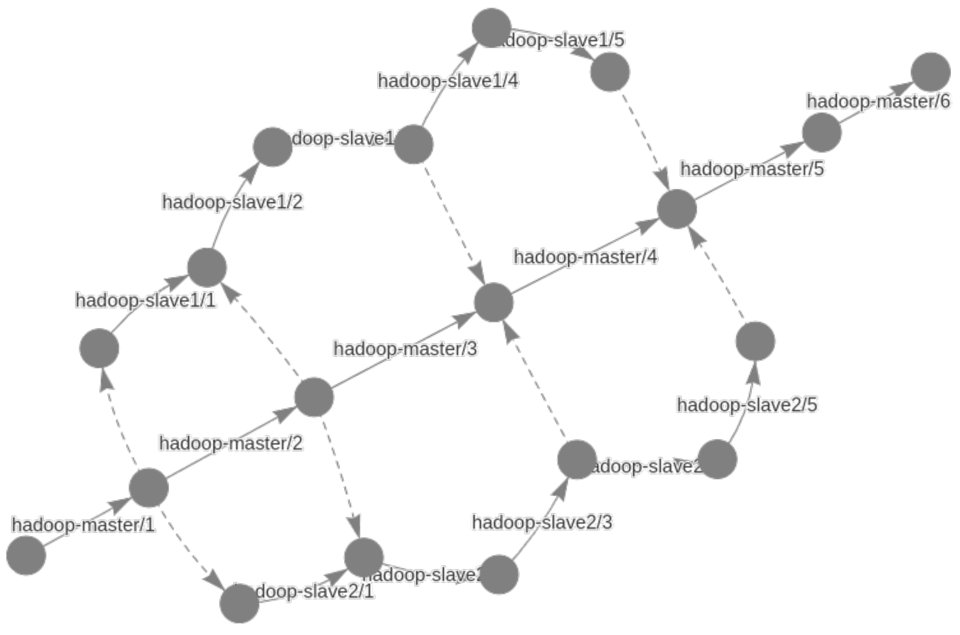


Figure A.6: Hadoop deployment graph with 1 master and 2 slave nodes

suffix, e.g., `hadoop-slave2/3` denotes the third configuration script for the second hadoop slave node.

**Methodology:** In order to quantify AURA's efficiency, we deployed the previously described applications, using different deployment parameters (e.g., different module multiplicity, filesystem snapshot methodology, etc.) and studied the deployments' behavior. In order to eliminate the unavoidable noise attributed to the randomness of our setup, we executed each deployment 10 times and provide the mean of our results. Our evaluation unfolds in four dimensions. First, we test the deployment behavior for varying module multiplicity, measuring the *scalability* of the proposed deployment scheme, i.e., the ability to deploy an application comprising more nodes with minimal execution overhead. Second, we study the deployment behavior when transient errors appear with *varying frequency*, measuring not only the total execution time but also the overhead introduced due to our filesystem snapshot mechanism. Third, we outline the differences between the implemented *snapshot mechanisms*, i.e., *AUFS* and *BTRFS*. Finally, we compare AURA to Openstack Heat, a popular, state-of-the-art deployment system.

## A.5.1 Deployment Model Scalability

We begin our discussion through evaluating our scheme's scalability when an increasing number of modules/VMs is deployed to an error-free environment, i.e., no transient failures occur. We deploy the two considered applications and increase the `multiplicity` parameter for one of their modules, i.e., the *Web Server* and *Hadoop Slave* modules for Wordpress and Hadoop respectively and measure the time needed to complete each deployment phase. We consider three deployment phases: (a) The resource allocation (*Alloc*) phase, in which Openstack allocates the necessary resources, (b) the VM booting phase (*Boot*), in which the guest OS boots and (c) the software configuration phase (*Conf*), where the deployment scripts are executed. Note that, when multiple VMs/modules are considered, each phase is concurrently executed on each one separately. The presented time equals the *real* execution time, i.e., the time between the first VM entering a phase until the last VM leaving this phase. Figure A.7 presents our results (mean times of 10 runs).

Figure A.7 demonstrates that both applications present similar behavior. The resource allocation time presents a marginally increasing trend when more VMs are deployed, whereas the boot time remains constant. In both cases, the configuration phase presents the largest increase, which is, in fact, linear to the number of deployed VMs. The reason behind this rapid increase, though, is not attributed to the deployment model, but to the resource contention introduced when multiple VMs compete for the same resources. Specifically, the configuration scripts for both toy applications assume that they operate on a vanilla VM image and try to configure the entire environment from the beginning, download many software packages from the Web and increasing the deployment's network requirements. This means that an increase in module multiplicity results in a linear increase to the size of files needed to be downloaded and, hence, linear

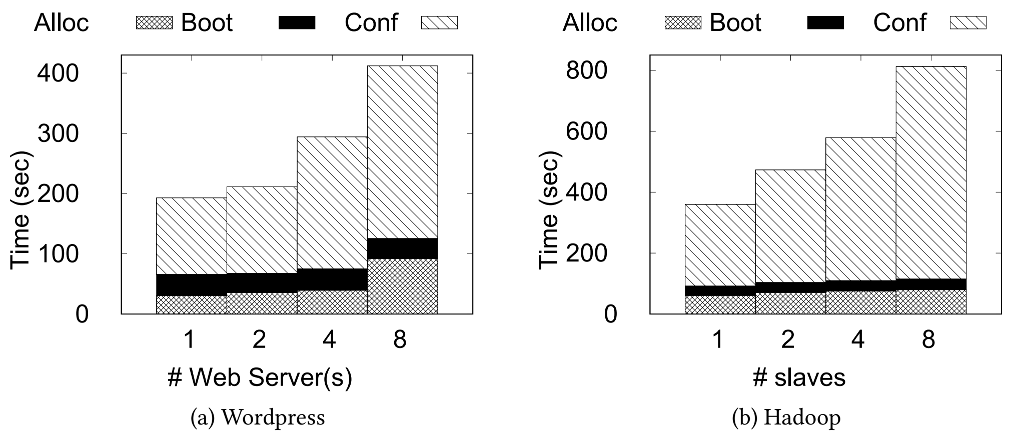


Figure A.7: Execution times of different deployment phases for varying module multiplicity using vanilla images

increase in the configuration time. In order to eschew this misleading behavior and avoid the network bottleneck, we repeat the same experiment, but this time we utilize a *prebaked* image that contains the raw software packages (though unconfigured). Figure A.8 demonstrates our findings.

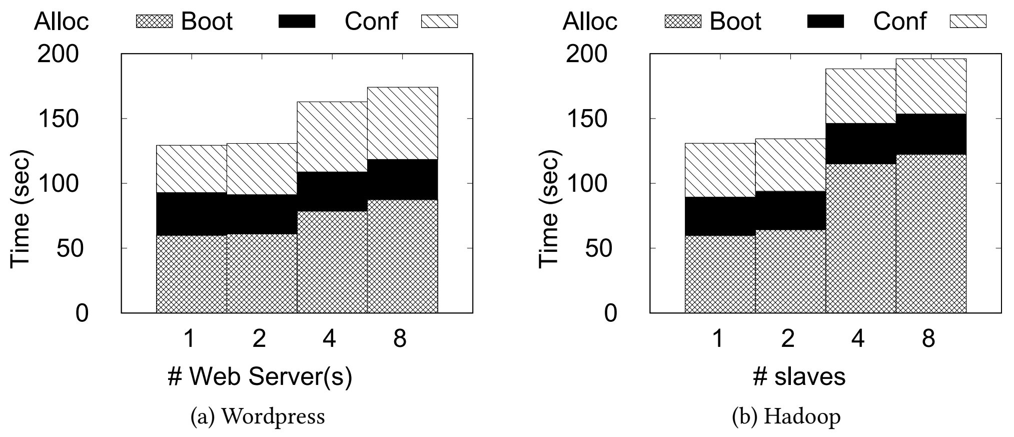


Figure A.8: Execution times of different deployment phases for varying module multiplicity using prebaked images

Figure A.8 depicts that now that the network bottleneck is removed, our deployment scheme achieves to execute the configuration phase in practically constant time, regardless of the number of deployed VMs. This behavior drastically changes the relationships between the times of the deployment phases, making the resource allocation phase dominant of the entire deployment, whereas, again, the booting time is constant and the configuration phase presents marginal increase with the number of deployed VMs. Moreover, note that the absolute times remain extremely low: AURA achieved to deploy a Hadoop cluster of 1 Master and 8 Slaves in less that

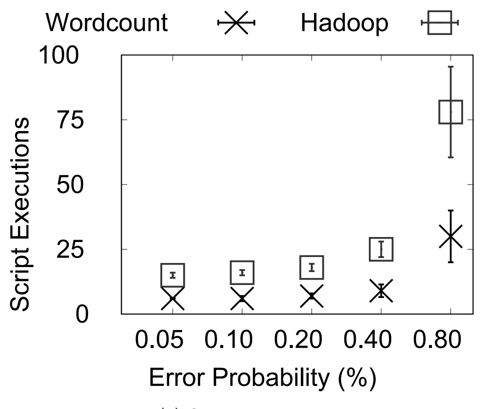
200 seconds, a time that can be further decreased if AURA operates on an enterprise cluster with a faster storage medium that accelerates resource allocation.

### A.5.2 Transient Error Frequency

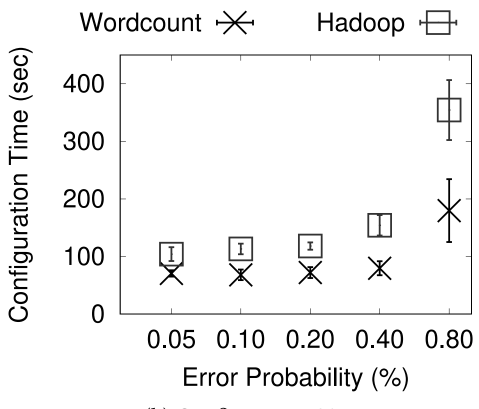
We now evaluate our deployment model's behavior when transient errors appear. In order to produce such transient errors in a controllable and reproducible way, we inserted code snippets inside all application configuration scripts that lead them to failure with a given probability. Specifically, every time a deployment script is executed, a random number is drawn from a uniform distribution between $[0, 1]$ and if it is lower than $p$, where $0 < p < 1$, the script is terminated with a non-zero exit code, leading AURA to interpret this as a failure. The code snippet that generates this behavior is introduced *in the beginning* of each deployment script, hence, each failing script has minimal running time.

One option that highly impacts AURA's behavior for error identification is the timeout period used for detecting errors. As described previously, the default value equals 1 minute. However, when comparing this interval with the actual configuration time for both applications depicted in Figure A.7, one can notice that 60 seconds is a sizeable portion of the entire configuration time. Hence, it can produce slow reactions to errors. To this end, and after experimenting with different values, we concluded that a good approximation of the best timeout interval equals $25\% \times \frac{real\ deployment\ time}{\max\ no.\ scripts/module}$. Intuitively, the fractional part represents the "mean deployment time per script", as if the deployment time was uniformly distributed to all scripts of the module with the most scripts. The percentage indicates the portion of this "mean deployment time" that AURA should wait before checking for errors. In our case, 25% indicates that AURA will approximately do $3--4$ health-checks before the script terminates (in an error-free case). This allows both fast reactions and a minimal number of unnecessary health-checks. With this rule of thumb in mind, we calculate the timeout interval for both applications as follows: $t_{Wordpress} = 25\% \times \frac{127}{3} \approx 10\ sec$ and $t_{Hadoop} = 25\% \times \frac{257}{6} \approx 10\ sec$. The real deployment time for each application is obtained by Figure A.7 for the 1 Slave/Web Server case and the maximum number of scripts/module is obtained by Figures A.5 and A.6: 3 (for `db-server`) and 6 (for `hadoop-master`) respectively.

Given this, we deploy Wordpress (2 Web Servers and 1 Database Server) and Hadoop (1 Master and 2 Slaves) 10 times for varying $p$ values from $0.05$ to $0.8$, measuring the total number of script executions and the real time of the configuration deployment phase. In Figure A.9 we provide the mean values and the respective deviation. We utilized AUFS as the filesystem snapshot mechanism.



(a) Script Executions



(b) Configuration Time

Figure A.9: Number of script executions and configuration times for varying error probability

Both Figures A.9 (a) and (b) present very similar patterns: When the error probability is low (e.g., $0.05 - 0.20$), a minimal number of script executions occur and the configuration time remains very close to the configuration time witnessed to the error-free (i.e., $p = 0$) case. However, when $p$ increases one can observe that both the mean and the standard deviation values rapidly increase. This is attributed to the fact that achieving error-free execution becomes exponentially more difficult, since the execution of a deployment script requires the successful execution of all the scripts it depends on and, hence, much more script executions and time is needed in order to achieve this. Moreover, the two plots also showcase that an increase in $p$ has a greater impact on more complex deployment graphs: Indeed, Hadoop presents a faster increase both in terms of the number of Script Executions and the respective configuration time it requires to be deployed, since it presents more dependencies (Figure A.6) than Wordpress (Figure A.5) and, hence, is more susceptible to script re-executions when errors appear more frequently.

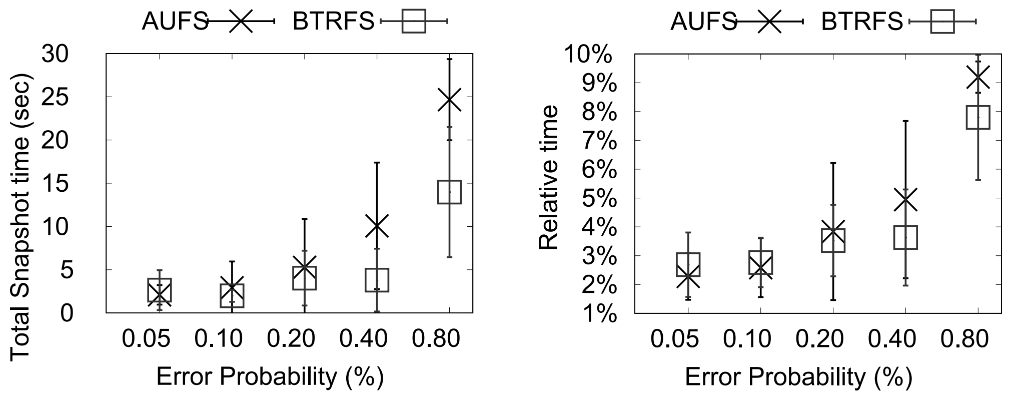
A subtle point of the above discussion, is that this experiment implies that transient failures are independent of each other, in the sense that the emergence of a transient failure in a certain deployment script does not affect another, concurrently executed deployment script. In reality, different transient errors may be strongly correlated: If a network glitch occurred due to a failed switch, chances are that VMs belonging to the same host or rack will equally be affected, hence, their error probabilities are not independent. Although we do recognize that such correlated failures may prolong deployment times, we opted for a simpler transient failure generator in order to simplify the evaluation and obtain a better understanding of AURA's ability to overcome random failures. Furthermore, we should note that the $p$ values used for this discussion are extremely large and only used in order to investigate what happens even in the most unstable cloud environments. It is interesting, though, that even when $p = 0.2$, AURA achieves to overcome any transient error and lead the deployment to successful termination with marginal delays, i.e., less than 10% when compared to the error-free case, both for Wordpress and Hadoop.

### A.5.3 Snapshot Implementation Overhead

We now evaluate the performance of the two snapshot mechanisms we implemented in order to guarantee the idempotent script execution. Specifically, we want to evaluate the overhead that AUFS and BTRFS introduce to the configuration time, i.e., how much time is spent to snapshots and rollbacks against the "useful" deployment time, i.e., the time spent executing the configuration scripts. To this end, we deploy Wordpress and Hadoop using the same multiplicities as before, but now repeat each deployment twice: Once using AUFS for snapshots and once using BTRFS. We launch deployments for varying error frequencies, i.e., different error probabilities, and repeat each deployment 10 times. In Figures A.10 and A.11 we provide the mean values of those runs for Wordpress and Hadoop respectively. The left figures depict the total snapshot time (including both snapshots and rollbacks) for all application scripts and the right figures express this time as a percentage of the total execution time. Observe the difference between this time expression and the time expression used so far: The total snapshot time represents the sum of time periods that the snapshot mechanism is triggered for each module without taking into consideration that different modules may run in parallel. For example, if module (1) and (2) required times $t_1$ and $t_2$ for snapshotting, the Total Snapshot time equals $t_1 + t_2$ whereas the *real* snapshot time (if running in parallel) would be $\max(t_1, t_2)$[5]. Finally, the Relative Time equals the Total Snapshot Time divided by the Total Running Time, i.e., the sum of execution time for each script of each module.

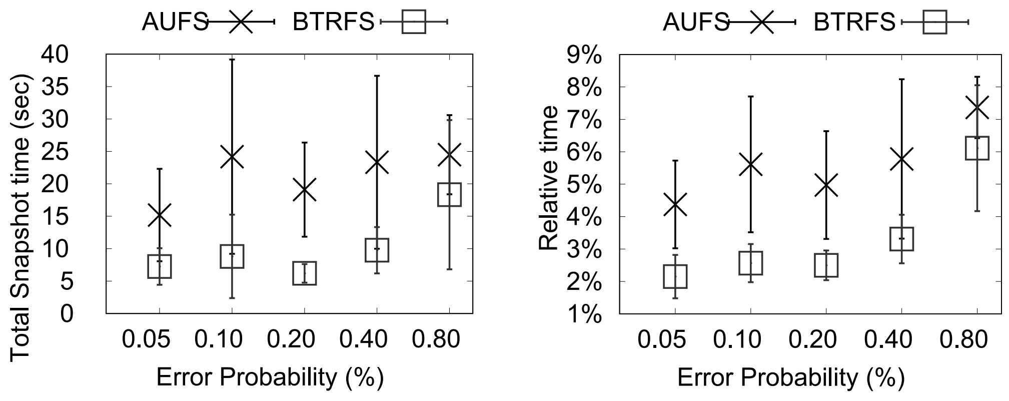
Both Figures demonstrate that BTRFS outperforms AUFS measured both in terms of Total Snapshot Time and in terms of relative time. In fact, the difference between the two mechanisms is increasing for larger $p$ values, i.e., more snapshots and rollbacks are essential. Interestingly, one can also observe that AUFS achieves similar to BTRFS snapshot times for lower $p$ values for Wordpress but, in the Hadoop case, AUFS requires much more time that BTRFS to snapshot and restore the filesystem layers. This interesting finding can be explained when examining the content that is snapshot: In both cases, AURA snapshots the `/opt` directory which is used as the root directory for all application modules. In the Wordpress case, `/opt` contains fewer files (Wordpress files) that aggregate 30MB, whereas in the Hadoop case `/opt` contains much more files that aggregate 500MB (Hadoop bin and configuration files). This is indicative of BTRFS' ability to handle massive storage in a more efficient way: Due to the powerful copy-on-write

[5]In the Operating Systems realm, the running time of a process is distinguished to *real*, *user* and *sys*, where the first denotes the actual clock time, the second equals the total time spent at user space for all threads and the last equals the time spent at kernel for all threads. So far, we have used the *real* time; in this experiment we prefer to demonstrate the *user* time in order to better isolate the snapshot behavior.



(a) Total Snapshot Time (b) Relative Time

Figure A.10: Overhead of AUFS and BTRFS for Wordpress



(a) Total Snapshot Time (b) Relative Time

Figure A.11: Overhead of AUFS and BTRFS for Hadoop

mechanism that operates on the inode level, BTRFS is able to create snapshots and revert to existing snapshots much faster than AUFS that needs to re-assemble the chain of layers from scratch any time a new layer is added/removed. On the contrary, when fewer data are persisted (as in the Wordpress case), both mechanisms can produce equivalent results.

Table A.1: Requirements of AUFS and BTRFS

| **Requirement** | **AUFS** | **BTRFS** |
|---|---|---|
| kernel support | Yes | Yes |
| block device | No | Yes |
| Cooperates with other FS | Yes | No |

Despite presenting superior performance, BTRFS has more runtime requirements than AUFS. To begin with, both mechanisms require kernel support. However, AUFS can operate on top of other filesystems and require no special devices (e.g., a dedicated block device) since it only requires a set of directories in order to work, which are located in the VM's filesystem. On the contrary, BTRFS requires a dedicated block device to be formatted and mounted in order to work. Although this does not limit its applicability, it reduces AURA's transparency towards the application to-be-deployed, as the user needs to take into consideration BTRFS' requirements in order to utilize it. In Table A.1 we summarize AUFS' and BTRFS' requirements. Given the above, we can conclude that when one wants to snapshot a filesystem containing massive amounts of data, BTRFS must be preferred, despite its higher requirements; in cases where one needs to snapshot filesystems with fewer data, AUFS presents a decent behavior and has much fewer requirements.

### A.5.4 End-to-End Performance Comparison

We now wish to evaluate AURA's end-to-end performance in comparison to Openstack Heat [heaa], i.e., a state-of-the-art deployment system that is frequently used in the Openstack ecosystem in order to orchestrate application deployments. Our evaluation's objective is to compare the deployment times of Hadoop and Wordpress for different multiplicities, when these are deployed through AURA and Heat. Since Heat does not support recovery from transient failures, only the error-free cases are considered. Furthermore, since Heat does not support circular dependencies between different software modules, we re-wrote the application descriptions of both applications, constructing two *Heat Orchestration Templates* (*HOTs*) that avoid such loops. Specifically, we conducted the following modifications:

- In the *Hadoop* case, we used predefined SSH keys (hence no negotiation is required between the master and the slaves) and only kept the dependencies from the slaves to the master (sending their IPs and informing the latter that the former ones are ready). Schematically, only the dotted edges beginning from Hadoop slaves towards the Hadoop master are kept, as seen in Figure A.6.

- In the *Wordpress* case, the Database Server does not wait for obtaining the IP address(es) of the Web Server(s), as it is configured to listen to any connection. Schematically, we removed the dotted edges from the output states of `web-server1/1`, `web-server2/2` scripts to the input of `db-server/3` of Figure A.5.

Finally, since Heat does not support an information exchange mechanism as the one supported by AURA [heab], messages are exchanged inside the deployment scripts that securely transfer (through `scp`) any piece of information is anticipated by a consumer script. Given the above, Table A.2 provides the relative deployment times for each application when a varying number of modules (i.e., slaves for Hadoop and Web Servers for Wordpress) is utilized. The relative deployment time is expressed as the ratio between the deployment time in Heat divided by AURA's time, i.e., $T_{relative} = \frac{T_{Heat}}{T_{AURA}}$. Each deployment was repeated 10 times and the mean values are presented.

Table A.2: Relative Deployment Time for Openstack Heat vs AURA

| **Application** | **Deployment Phase** | **Number of modules** | | | |
|---|---|---|---|---|---|
| | | **1** | **2** | **4** | **8** |
| Hadoop | Allocation | 1.0010 | 0.9991 | 1.0001 | 1.0020 |
| | Booting | 1.0001 | 0.9999 | 0.9991 | 0.9997 |
| | Configuration | 1.0000 | 0.9964 | 0.9951 | 0.9889 |
| Wordpress | Allocation | 1.0005 | 1.0030 | 0.9997 | 0.9995 |
| | Booting | 1.0001 | 0.9999 | 1.0003 | 1.0020 |
| | Configuration | 0.9991 | 0.9965 | 0.9903 | 0.9862 |

A closer examination of Table A.2 presents some interesting findings. First of all, both Heat and AURA present similar times during the *Allocation* and *Booting* phases of the deployment and this is not affected by the number of employed modules. This is a reasonable finding, since both tools use the same mechanism to instantiate the VMs, i.e., they both contact the `nova` component in order to issue requests for new VMs. Furthermore, the VM boot time is independent of the employed deployment tool, hence no differences in the VM booting time is observed. This is the reason behind obtaining $T_{relative}$ values close to 1 for both applications during these first deployment phases.

Investigating the respective times during the *Configuration* phase, nevertheless, showcases that Heat requires slightly less time when deploying applications of higher multiplicity (i.e., for 8 modules) than AURA; a difference that does not surpass 2% in the worst case. This marginal difference is ascribed to two factors. First, and as described previously, before the execution of any configuration script, AURA keeps a snapshot of the underlying filesystem in order to achieve idempotency. Although this extra time is negligible, it is an extra step that is avoided by Heat. Second, in this experiment Heat deploys slightly modified deployment graphs that present fewer dependencies between different modules. Consequently, the synchronization points for Heat are fewer and the deployment scripts are allowed to be executed without waiting for other scripts to finish.

It should be stressed, though, that Heat's lack of support for dynamic information exchange between different modules leads to slightly reduced deployment times in comparison to AURA, but also limits its expressiveness for describing complex applications. On the other hand, AURA's ability to execute complex deployment graphs and support recovery from transient failures, add a negligible time overhead that is instantly counterbalanced in the emergence of transient errors.

## A.6 Conclusions and Future Work

In this work, we revisited the problem of application deployment to cloud infrastructures that present transient failures. The complexity of the architecture of modern data centers makes them susceptible to errors that hinder automation and jeopardize the execution of application deployment, a complex task that requires coordination between multiple different components. To address this challenge, we introduced AURA, a cloud deployment system that attempts to fix transient errors through re-executing the part of the deployment that failed. In order to enforce script idempotency, AURA implements a lightweight filesystem snapshot mechanism and, in case of script failure, cancels any unwanted side effects through reverting the filesystem to a previous, healthy state. Our evaluation, conducted for popular, real-world applications, frequently deployed to cloud environments, indicated that AURA manages to deploy applications even in infrastructures presenting high error probabilities and only introduces a minimal overhead to the deployment time that does not surpass 10% of the total deployment time in the worst case.

Let us, now, summarize the takeaways of our work. The suggested deployment model that formulates an application deployment as a DAG of dependencies, provides the necessary building blocks for expressing extremely complex tasks in an efficient way. The introduction of synchronization through message exchange between different modules (or VMs), in particular, increases our model's expressiveness and efficiently addresses the limitation of many real-world deployment systems that assume that two software modules will not depend on each other on the same time, i.e., they only support unidirectional dependencies. Furthermore, in this work we attempted to utilize the well-known technique of filesystem snapshot, in order to give a solution to the problem of idempotent script execution, which is a key requirement for re-executing failed scripts. In our best knowledge, our work is the first that tries to achieve idempotency through it and, according to our evaluation, this technique achieves to nullify any unwanted script side effects to filesystem related resources in an efficient manner and only introduces marginal overhead. As the existing filesystems improve and new ones emerge, this technique can produce even better results.

Finally, the contributions of this work provide a strong foundation for future work. First, the expressiveness of the suggested deployment model renders it as a suitable candidate for expressing complex tasks not only in the initial application deployment phase, but also during the runtime of an application. Elastic scaling, which is a key concept to the cloud industry [GPM+14], requires automated resource provisioning and software configuration, so that resources are utilized appropriately. The adoption of a deployment model as the one suggested in our work would facilitate enforcing complex application resizing actions and increase resource reliability. Second, in this work we studied script idempotency in the light of resources that reside in the filesystem. Even if this hypothesis seems realistic for the deployment phase, as a typical software configuration script handles and modifies files (configuration files, binaries, libraries, etc.), in the general case, idempotency is not achieved to other resources, e.g., the memory state of a process. This is an interesting future direction of investigation that would maximize our work's applicability and allow for considering alternative tasks that exceed the scope of this work. Third, although AURA is capable of addressing concurrent errors in different software modules, it assumes that these errors are independent of each other, as each part of the deployment graph is treated independently. However, as previously discussed, in practice errors can be strongly correlated. This observation provides an interesting foundation for future research as studying the nature of this correlation can contribute to taking smarter decisions during recovery actions and further accelerate deployments.