ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Πρόβλεψη χρόνου επικοινωνίας παράλληλων εφαρμογών με τεχνικές μηχανικής μάθησης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Σωτηρίου Β. Αποστολάκη

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2015

Εθνικο Μετσοβιο Πολυτεχνειο
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Πρόβλεψη χρόνου επικοινωνίας παράλληλων εφαρμογών με τεχνικές μηχανικής μάθησης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Σωτηρίου Β. Αποστολάκη

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή στις 22 Ιουλίου 2015.

.....	.....	.....
Νεκτάριος Κοζύρης	Γεώργιος Γκούμας	Ανδρέας-Γεώργιος Σταφυλοπάτης
Καθηγητής Ε.Μ.Π.	Λέκτορας Ε.Μ.Π	Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2015

.....

Σωτήριος Β. Αποστολάκης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Abstract

Communication time is an integral part of the execution time of large-scale applications and depends on a wide set of factors, including system architecture and software, application's features and obscure network effects. Its prediction is a subject which was studied by many from various perspectives. With our work, we wish to contribute and add a novel approach to this matter. We propose a machine-learning scheme to predict the total communication time of parallel applications. We define a set of features extracted from application's characteristics, allocation size, process mapping and system's architecture, and devise a simple benchmark which sweeps over the parameter space and provides valuable insight on the network effects. Forests of extremely randomized trees - a supervised learning algorithm - is applied to perform regression analysis on communication data, derived from the benchmark, and application execution time. Our prediction model achieves noteworthy results in making predictions for two different communication kernels with various problem sizes and system configurations, and successfully selects optimal execution configurations in a decision-making scenario. Finally, except for making predictions, our work succeeded in identifying the culprits behind performance variability for two dissimilar network architectures.

Keywords: performance prediction, communication time, MPI applications, large-scale systems, machine learning.

Περίληψη

Ο χρόνος επικοινωνίας είναι ένα αναπόσπαστο κομμάτι του συνολικού χρόνου εκτέλεσης εφαρμογών μεγάλης κλίμακας και εξαρτάται από ένα ευρύ φάσμα παραγόντων, συμπεριλαμβανομένων της αρχιτεκτονικής και του λογισμικού του συστήματος, τα χαρακτηριστικά της εφαρμογής και δυσνόητα δικτυακά φαινόμενα. Η πρόβλεψη του κόστους επικοινωνίας είναι ένα ζήτημα που έχει μελετηθεί από πολλούς από διάφορες οπτικές γωνίες. Με τη δουλειά μας αποσκοπούμε να συνεισφέρουμε σε αυτή την προσπάθεια και να παρουσιάσουμε μία πρωτοποριακή προσέγγιση αυτού του θέματος. Προτείνουμε μία machine learning μεθοδολογία για την πρόβλεψη του συνολικού χρόνου επικοινωνίας παράλληλων εφαρμογών. Ορίσαμε ένα σύνολο από παραμέτρους, προερχόμενες από τα χαρακτηριστικά της εφαρμογής, το μέγεθος των χρησιμοποιούμενων πόρων, την αντιστοίχηση των διεργασιών στους πυρήνες του συστήματος και την αρχιτεκτονική του δικτύου. Κατόπιν, συγκεντρώνοντας μετρήσεις, που καλύπτουν όλο το φάσμα των χρησιμοποιούμενων παραμέτρων, με την εκτέλεση μιας απλής γενικευμένης εφαρμογής, αποκτήσαμε μία εικόνα της συμπεριφοράς του υπό εξέταση δικτύου. Ο αλγόριθμος forests of extremely randomized trees χρησιμοποιήθηκε για να προβλέψουμε τον χρόνο επικοινωνίας στηριζόμενοι στα δεδομένα επικοινωνίας που συλλέξαμε με την προηγούμενη διαδικασία. Το μοντέλο πρόβλεψης μας πετυχαίνει αξιοσημείωτα αποτελέσματα στην πρόβλεψη επίδοσης δύο διαφορετικών εφαρμογών για ποικίλα μεγέθη προβλήματος και ρυθμίσεις συστήματος. Επίσης, διαλέγουμε επιτυχώς τις βέλτιστες ρυθμίσεις εκτέλεσης σε ένα σενάριο λήψης αποφάσεων. Τέλος, πέρα από τις προβλέψεις, η δουλειά μας πετυχαίνει να αναγνωρίσει τα αίτια της διαφοροποίησης της επίδοσης σε δύο αρκετά διαφορετικά δίκτυα μεγάλης κλίμακας.

Λέξεις-κλειδιά: πρόβλεψη επίδοσης, χρόνος επικοινωνίας, MPI εφαρμογές, μεγάλης κλίμακας συστήματα, machine learning.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή της εργασίας αυτής, κ.Νεκτάριο Κοζύρη, για τη συνεχή καθοδήγησή του και την καθοριστική συμβολή του στη συνέχιση της ακαδημαϊκής μου πορείας.

Νιώθω την ανάγκη να ευχαριστήσω ιδιαιτέρως τον κ.Γεώργιο Γκούμα για το ενδιαφέρον του και τις συμβουλές που μου παρείχε τα δύο τελευταία χρόνια, καθώς και για την συνδρομή του στην επιλογή της παρούσας και μελλοντικής επιστημονικής μου κατεύθυνσης.

Θα ήθελα να ευχαριστήσω ακόμα τον κ.Ανδρέα-Γεώργιο Σταφυλοπάτη που διατέλεσε μέλος της τριμελούς επιτροπής μου.

Ένα ιδιαίτερο ευχαριστώ οφείλω στην υποψήφια διδάκτορα Νικέλα Παπαδοπούλου για την αμέριστη βοήθειά της και για την άριστη συνεργασία που είχαμε κατά τη διάρκεια της εκπόνησης της παρούσας διπλωματικής εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω ιδιαίτερα την οικογένειά μου για τη διαρκή στήριξη της και τους φίλους μου για όσα περάσαμε μαζί κατά τη διάρκεια της φοιτητικής μας ζωής.

Εκτενής περίληψη

Οι σύγχρονοι υπερυπολογιστές αυξάνουν συνεχώς σε μέγεθος και υπολογιστική ισχύ. Δυστυχώς όμως αυτές οι αυξανόμενες δυνατότητες δεν μπορούν να χρησιμοποιηθούν πλήρως λόγω προβλημάτων συμφόρησης που προκύπτουν στα δίκτυα διασύνδεσης. Το κόστος επικοινωνίας σε μερικές περιπτώσεις αποτελεί μέχρι και το 40% του συνολικού χρόνου εκτέλεσης δημιουργώντας προβλήματα κλιμακωσιμότητας. Συνεπώς, η πρόβλεψη του κόστους επικοινωνίας σε μεγάλης κλίμακας παράλληλες εφαρμογές γίνεται όλο και περισσότερη απαραίτητη τόσο για την αλγοριθμική σχεδίαση όσο και για αποφάσεις που αφορούν τους χρήστες των μηχανημάτων αλλά και τους χρονοδρομολογητές του συστήματος.

Η προσπάθεια που έχει γίνει προς την επίτευξη ακριβούς πρόβλεψης του επικοινωνιακού χρόνου δεν είναι επαρκής και πλήρης και σίγουρα δεν είναι αντίστοιχη με αυτήν που έχει γίνει για το υπολογιστικό κομμάτι. Θελήσαμε λοιπόν να συνεισφέρουμε με τη σειρά μας στη δημιουργία ενός μοντέλου πρόβλεψης για το χρόνο επικοινωνίας παράλληλων εφαρμογών μεγάλης κλίμακας. Χρησιμοποιήσαμε διάφορα γνωρίσματα, τα οποία δίνουν ποικίλες πληροφορίες σχετικά με την εφαρμογή που τρέχουμε, τα χαρακτηριστικά του μηχανήματος και τον τρόπο εκτέλεσης της εφαρμογής στο σύστημα. Εκτελέσαμε ακόμα μια σειρά μικρών προγραμμάτων ανταλλαγής μηνυμάτων στο υπό εξέταση μηχάνημα. Με τα αποτελέσματα αυτών των εκτελέσεων και χρησιμοποιώντας τα γνωρίσματα που ορίσαμε, εκπαιδεύσαμε ένα μοντέλο που καταφέρνει να προβλέψει επιτυχώς το κόστος επικοινωνίας για δύο διαφορετικές εφαρμογές και να εντοπίσει τη βέλτιστη, από άποψη χρόνου εκτέλεσης, επιλογή πόρων σε ένα πρόβλημα απόφασης.

Ο χρόνος εκτέλεσης της επικοινωνιακής φάσης μίας εφαρμογής εξαρτάται από ποικίλες παραμέτρους. Πρώτα από όλα, η επίδοση επηρεάζεται από την αρχιτεκτονική του υπό εξέταση δικτύου και τα χαρακτηριστικά του όπως το εύρος ζώνης και ο χρόνος απόκρισης. Επιπλέον, το λογισμικό του συστήματος επηρεάζει την απόδοση. Οι ρυθμίσεις του χρονοδρομολογητή και το σχήμα της κατανομής πόρων καθορίζουν τα επίπεδα συμφόρησης στο δίκτυο και κατ΄ επέκταση το χρόνο επικοινωνίας. Το πλήθος των βημάτων των πακέτων στο δίκτυο, η ανταλλαγή μηνυμάτων μεταξύ των πυρήνων ενός κόμβου καθώς και η υλοποίηση των βιβλιοθηκών παράλληλου προγραμματισμού διαμορφώνουν και αυτά με τη σειρά τους το τελικό αποτέλεσμα. Τέλος, είναι προφανές ότι και τα χαρακτηριστικά και η φύση της εφαρμογής διαδραματίζουν καταλυτικό ρόλο στη διαμόρφωση του επικοινωνιακού κόστους.

Δυστυχώς υπάρχουν κάποιες δυσκολίες οι οποίες μας εμποδίζουν από το να κάνουμε μία πάρα πολύ ακριβή πρόβλεψη. Από την μία, έχουμε την παρεμπόδιση της εκτέλεσης της εφαρμογής μας από άλλες που τρέχουν παράλληλα με εμάς. Αν παράλληλα στον ίδιο ή σε παραπλήσιο κόμβο με εμάς τρέχει μία εφαρμογή που περιλαμβάνει συνεχή και έντονη ανταλλαγή μηνυμάτων τότε αναπόφευκτα θα επηρεαστεί αρνητικά η επίδοση της δικιάς μας εφαρμογής λόγω της επιπλέον συμφόρησης. Οι επιπρόσθετες καθυστερήσεις λόγω της παρεμβολής είναι πολύ δύσκολο να προβλεφθούν δεδομένου ότι σχετίζονται με εξωγενείς παράγοντες. Πέρα από την παρεμβολή και ο θόρυβος από την παρέμβαση του λειτουργικού συστήματος μπορεί να προσθέσει επιπρόσθετες δυσκολίες.

Διάφορα μοντέλα έχουν προταθεί για την πρόβλεψη του κόστους ανταλλαγής μηνυμάτων. Μία βασική κατηγορία περιλαμβάνει αναλυτικά μοντέλα που δεν πετυχαίνουν αξιοθαύμαστη ακρίβεια και αψηφούν κάποιες λεπτομέρειες που εν τέλει αλλάζουν το κόστος επικοινωνίας. Η χρησιμότητα τους έγκειται στην διευκόλυνση της ανάπτυξης αλγορίθμων και στην κατανόηση κάποιων ιδιοτήτων των δικτύων και όχι στην ακρίβεια πρόβλεψης. Κάποιοι άλλοι χρησιμοποίησαν machine learning τεχνικές και πέτυχαν ιδιαίτερα ακριβείς προβλέψεις. Απαιτούσαν όμως πληροφορίες διαθέσιμες κατά την εκτέλεση της εφαρμογής. Συνεπώς, αυτές οι τεχνικές δεν έχουν πρακτικό ενδιαφέρον ως προς το κομμάτι της πρόβλεψης. Σχετικά με την δικιά μας συνεισφορά, πετύχαμε καλύτερη ακρίβεια από τα αναλυτικά μοντέλα χρησιμοποιώντας δεδομένα που ήταν διαθέσιμα πριν την εκτέλεση της εφαρμογής. Η μεθοδολογία μας εφαρμόστηκε με επιτυχία σε δύο διαφορετικά μηχανήματα και άρα μπορούμε να υποθέσουμε ότι δεν εξαρτάται από κάποια συγκεκριμένη αρχιτεκτονική, όπως ισχύει και για κάποιες από τις προαναφερθείσες εργασίες.

Χρησιμοποιήθηκαν δύο μηχανήματα με αρκετές διαφορές. Το πρώτο είναι το Vilje που είναι ένας υπερυπολογιστής με ένα ενισχυμένο υπερκύβο ως δίκτυο διασύνδεσης. Κάθε μεταγωγέας είναι συνδεδεμένος με 18 κόμβους καθένας από τους οποίους έχει 16 πυρήνες. Η δρομολόγηση γίνεται ντετερμινιστικά και από το συντομότερο μονοπάτι. Το άλλο μηχάνημα είναι το Piz Daint . Ως δίκτυο διασύνδεσης έχει την dragonfly τοπολογία. Η δρομολόγηση των πακέτων επηρεάζεται από τα φαινόμενα συμφόρησης και δεν είναι ντετερμινιστική, ενώ χάρις στο αρκετό εύρος ζώνης του μηχανήματος περιορίζονται πολύ τα φαινόμενα συμφόρησης.

Οι παράγοντες που επηρεάζουν το χρόνο επικοινωνίας είναι πάρα πολλοί και συνεπώς είναι σχεδόν αδύνατο να κατασκευάσουμε ένα αναλυτικό μοντέλο πρόβλεψης μεγάλης ακρίβειας. Καταλήξαμε λοιπόν στο να εξάγουμε ένα μοντέλο πρόβλεψης από δεδομένα εκτέλεσης. Ως βασικό μοντέλο πρόβλεψης χρησιμοποιήθηκαν τα δέντρα αποφάσεων, δεδομένου ότι κάνουν αυτοματοποιημένη

επιλογή παραμέτρων, δεν απαιτούν γραμμική συσχέτιση μεταξύ των μεταβλητών και είναι εύκολο να κατανοηθούν και να ερμηνευθούν. Ο συνδυασμός πολλών τέτοιων δέντρων μας δίνει εν τέλει ένα πολύ καλό αποτέλεσμα. Η μέθοδος που τελικά χρησιμοποιήθηκε είναι η extremely randomized trees.

Χρησιμοποιήθηκαν πολλές μετρικές για να ποσοτικοποιήσουν τις διαφορετικές παραμετροποίησεις της εφαρμογής και του συστήματος. Καταρχάς, έχουμε το πλήθος των κόμβων και τον αριθμό των διεργασιών ανά κόμβο. Κατόπιν, έχουμε τις σχετιζόμενες με την εφαρμογή μετρικές, που είναι το πλήθος και το μέγεθος των μηνυμάτων που στέλνονται. Ακολούθως, έχουν οριστεί μετρικές που σχετίζονται με την κατανομή των διεργασιών στους πυρήνες του συστήματος. Έχουμε το πλήθος των bytes και μηνυμάτων που στέλνονται είτε μεταξύ των διεργασιών ενός κόμβου είτε από ένα κόμβο σε άλλους. Περιλαμβάνονται επίσης και μετρικές για την συνολική κίνηση που δημιουργούν όλες οι διεργασίες τις εφαρμογής συνολικά. Επιπρόσθετα, για κάθε μηχάνημα ορίσαμε κάποιες μετρικές που σχετίζονται με την αρχιτεκτονική του εκάστοτε συστήματος.

Με βάση τις παραπάνω μετρικές δημιουργήσαμε τρεις κλάσεις μηχανισμών πρόβλεψης. Η πρώτη κλάση είναι η πιο γενική και λιγότερο ακριβής και περιλαμβάνει πληροφορίες μόνο σχετικά με τα χαρακτηριστικά της εφαρμογής και το μέγεθος της χρησιμοποιούμενης κατανομής του συστήματος. Η δεύτερη εμπεριέχει επιπρόσθετα τις μετρικές που σχετίζονται με την κατανομή των διεργασιών στους πυρήνες του μηχανήματος. Η τελευταία κλάση, που είναι και η πιο ακριβής περιέχει και μετρικές που σχετίζονται με το υπό εξέταση δίκτυο διασύνδεσης.

Για να συλλέξουμε τα απαραίτητα δεδομένα για να προπονήσουμε το μοντέλο μας, εκτελέσαμε μια σειρά από ανταλλαγές μηνυμάτων μεταξύ τυχαίων διεργασιών για ποικίλα μεγέθη και πλήθη μηνυμάτων και διάφορες ποσότητες των χρησιμοποιούμενων πόρων (πυρήνες και κόμβοι).

Διασπάσαμε τα δεδομένα μας σε υποσύνολα με βάση το μέγεθος των μηνυμάτων. Η διάσπαση συνέβαλε στην αύξηση της ακρίβειας πρόβλεψης των μοντέλων μας και στην βαθύτερη κατανόηση των παραγόντων που επηρεάζουν το επικοινωνιακό κόστος στο κάθε μηχάνημα.

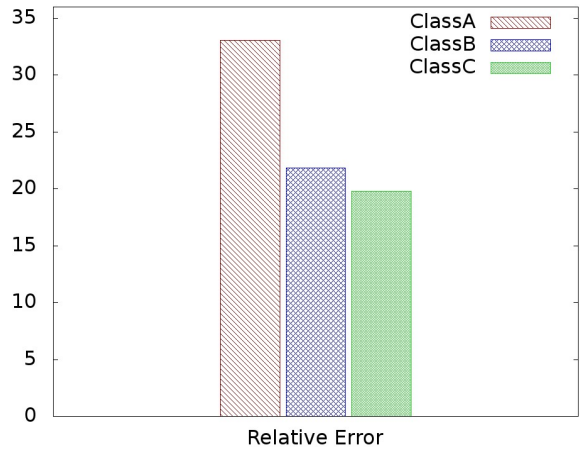
Στη συνέχεια θα παρουσιάσουμε και θα σχολιάσουμε τα πειραματικά αποτελέσματα. Δοκιμάσαμε την μεθοδολογία μας για δύο διαφορετικές εφαρμογές στα δύο μηχανήματα που είχαμε διαθέσιμα.

Piz Daint

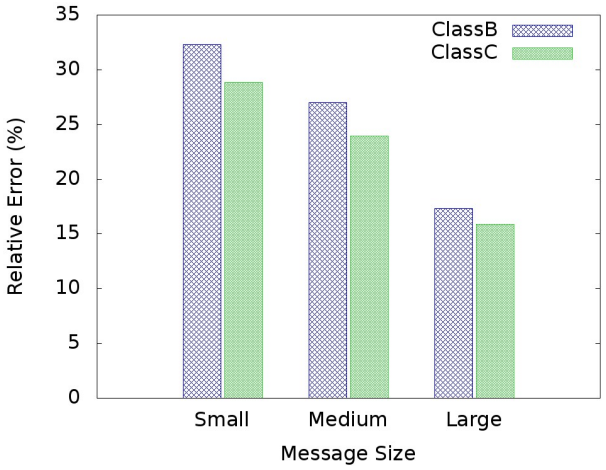
Αξιολόγηση της πρόβλεψης επίδοσης

Μέσο σχετικό απόλυτο σφάλμα πρόβλεψης

Το σχήμα 1α΄ παρουσιάζει τις διαφορές σε σχετικό σφάλμα για τις τρεις κλάσεις. Η πρώτη κλάση δίνει σχετικά καλά αλλά όχι ικανοποιητικά αποτελέσματα. Η προσθήκη των μετρικών της δεύτερης κλάσης μειώνει το μέσο σχετικό σφάλμα σε 21.85% από 33.07% ενώ η τρίτη κλάση μειώνει περαιτέρω το σφάλμα σε 19.78%.



(α΄) Σφάλμα για όλες τις κλάσεις



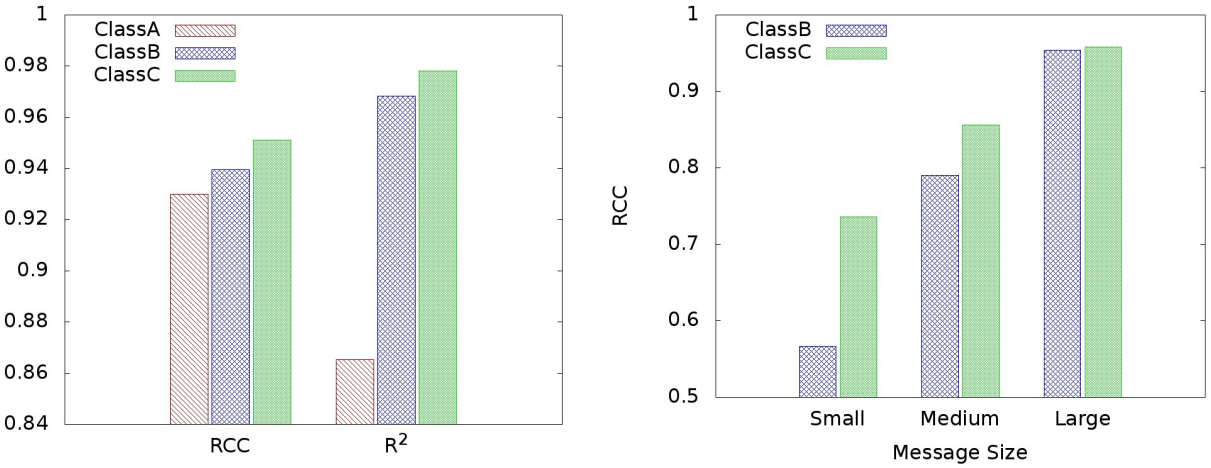
(β΄) Σφάλμα για όλα τα μεγέθη μηνυμάτων (Κλάσεις Α και Β)

Σχήμα 1: Γραφικές για μέσο απόλυτο σχετικό σφάλμα

Ως προς τις διαφορές μεταξύ των κλάσεων Α και Β, μπορούμε να επισημάνουμε ότι οι σημαντικότερες βελτιώσεις λαμβάνουν χώρα για τα μικρά και τα μεσαία μηνύματα που επωφελούνται από την προσθήκη μετρικών σχετικών με την αρχιτεκτονική του συστήματος. Από την άλλη, τα μεγάλα μηνύματα δεν επωφελούνται ιδιαιτέρως από τις νέες μετρικές.

RCC **and** R^2

Το σχήμα 2α′ παρουσιάζει τις βελτιώσεις στις μετρικές RCC και R^2 κατά την μετάβαση σε όλο και πιο εξελιγμένες κλάσεις πρόβλεψης. Η κλάση A είναι σαφώς χειρότερη από τις δύο άλλες κλάσεις σε πρόβλεψη απόλυτων τιμών, δεδομένου των γενικών μετρικών που την χαρακτηρίζουν. Αντίθετα, η δεύτερη και τρίτη κλάση πετυχαίνουν εξαιρετικά αποτελέσματα που προσεγγίζουν μία τέλεια πρόβλεψη.



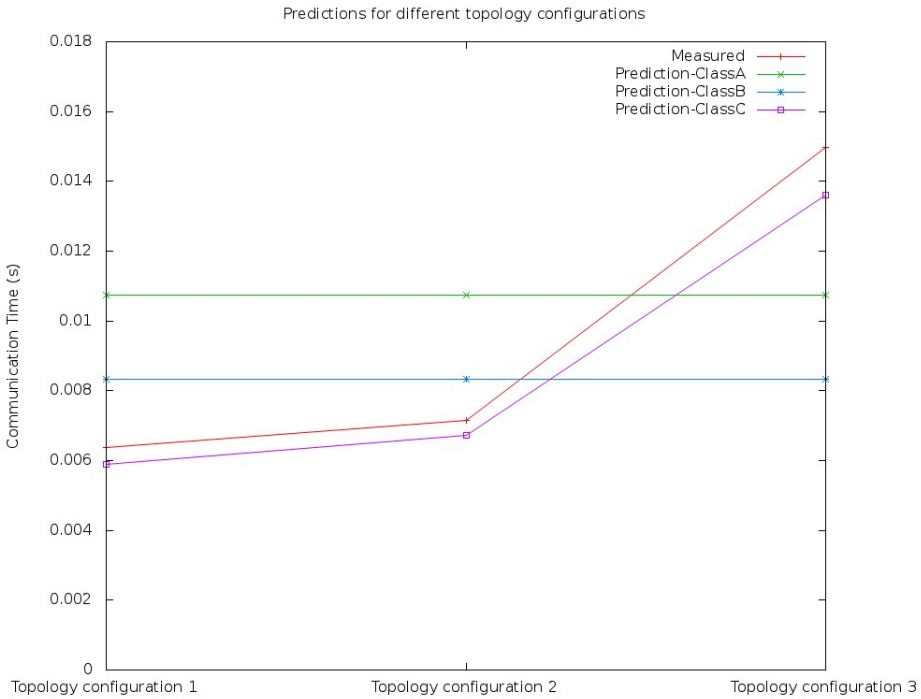
(α′) Σύγκριση των κλάσεων με βάση το RCC και R^2 (β′) RCC για όλα τα μεγέθη μηνυμάτων (Κλάσεις A και B)

Σχήμα 2: RCC και R^2 σκορ

Στο σχήμα 1β′, μπορούμε να παρατηρήσουμε ότι το σκορ στη μετρική RCC βελτιώνεται με την προσθήκη μετρικών που σχετίζονται με την αρχιτεκτονική του συστήματος κυρίως για τα μικρά μηνύματα, το κόστος επικοινωνίας των οποίων εξαρτάται από την απόσταση που διανύουν και από τα δικτυακά επίπεδα που περνούν.

Κατανόηση τοπολογικών διαφορών

Το σχήμα 3 μας παρουσιάζει γιατί η τρίτη κλάση πρόβλεψης έχει καλύτερα RCC αποτελέσματα στα μικρά μηνύματα. Οι άλλες δύο κλάσεις δεν διαθέτουν τις κατάλληλες μετρικές προκειμένου να αντιληφθούν κάποιες λεπτές διαφορές τοπολογικού χαρακτήρα μεταξύ διαφορετικών configurations , με αποτέλεσμα να κάνουν ακριβώς την ίδια πρόβλεψη και για τα τρία σημεία. Αντιθέτως, η τρίτη κλάση είναι σε θέση να ακολουθήσει αυτές τις διακυμάνσεις στο χρόνο εκτέλεσης.



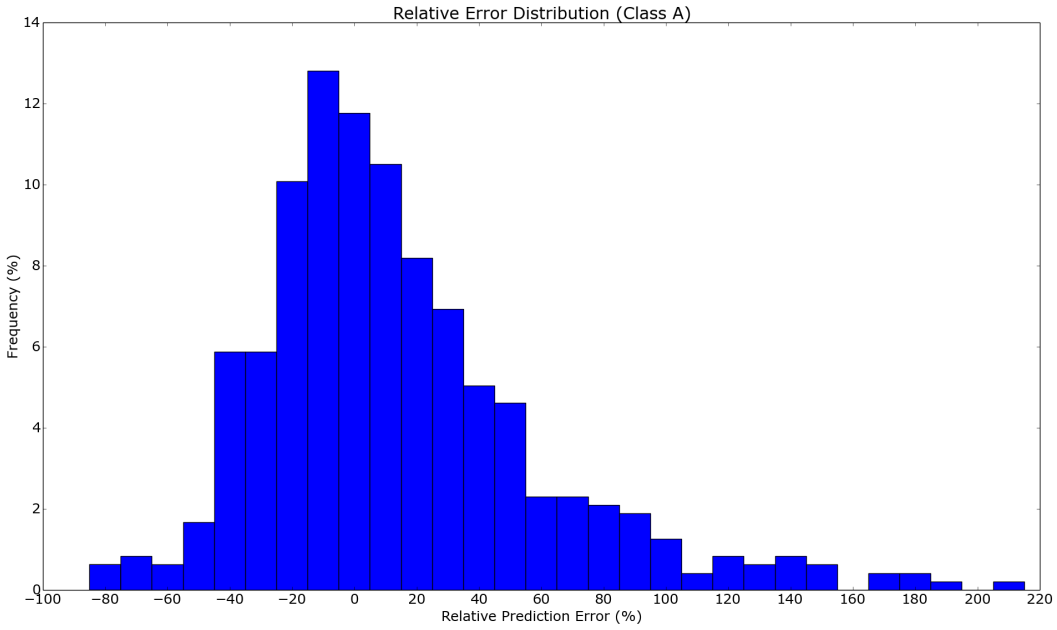
Σχήμα 3: Σύγκριση ακρίβειας πρόβλεψης για δεδομένα με διαφορετική τοπολογία

Κατανομή σφάλματος

Σε αυτήν την ενότητα θα παρατηρήσουμε πως τα σχετικά σφάλματα κατανέμονται σε κάθε κλάση και κατ ΄ επέκταση θα διαπιστώσουμε τα οφέλη της χρησιμοποίησης όλο και πιο ειδικευμένων μετρικών.

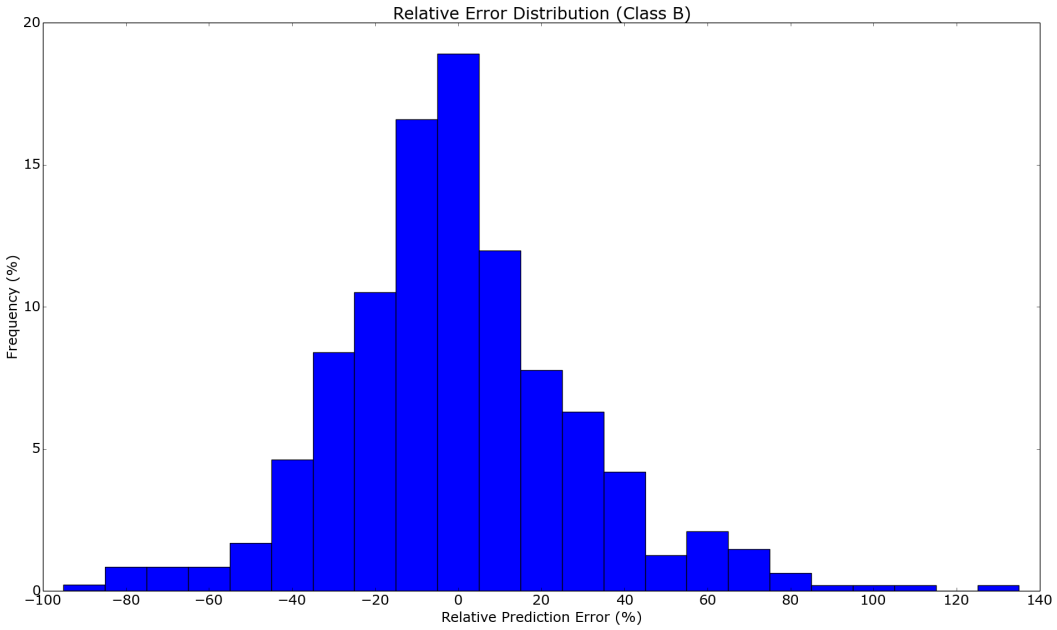
Κλάση Α

Στο σχήμα 4 βλέπουμε ότι η κλάση Α δίνει μερικές φορές σχετικά μικρά σφάλματα. Όμως σε κάποιες περιπτώσεις έχουμε πραγματοποιήσει απογοητευτικές προβλέψεις με τα σφάλματα να κυμαίνονται μεταξύ 84.19% και 211.87%.



Σχήμα 4: Κατανομή σφάλματος για την πρώτη κλάση

Κλάση Β

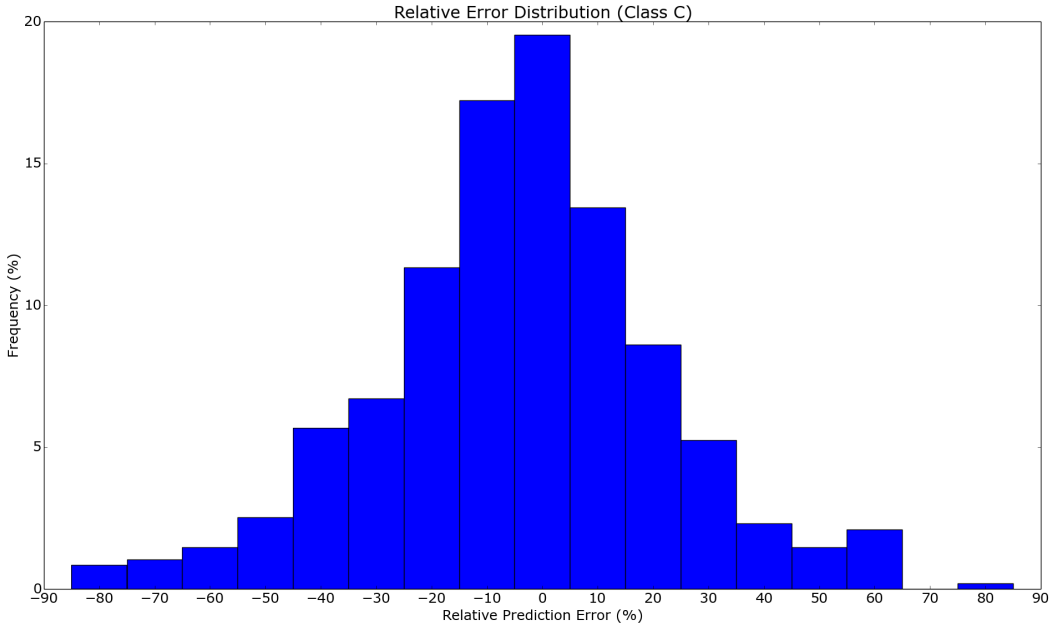


Σχήμα 5: κατανομή σφάλματος για τη δεύτερη κλάση

Με τη δεύτερη κλάση τα σημεία με σφάλματα μεταξύ -40% και 40% αυξήθηκαν σημαντικά σε σχέση με την κλάση Α. Παρόλο αυτά, σφάλματα που ξεπερνούν το 100% συνεχίζουν να υπάρχουν.

Κλάση C

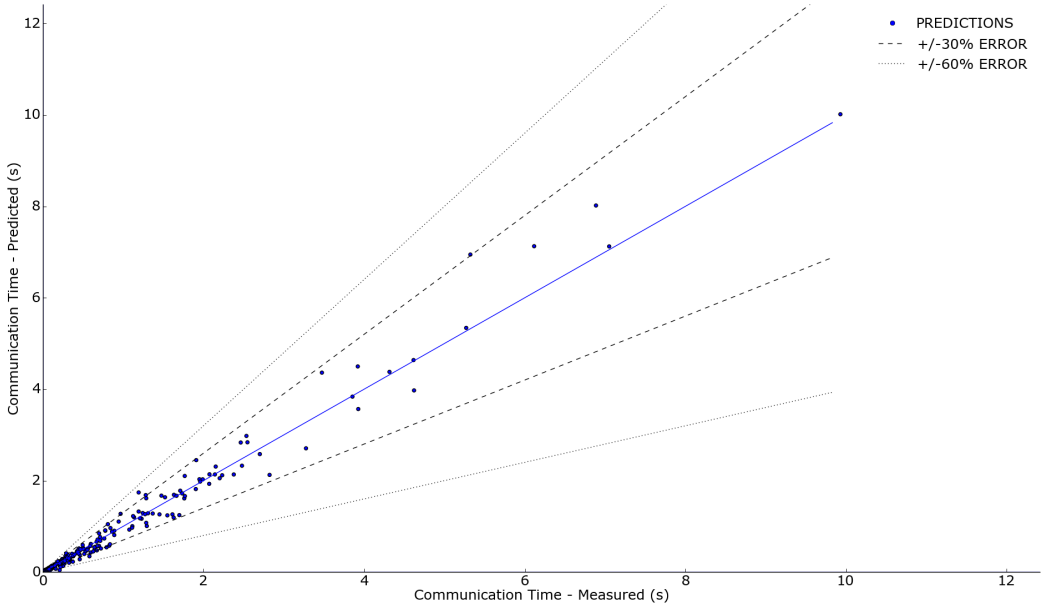
Σε αυτήν την περίπτωση πετύχαμε εξαιρετική ακρίβεια πρόβλεψης. 22% των δεδομένων προβλέφθηκαν με σχετικό σφάλμα $\pm 5\%$, ενώ περίπου 50% των προβλέψεων μας δίνουν $\pm 15\%$ σφάλμα. Επιπρόσθετα το εύρος των σχετικών λαθών περιορίζεται μεταξύ του -84.79% και 85.78%.



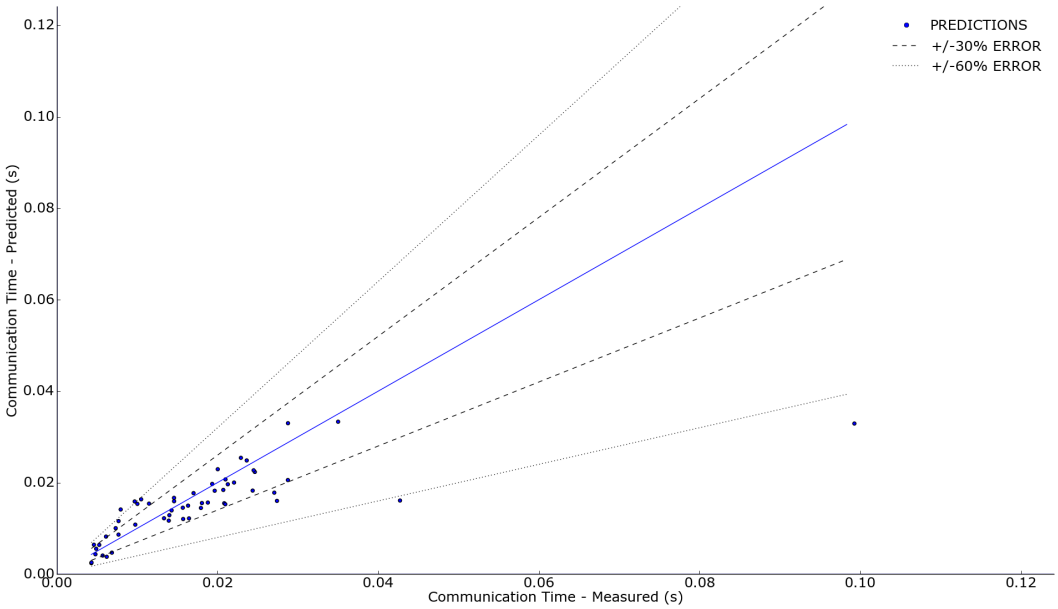
Σχήμα 6: Κατανομή σφάλματος για την τρίτη κλάση

Θα αναφερθούμε λίγο πιο λεπτομερώς στην τρίτη κλάση παρουσιάζοντας γραφικές για κάθε μέγεθος μηνύματος.

Μεγάλα μηνύματα : Από το σχήμα 7, φαίνεται ότι η πλειοψηφία των προβλέψεων βρίσκεται εντός της περιοχής $\pm 30\%$. Επίσης, υπάρχει μία ισορροπία ως προς το πλήθος των υπέρ- και υπό-εκτιμήσεων. Το μέσο σχετικό σφάλμα είναι 15.91%, ενώ το μέγιστο και ελάχιστο σφάλμα είναι 63.47% ανδ -79.19% αντίστοιχα. Η περιοχή αυτή δίνει τα πιο ακριβή αποτελέσματα.



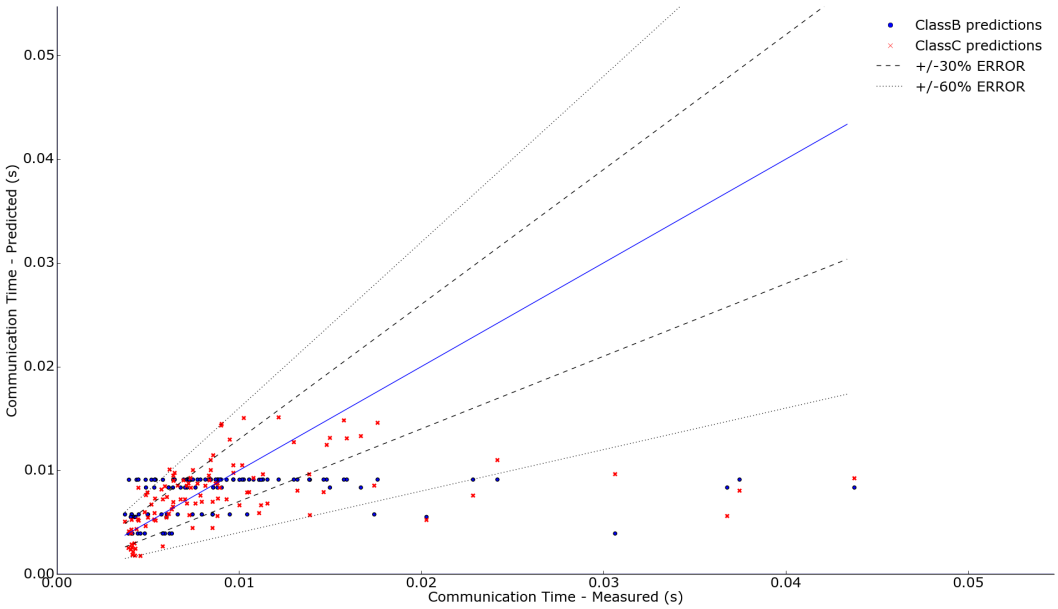
Σχήμα 7: Διασπορά των σφαλμάτων για μεγάλα μηνύματα (τρίτη κλάση)



Σχήμα 8: Διασπορά των σφαλμάτων για μεσαία μηνύματα (τρίτη κλάση)

Μεσαία μηνύματα : Στο σχήμα 8 βλέπουμε πόσο καλά είναι κατανεμημένα τα σφάλματα και για τα μεσαία μηνύματα. Το μέσο σχετικό σφάλμα είναι 23.94%, ενώ διακυμαίνεται μεταξύ του -66.84% και του 77.65%. Τα δύο σημεία με πραγματικό χρόνο επικοινωνίας μεγαλύτερο από

0.04 δευτερόλεπτα και με υψηλό σφάλμα πρόβλεψης μάλλον έχουν υποστεί θόρυβο είτε από άλλες εφαρμογές είτε από το λειτουργικό σύστημα.



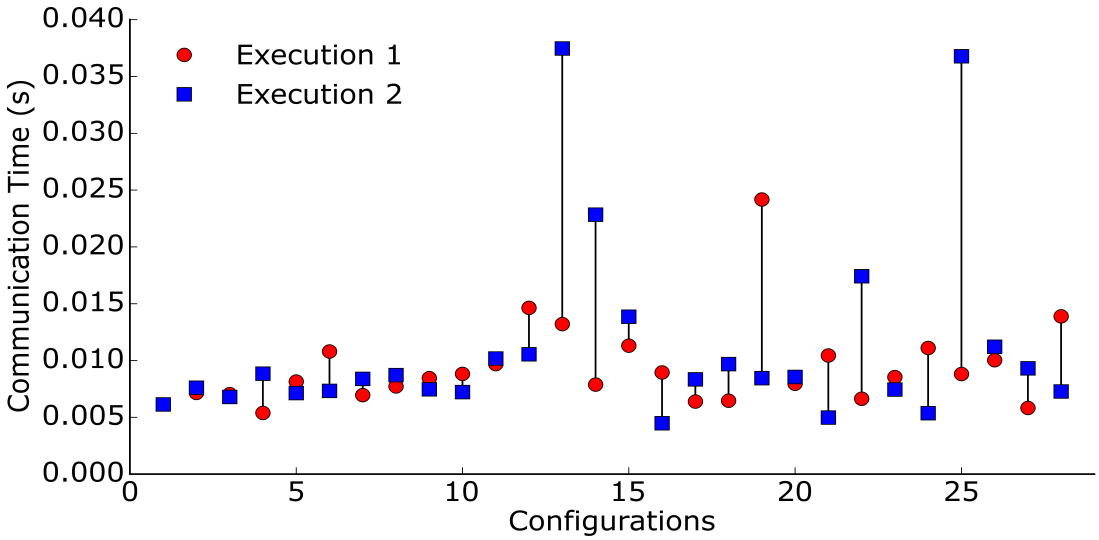
Σχήμα 9: Διασπορά των σφαλμάτων για μικρά μηνύματα (δεύτερη και τρίτη κλάση)

Μικρά μηνύματα : Για τα μικρά μηνύματα παρουσιάζουμε τη διασπορά των σφαλμάτων τόσο για την τρίτη όσο και για τη δεύτερη κλάση. Φαίνεται ξεκάθαρα η διαφορά μεταξύ των δύο κλάσεων ως προς τη δυνατότητα κατανόησης διαφορετικών τοπολογικών χαρακτηριστικών. Όπως και στα μεσαία μηνύματα παρατηρούμε κάποιες ακραίες μετρήσεις που πιθανώς να οφείλονται σε εξωγενείς παράγοντες. Το μέσο σχετικό σφάλμα είναι 28.83%, ενώ το μέγιστο και ελάχιστο σφάλμα είναι είναι 85.79% και -84.79% αντίστοιχα.

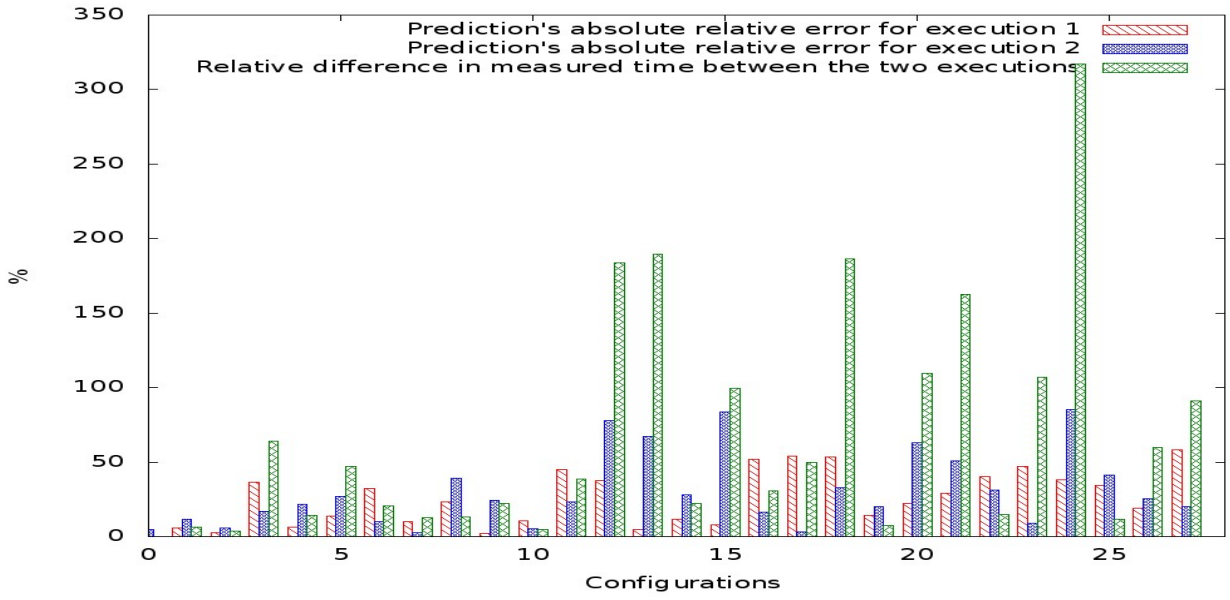
Διαφοροποίηση απόδοσης

Τα παρακάτω σχήματα δείχνουν ότι υπάρχουν μεγάλες διαφορές στο χρόνο εκτέλεσης μεταξύ δύο εκτελέσεων του ακριβώς ίδιου configuration . 30% από τα παραπάνω ζευγάρια ξεπερνούν σε ποσοστιαία διαφορά το 100% σε μετρηθέντα χρόνο, ενώ τέσσερα ξεπερνούν και το 200%. Αυτά τα δεδομένα αφορούν μεταφορά μικρών μηνυμάτων και οι διαφοροποιήσεις πιθανών να οφείλονται στην αλλαγή της πολιτικής δρομολόγησης μεταξύ των δύο εκτελέσεων από συντομότερης διαδρομής σε

δρομολόγηση μέσω ενδιάμεσου κόμβου. Από το σχήμα 11, φαίνεται ότι σε περιπτώσεις που η διαφορά μεταξύ των δύο εκτελέσεων είναι μεγάλη, το μοντέλο μας υποφέρει από υψηλά σφάλματα πρόβλεψης. Μπορούμε συνεπώς να θεωρήσουμε ότι τυχόν κακές προβλέψεις δεν οφείλονται σε αδυναμία του μοντέλου μας αλλά σε διαφοροποιήσεις επίδοσης του υπό εξέταση συστήματος.



Σχήμα 10: Διαφοροποίηση απόδοσης μεταξύ δύο εκτελέσεων του ίδιου configuration



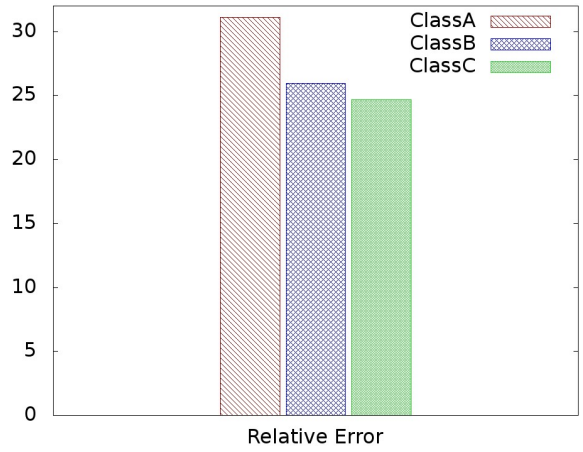
Σχήμα 11: Διαφοροποίηση απόδοσης σε συνδυασμό με σχετικά σφάλματα προβλέψεων

Vilje

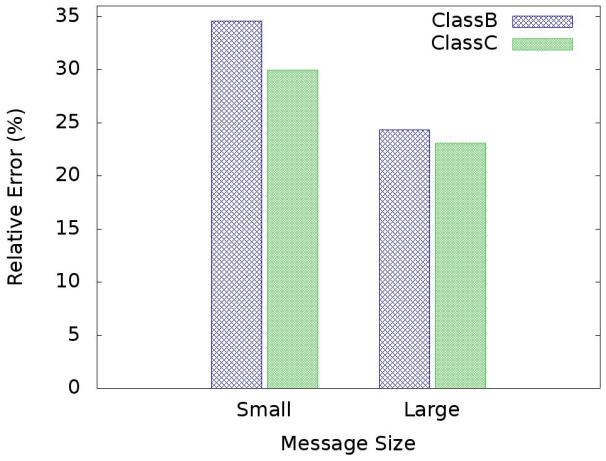
Αξιολόγηση της πρόβλεψης επίδοσης

Μέσο σχετικό απόλυτο σφάλμα πρόβλεψης

Από το σχήμα 12α΄, είναι σαφές ότι η δεύτερη και η τρίτη κλάση μειώνουν σημαντικό το μέσο σχετικό σφάλμα σε σύγκριση με την πρώτη κλάση. Μεταξύ της δεύτερης και της τρίτης κλάσης ουσιαστικές διαφορές παρατηρούνται κυρίως για τα μικρά μηνύματα, στα οποία μπορούν να προκληθούν προβλήματα συμφόρησης μόνο στα υψηλά δικτυακά επίπεδα που δεν συμπεριλαμβάνονται στις μετρικές της δεύτερης κλάσης.



(α΄) Σφάλμα για όλες τις κλάσεις

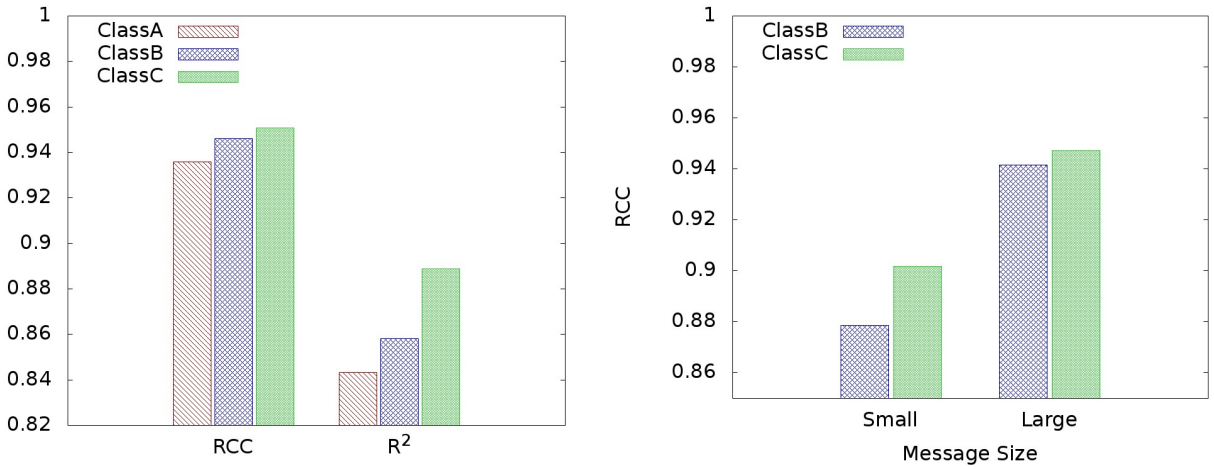


(β΄) Σφάλμα για όλα τα μεγέθη μηνυμάτων (Κλάσεις A και B)

Σχήμα 12: Γραφικές για μέσο απόλυτο σχετικό σφάλμα

RCC and R^2

Το σχήμα 13α΄ φανερώνει σημαντικές πληροφορίες ως προς τα οφέλη της ύπαρξης εξελιγμένων κλάσεων πρόβλεψης. Όλες οι κλάσεις έχουν ικανοποιητική επίδοση ως προς τη μετρική RCC και υπάρχει μικρή βελτίωση μεταξύ των κλάσεων για αυτή τη μετρική. Σχετικά με το R^2, βλέπουμε ότι υπάρχει σημαντική βελτίωση της επίδοσης με την προσθήκη όλο και περισσοτέρων μετρικών.



(α′) Σύγκριση των κλάσεων με βάση το R^2 RCC και(β′) RCC για όλα τα μεγέθη μηνυμάτων (Κλάσεις A και B)

Σχήμα 13: RCC και R^2 σκορ

Τα σκορ που πετύχαμε εδώ είναι σαφώς χειρότερα από τα αποτελέσματα που πετύχαμε στο Piz Daint . Θα προσπαθήσουμε στην συνέχεια να εξηγήσουμε γιατί δεν πετυχαίνουμε ικανοποιητική επίδοση.

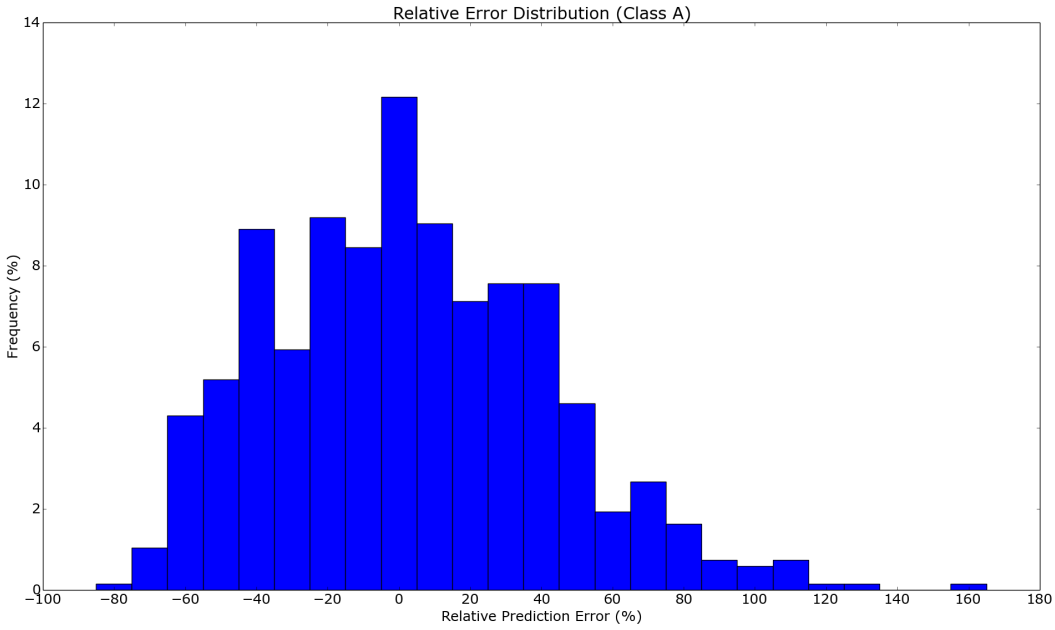
Κατανομή σφάλματος

Κλάση A

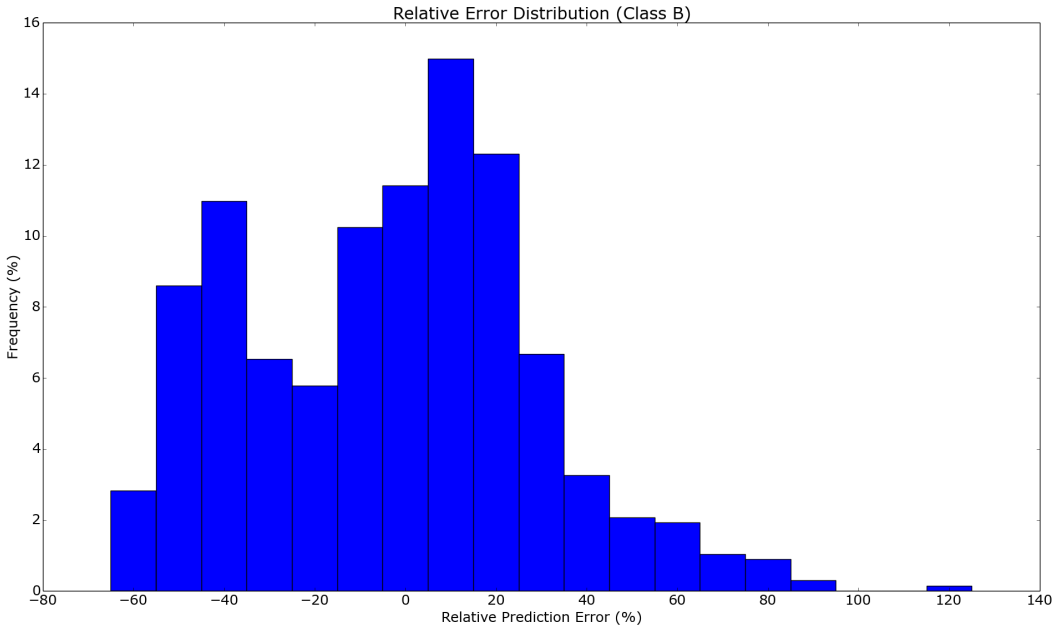
Στο σχήμα 14, βλέπουμε ότι υπάρχουν αρκετά σημεία με μικρό σχετικό σφάλμα. Όμως υπάρχουν και προβλέψεις με ακραία σφάλματα που φτάνουν τα -78.51% και 157.34%.

Κλάση B

Για τη δεύτερη κλάση, βλέπουμε πρώτον ότι τα ακραία αρνητικά σφάλματα περιορίζονται στο -64.20%, από το -78.51% που ήταν για την πρώτη κλάση. Τα ακραία θετικά σφάλματα μειώνονται επίσης σημαντικά. Πέρα από την μείωση του εύρους των σφαλμάτων, το πλήθος των μετρήσεων με σφάλμα $\pm 15\%$ αυξάνεται αρκετά και φτάνει σχεδόν το 40%.



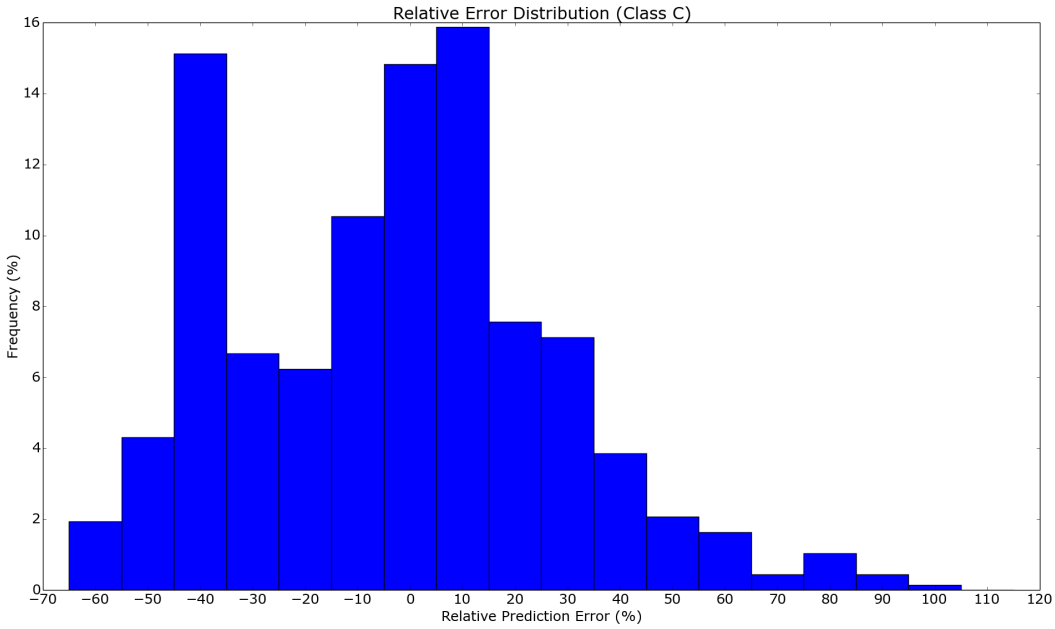
Σχήμα 14: Κατανομή σφάλματος για την πρώτη κλάση



Σχήμα 15: Κατανομή σφάλματος για την δεύτερη κλάση

Κλάση C

Το σχήμα 16 δείχνει ότι η επίδοση είναι λίγο καλύτερη σε σχέση με τη δεύτερη κλάση. Το μέγιστο θετικό σφάλμα μειώνεται από 122.33% σε 115.42%. Το ποσοστό των προβλέψεων με σφάλμα $\pm 5\%$ αυξάνεται κατά 4%. Επίσης, ο αριθμός των προβλέψεων με αρνητικά σφάλματα μεγαλύτερα από -45% περιορίζεται.



Σχήμα 16: Κατανομή σφάλματος για την τρίτη κλάση

Παρά τις όποιες βελτιώσεις σε σχέση με τη δεύτερη κλάση, η κατανομη των σφαλμάτων για την τρίτη κλάση δεν είναι ικανοποιητική. Υπάρχουν πάρα πολλές υποεκτιμήσεις του χρόνου επικοινωνίας. Αυτές οι υποεκτιμήσεις δικαιολογούν και το κακό R^2 που παρατηρήθηκε στο Vilje . Το πρόβλημα πιθανώς έγκειται στην ύπαρξη πολλών μετρήσεων που έχουν υποστεί έντονη παρεμβολή από εφαρμογές που τρέχουν παράλληλα με τη δικιά μας. Η ύπαρξη υψηλών επιπέδων παρεμβολής είναι απόρροια της σύνδεσης πολλών κόμβων σε κάθε δρομολογητή και των αραιών και ακανόνιστου σχήματος κατανομών που μας δίνονται. Η στατική δρομολόγηση δεν βοηθάει στην αποφυγή φαινομένων συμφόρησης και ακόμα και μικρά μηνύματα υφίστανται καθυστερήσεις. Είναι λοιπόν σαφές ότι είμαστε αρκετά επιρρεπείς σε παρεμβολή από άλλες δουλειές και συνεπώς έχουμε αρκετές θορυβώδεις εκτελέσεις, των οποίων τον χρόνο εκτέλεσης υποεκτιμούμε. Η υποεκτίμηση πολλών σημείων οφείλεται και στο γεγονός ότι κατά τη διαδικασία του benchmarking

απορρίπτουμε τις ακραίες τιμές και κρατάμε μία μέτρηση που είναι λίγο χειρότερη από την μεσαία περίπτωση. Συνεπώς, είναι εύλογο να υποεκτιμούμε κάποια σημεία με μεγάλο χρόνο εκτέλεσης.

Με βάση την παραπάνω επιχειρηματολογία, πιστεύουμε ότι υψηλά επίπεδα παρεμβολής οδηγούν σε κακές προβλέψεις, ενώ πιο μετριοπαθείς παρεμβολές αναμένονται να μπορούν να προβλεφθούν με ακρίβεια με βάση τα δεδομένα επικοινωνίας που έχουμε συλλέξει. Παρακάτω εξετάζουμε πειραματικά αυτήν τη σκέψη.

Διαφοροποίηση απόδοσης

Προσπαθήσαμε να εντοπίσουμε ένα τρέξιμο της εφαρμογής 3D-Jacobi που παρουσίαζε αυξημένους χρόνους εκτέλεσης σε σχέση με τα άλλα τρεξίματα. Αφού εντοπίσαμε ένα τέτοιο τρέξιμο της εφαρμογής το αφαιρέσαμε από το testing set και διαπιστώσαμε ότι το μοντέλο πέτυχε αρκετά ικανοποιητική επίδοση και σαφώς καλύτερη από πριν όπως φαίνεται στο παρακάτω πίνακα. Πιστοποιούμε επομένως ότι το μοντέλο μας μπορεί να κάνει ακριβείς προβλέψεις για μετρήσεις που δεν έχουν υποστεί μεγάλη αλλοίωση από θόρυβο.

Πίνακας 1: Ακρίβεια πρόβλεψης για διαφορετικά δεδομένα δοκιμής

Σετ δοκιμής	Μέσο σχετικό σφάλμα	Μέγιστο σχετικό σφάλμα	Ελάχιστο σχετικό σφάλμα	R^2	RCC	Μέσο απόλυτο σφάλμα
ολο	24.69%	115.42%	-61.79%	0.889	0.9507	0.1562
μειωμενο	22.90%	109.68%	-57.13%	0.934	0.9568	0.1181

Σενάριο χρήσης

Η χρησιμότητα της μεθοδολογίας αξιολογείται με βάση με την ικανότητα της να προβλέπει κρίσιμα σημεία σε κάποια σενάρια λήψης αποφάσεων. Συνεπώς, χρησιμοποιώντας το μοντέλο μας θα επιχειρήσουμε να βρούμε το μέγεθος της κατανομής που θα μας δώσει το μικρότερο χρόνο εκτέλεσης για ένα συγκεκριμένο πρόβλημα. Τα αποτελέσματα φαίνονται στον παρακάτω πίνακα.

Περιπτώσεις	Βέλτιστη κατ. (NxPPN)	Προβλεφθείσα κατ. (NxPPN)	Αποτέλεσμα	Προβλ. vs Μετρ. Σχετικό σφάλμα
Daint-3d-Jac -256^3	1024 x 4	512 x 8	Λάθος	11.4%
Daint-3d-Jac -512^3	1024 x 8	1024 x 8	Σωστό	-
Daint-3d-Jac -1024^3	1024 x 4	1024 x 8	Λάθος	8.5%
Daint-3d-Jac -2048^3	1024 x 4	1024 x 4	Σωστό	-
Vilje-3d-Jac -256^3	512 x 2	512 x 2	Σωστό	-
Vilje-3d-Jac -512^3	512 x 16	512 x 16	Σωστό	-
Vilje-3d-Jac -1024^3	512 x 16	512 x 16	Σωστό	-
Vilje-3d-Jac -2048^3	512 x 16	512 x 16	Σωστό	-
Daint-4d-Halo	1024 x 1	1024 x 1	Σωστό	-
Vilje-4d-Halo	256 x 1	256 x 1	Σωστό	-

Πίνακας 2: Εύρεση βέλτιστης κατανομής

Σε οκτώ από τις δέκα περιπτώσεις πετύχαμε το σωστό μέγεθος κατανομής, ενώ στις δύο που αστοχήσαμε υπήρχε μικρή διαφορά μεταξύ της προβλεφθείσας και της σωστής κατανομής. Επομένως, το μοντέλο μας κατάφερε να ανταποκριθεί επιτυχώς σε αυτό το κρίσιμο και σύνηθες πρόβλημα απόφασης.

Συμπερασματικά μπορούμε να κάνουμε τις εξής παρατηρήσεις ως προς τα αποτελέσματα των πειραμάτων μας:

Στο Piz Daint καταφέραμε να πραγματοποιήσουμε πολύ ακριβείς προβλέψεις του χρόνου επικοινωνίας. Δημιουργήσαμε τρεις κατηγορίες: μικρά, μεσαία και μεγάλα μηνύματα. Ο χρόνος διάδοσης των μικρών μηνυμάτων εξαρτάται κυρίως από την απόσταση που θα πρέπει να διανύσουν. Τα μεγάλα μηνύματα δημιουργούν συνήθως φαινόμενα συμφόρησης τα οποία καθορίζουν σε μεγάλο βαθμό τον χρόνο επικοινωνίας σε αυτήν την περίπτωση. Αυτά τα φαινόμενα συμφόρησης συμβαίνουν κυρίως στους συνδέσμους εισαγωγής δεδομένων στο δίκτυο. Τα μεσαία μηνύματα είναι μία ενδιάμεση κατάσταση καθώς το κόστος επικοινωνίας εξαρτάται και από την απόσταση αλλά και από τα επίπεδα συμφόρησης.

Δύο είναι οι περιοχές για τις οποίες πραγματοποιήσαμε τις χειρότερες προβλέψεις στο Piz Daint. Η πρώτη αφορά τα μικρότερα από τα μεγάλα μηνύματα, η πρόβλεψη των οποίων δυσχεραίνεται από το γεγονός ότι ενώ χρησιμοποιήθηκαν εκτελέσεις μέχρι 128 κόμβων για την διαμόρφωση του μοντέλου, το μοντέλο δοκιμάζεται με κατανομές μέχρι και 1024 κόμβους. Η δεύτερη περιοχή περιέχει μικρά μηνύματα, των οποίων ο χρόνος συνολικής μεταφοράς επηρεάζεται από την απόσταση. Δυστυχώς, διαφορετικά επίπεδα παρεμβολής από άλλες εφαρμογές οδήγησαν σε διαφορετικές πολιτικές δρομολόγησης και σημαντικές διαφοροποιήσεις στο χρόνο επικοινωνίας, παρεμποδίζοντας τις προσπάθειες μας.

Από την άλλη, για το Vilje η συμφόρηση επηρεάζει το χρόνο διάδοσης όλων των μηνυμάτων. Η ντετερμινιστική δρομολόγηση και η ανεπάρκεια εύρου ζώνης δεν επιτρέπουν το χειρισμό των καθυστερήσεων που προκαλούνται από υπέρ-χρησιμοποιούμενους πόρους δικτύου. Φαινόμενα συμφόρησης μπορούν κάλλιστα να προκληθούν από την εφαρμογή μας. Τα μικρά μηνύματα μπορούν να δημιουργήσουν συμφόρηση μόνο στα υψηλά δικτυακά επίπεδα (π.χ. στο εσωτερικό ενός rack), ενώ η επικοινωνία μεγάλων μηνυμάτων μπορεί να υπερκεράσει τους συνδέσμους εισαγωγής στο δίκτυο και να στρεσάρει και συνολικά το δίκτυο. Αυτά τα φαινόμενα συμφόρησης δυστυχώς επιδεινώνονται από εξωτερικές εφαρμογές δεδομένης της αρχιτεκτονικής του δικτύου και των κατανομών πόρων που μας δίνονται. Οι εξωγενείς εργασίες προσθέτουν επιπλέον κίνηση και συμφόρηση στο δίκτυο αναγκάζοντας ακόμα και μικρά μηνύματα να υποστούν καθυστερήσεις σε ενδιάμεσους κόμβους. Με αυτόν τον τρόπο κρύβονται συνήθως οι καθυστερήσεις που υφίστανται τα μικρά μηνύματα λόγω της απόστασης που διανύουν. Κάναμε πολλές υποεκτιμήσεις και δεν επιτύχαμε ακριβή πρόβλεψη απόλυτης τιμής του κόστους επικοινωνίας. Αυτά είναι απόρροια της απόρριψης

των ακραίων τιμών από τα δεδομένα επικοινωνίας μας και γενικότερα των διαφορετικών επιπέδων παρεμβολής ανάμεσα στα δεδομένα διαμόρφωσης του μοντέλου και στα δεδομένα δοκιμής του. Παρόλο αυτά, με την αφαίρεση μίας θορυβώδους εκτέλεσης της εφαρμογής καταφέραμε να πετύχουμε ικανοποιητικά αποτελέσματα πρόβλεψης.

Σκοπεύουμε στο μέλλον να επεκτείνουμε το μοντέλο για εφαρμογή σε ακανόνιστα μοτίβα επικοινωνίας και να το δοκιμάσουμε σε επιπρόσθετες εφαρμογές και μηχανήματα. Με αυτόν τον τρόπο θα διαπιστώσουμε την εφαρμοσιμότητα και τις προοπτικές γενικευμένης χρήσης του.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Large scale systems grow in size and in computation power constantly. Shortly, we will reach the exascale era. High level of performance at these extreme scales seems to be harder and harder. Scalability issues do not stem from the computation part which scales remarkably well by efficiently exploiting the abundance of cores, vector processing units and heterogeneous accelerators. Problems derive from the increased communication cost that can outweigh the speedup of the computation phase. The increased number of cores and network components can induce extended delays from congestion, increased synchronization costs and augmented latency due to dilation. The importance of communication time for reaching high performance is further indicated by the fact that it consists in many scientific applications up to 40% of their total execution time [1]. Having said all that, it is clear that the communication part of an application should be seriously taken into account at implementations of efficient algorithms and at the estimation of scalability and overall performance of an application. However, there are remarkably fewer attempts to predict the whole communication time of an execution than its computation part. We believe, thus, that it is worth putting considerable effort into this segment of the execution.

Accurate prediction is necessary for increasing the efficiency of HPC parallel computers. Users

should understand the application's ability to scale. There is no need for requests of large allocations that do not provide performance improvement and lead to long waiting hours on queues, waste of valuable core hours and needless spending of machine power. Sometimes, users could be overly conservative in requesting batch time making jobs fail through under estimation of the application execution time. Prediction is also necessary for better implementation of scheduling policies at large-scale systems so as to increase system throughput. Moreover, predictions are useful from an algorithmic viewpoint. Applying optimizations depends sometimes on the estimation of communication time. If the communication time lasts only a small faction of the total execution time and computation dominates, then an optimization on the communication part will be almost pointless.

1.2 Contribution

We propose a machine-learning approach to communication performance prediction for point-to-point communication of MPI applications. We use a benchmark to capture information regarding communication time on the interaction of the interconnect network and a given application and train with the benchmark data a model consisting of various features that represent application traits, allocation size, process mapping information and network characteristics. Our methodology is not dependent of a specific system architecture and can easily be applied to various interconnect networks by simply adapting some network specific features. We even devise less accurate predictors that do not need any architecture specific metrics and can more easily be applied to any underlying architecture. Our prediction model is, to the best of our knowledge, one of the first attempts to predict an entire communication phase of an application without needing run-time information (prior to execution prediction) and without explicitly modeling communication primitives and network effects. We evaluate our prediction scheme for two communication kernels and various problem sizes and system configurations. We succeed in predicting with reasonable accuracy the communication time using two systems with different network characteristics. The achieved accuracy is sufficient for supporting correct decisions on critical optimization problems, like the decision-making scenario that we present

in the end of this thesis. Our work additionally explores, through the devising of our model, the causes of performance degradation for these two platforms and possible bottlenecks of the interconnect networks that the algorithm developers should consider.

Chapter 2

Background

2.1 Communication features

The performance of the communication phase of an application depends on various parameters that need to be understood and taken into consideration for any accurate performance model. In general, variations in performance derive from the different application characteristics and the variety of large-scale systems, which includes various interconnection networks, configurations, routing protocols and MPI implementations. In addition to these parameters, the diverse process mappings vary the communication time between different executions. These concepts are analysed more thoroughly in the following sections.

2.1.1 Network architecture

First and foremost, the execution time of communication is determined to a great extent by the underlying network architecture of the system. According to the top500 list, many diverse interconnection networks dominate: Infiband, 10G or Gibabit Ethernet, Cray Interconnect and other custom interconnects. These technologies come along with various topologies like trees (e.g. Ethernet hierarchical trees, Infiband fat trees), k-nary n-cubes (e.g. Infiband hypercubes, Cray Gemini 3D-torus, BlueGene/Q 5D-torus) and dragonfly topology (Cray XC30). The

former topology is classified as indirect, given that non-terminal switches[1] exist, while the other two are considered direct networks, meaning that every switch is terminal and has at least one node attached to it. Each of these topologies presents different tradeoffs and is characterized by a unique combination of network diameter, bisection bandwidth, path diversity between source and destination and number of nodes attached to switches. Apart from the topology, the network's technology and material define several parameters, such as latency and link bandwidth. For example, an optical fibre cable and a coaxial cable deliver data at different rates. Additional parameters that affect performance are the switch size, the buffering capacity and the architecture of the network interface on the compute node.

Performance variability due to different network characteristics was noted by Bhatele et al. [2]. It was derived that different machines with dissimilar network traits, such as on-chip and off-chip bandwidth and latency, appear to be better for different applications. Significant performance deviations were noted among the examined machines in the presence of contention too. Similar conclusions were also made in [3], [4] for six in total different architectures.

2.1.2 System Software

Another crucial factor is the system software. Scheduling policies, decisions concerning the way the machine's topology is partitioned to various jobs, process mapping within an allocation and implementations of tools, frameworks and libraries differentiate the execution time of an application.

Process mapping : Every execution of an application would lead to a certain process mapping. This mapping is determined on the one hand from user demands regarding cores, nodes, memory usage, etc and on the other hand from the system's scheduler. The scheduler will allocate a system partition depending on the current system utilization by other applications in addition to some optimization techniques for high overall system throughput or Quality Of Service requirements.

[1]The term switch is used to refer to all network nodes

Congestion : The shape of the allocated partition will determine, except for the highest possible dilation, the chances of interference from nearby jobs (examined thoroughly in the next section). The way the processes are eventually mapped on the actual system will impose where network bottlenecks could occur as a result of the application communication volume. In other words, given a process mapping, there could be network components (e.g. switches, links) which would confront an excessive amount of bytes or messages that would go beyond their capacities and result in delays, degrading the communication performance [5], [6]. The terminal switches are more prone to contention, because they face all the incoming and outgoing traffic to/from the hosted processes in their connected compute nodes. In fact, Bhatele and Kale in [6] demonstrate that latency can increase up to 8 times when multiple large messages compete for network resources reducing the available effective bandwidth. Therefore, optimized task mapping should be used to avoid link contention, especially for communication bound applications. This need was clearly remarked in [7], where topology mapping algorithms, that reduced maximum congestion and average dilation, and improved benchmarked communication performance, were implemented.

Dilation : The diverse process mapping will produce different average and maximum dilation, i.e. the number of hops made by a message traversing the network. High dilation could delay the delivery of a small message owing to multiple hops, but more likely would increase the chances of stalling a message at the numerous intermediate switches [3]. Hop count does not affect significantly communication latency for large messages. In some cases, high dilation could even add up positively to path diversity for large messages Nevertheless, the number of hops affects small and medium messages' latency, especially in the presence of contention [5], [6]. Obviously, the system's routing and flow control mechanisms play a role in the severity of congestion phenomena, the value of maximum dilation and the eventual impact of distance on latency.

Intranode communication : Except for internode communication, intranode communication should also be examined. The latter takes place over shared memory and consequently

is faster than, and in most cases overlapped by, the former. However, intranode communication is susceptible to cache coherence effects [8],[9] and contention problems on the memory bus/controllers or NUMA channels [10]. All these factors could render our model inaccurate in some occasions, if intranode communication is neglected. Therefore, the number of processes residing on different cores of the same multi-core compute node, according to the process mapping, matters to some extent.

Interference : Provided that the system is not dedicated to one application, we expect performance loss due to operating system noise and interference from other applications running concurrently on the same machine. This problem is discussed in more detail in the next section.

MPI Implementation : The implementation details of the communication library (mostly MPI) together with the system-level optimizations and the threshold for protocol switching (eager to rendez-vous) differentiate the communication performance of the system.

2.1.3 Application's attributes

Clearly, the communication performance depends also highly on the application's communication profile. Different scenarios would lead to different execution times. The processes could communicate irregularly or in synchronized phases. The number and the sizes of messages may remain constant for every process or not and the communication pattern could be point-to-point, collective or both. Moreover, the overall data volume exchanged and the distribution of this volume to processes are performance factors as well. For instance, a large amount of messages transferred at once or frequent collective communication operations indicate that network contention problems will arise if the target machine's network bandwidth is not high enough.

Following sections : Later on we will see how, by gradually integrating all these factors in our prediction technique, we will get better and better predictors, achieving eventually an

accurate prediction scheme. On the next two sections, we will see some constraints that impede a highly accurate prediction and examine some related work on performance models.

2.2 Constraints

This section indicates that there are certain restrictions and factors that we are unable to monitor and include in a performance prediction attempt.

2.2.1 Interference

Provided that many applications run concurrently in a given machine, it is impossible to make an extremely accurate prediction of the communication time based on application's communication profile and current process mapping. Other jobs will undoubtedly affect the performance of our application. Although, a lot of research is already done on the direction of understanding the effects of interference on clusters and of improvement of scheduling policies, optimal scheduling is an NP-complete problem and we should sometimes except suboptimal performance due to external factors.

2.2.1.1 Network sharing

Bhatele et al.[11] claim that the most crucial parameter of performance variability is contention for shared network resources. They conclude that message passing applications suffer the most from interference and notice that, depending on what application runs nearby, different message rates are measured. High communication performance is achieved for a communication bound application, when a conflicting job is computation or I/O bound rather than communication intensive. Namely, if two applications' accesses to the network are complementary to each other, then that will result in good application performance and high system throughput, since every application will be able to have the same performance as if it had exclusive access to the network [12]. To the contrary, the worst case scenario is when two communication-intensive applications

reside on the same node and use concurrently the same network components to communicate with other nodes. Furthermore, the more fragmented the allocation is, the more variability in performance would there be, owing to increased chance of interference. Consequently, the difficulty of making communication time prediction would be subject to the target machine's allocation policy, which can vary greatly from one machine to the other. For example, on Blue Gene systems, a private torus is assigned to each job larger than 512 nodes or a mesh in some dimensions for smaller jobs. Therefore, other jobs do not affect a given job. On other systems, an arbitrary set of nodes, in possibly disjoint sections of the global machine topology (e.g. Vilje), is allocated to each job.

2.2.1.2 Intranode interference

The communication costs of an application which runs on some of the cores of a CMP varies greatly depending on the application that resides on the rest of the cores [13]. The performance is almost always better when an application runs alone and the rest of the nodes are idle. Optimal performance is nearly reached if the concurrent application does less communication or if its nature and phases of communication are orthogonal to the other's. Nevertheless, if the two prerequisites mentioned above are not satisfied, processes will compete for shared hardware such as network interface card or switches and they will experience higher communication costs.

It should be pointed out, however, that a lot of research is done towards the reduction of conflicts for memory and disk resources, and additionally, cache and memory bus contention are being addressed by vendors. Thus, the effects of intranode interference are not so severe as those of an interference in a network level.

2.2.2 OS noise

Hoefler et al. [14] indicated that both collective operations and point-to-point communication render an application's performance more noise sensitive and that noise becomes a bottleneck at some point, impeding benefits from faster networks. According to [15], when we have collective-

communication operations or global synchronization then a small delay of one process due to system noise (e.g. interrupts, operating system daemons) will delay the whole application greatly increasing communication costs. On the other hand, Bhatele et al. [11] claim that little performance variability derives from OS jitter.

To us, it seems that, system noise should be considered for applications with frequent barriers, and the existence of opposite opinions could be justified by the different effect of system noise on performance depending on the target machine.

2.3 Modeling and Prediction

One of the first and very successful model of describing point-to-point communication on parallel machines was Hockney's model [16]. According to this model, the communication latency can be expressed as :

$$t = t_0 + \frac{m}{r_\infty}$$

where t_0 is the startup time, m is the message length and r_∞ is the maximal bandwidth achievable when the length of the message approaches infinity.

Another machine-independent and communication protocol agnostic model for communication cost - in fact an extension of Hockney's model - is the LogP model [17]. This simple model included only four parameters: latency(L), overhead(o) - i.e. length of time that a processor is engaged in sending or receiving a message - , communication bandwidth(g) and number of processors(P). The simplicity of the model makes it useful for detailed algorithmic analysis, parallel algorithmic design and characterization of the scalability of a machine.

Many extensions of the LogP model were proposed over the years. One of the first ones was the LogGP [18] which incorporated long messages into the model by adding an additional parameter G. This extra parameter captures the obtained bandwidth for long messages. The most important advantages of the LogGP model over simple latency-bandwidth models is the ability to model network pipelining and computation/communication overlap. Supplementary

parameters were added to the last model as well.

The LogGPS model [19] included also the parameter S, which indicated the threshold for message length above which the rendezvous protocol of MPI was applied. By considering the synchronization costs that occur for long messages, according to high-level communication libraries, the accuracy of the model improved. A different attempt to extend the aforementioned techniques was the LoGPC [20]. This model tried to address the issues of network contention and network interface DMA behaviour.

The parameter estimation of the LogP family was not efficiently and accurately implemented in the earlier attempts. For instance, in order to estimate the g parameter, the network was flooded with a huge number of packets and, for L parameter, second or third order errors occurred. A new accurate LogGP parameter measurement scheme (Netgauge) was proposed by Hoefler et al. [21]. This tool avoids network flooding and allows the detection of protocol changes in the underlying communication subsystem.

In spite of their benefits, analytic methods can quickly become too complex and non-applicable, and sometimes require benchmark runs on the target machine for accurate model parameters assessment. Simulation, on the other hand, can provide white-box analysis of application performance and help explore the behaviour of large-scale communication algorithms, saving on expensive real runs. Simulations could also be used to make performance prediction on future systems. The LogGOPSim [22] enables simulations in a slightly extended version of LogGPS, in addition to other LogP family models. The new version has a new parameter O, in order to model more accurately the overhead per byte, which grows with message size, and better captures communication and computation overlap. Nevertheless, the LogGOPS model ignores contention on the network and might thus underestimate communication costs. A significant contribution of this work was a tool chain for MPI profiling trace gathering and evaluation that eventually creates predictions based on the selected LogP family model. The proposed scheme allows also extrapolation of collective operations with high accuracy by rebuilding the communication pattern and simple trace extrapolation to larger communicator sizes.

All the generic models and techniques that were described above are great attempts to describe

communication in a network and give some useful insights, necessary for productive and efficient algorithmic and system design. However, they fail to incorporate computer architecture concepts that affect intra-node communication and, in general, it was proven extremely difficult to incorporate all the diverse notions proposed in a single model so as to achieve great absolute communication time prediction. The latter is illustrated by the fact that most of the extensions of LogGP tackle one or two supplementary problems instead of building an all-inclusive model. Such a model that would consider all the different parameters, if it was feasible to be created, would certainly be very complicated. But this should not be the case, as application models are supposed to be simple enough to facilitate algorithm design, reveal application and network traits, and simplify and abstract complex systems. They are not generally meant to give accurate absolute execution time predictions.

A computer architecture oriented work was done by Hoefler and Ramos [9]. They created performance models for intra-node communication in cache coherent systems that do not provide precise predictions, but rather give some helpful range of performance for algorithm design and development.

To further facilitate algorithmic decisions and address scalability issues, less general performance models needed to be created too. In [23] the importance of communication models for MPI implementations at large scale is underlined. A guideline of how simple performance models can be created, possible problems and a hierarchy of modeling approaches of various accuracies and complexities are presented. Even simple asymptotic models will yield some insight for a parallel application.

Another approach was to make application-specific performance models to examine the scalability of certain algorithms. Gahvari et al. [3] suggested some extensions to the basic latency-bandwidth model for the Algebraic Multigrid solve cycle by adding penalties to the parameters based on machine-specific constraints. In particular, they modified the latency parameter to take into account communication distance and switching delays on the interconnect and added to it a penalty to encapsulate performance degradation arising from multiple cores on a single node contending for available resources. Moreover, they proposed the replacement of the

best-case bandwidth measurement taken from the latency-bandwidth benchmark in the HPC Challenge suite with a reduced one that will approach more accurately the actual per node bandwidth. The above-mentioned work referred to torus and fat-tree interconnects. Later, they validated these results for dragonfly interconnect as well [4]. The only difference was that contention was better handled in dragonfly topology and they insisted on the distance penalty, as communication between different router groups could suffer from substantial delays in this topology.

Apart from the analytical models that were examined and used by many, there were lately some attempts to use supervised learning algorithms for communication time prediction. Jain et al. [24] tried to model the performance by utilizing communication data and network hardware counters. They examined the life cycle of a message in the interconnect of Blue Gene/Q and found various resources where delays could occur. Except for maximum dilation and bytes per link, that were used previously in literature without giving accurate predictions, they introduced some new metrics such as number of messages in injection network FIFOs and number of packets in buffers in order to take into account contention for resources other than the links. By combining these metrics they predicted with great accuracy the rank correlation of different mappings, predicting at the same time, accurately enough, the absolute performance. In a more recent work Bhatele et al. [25] improved the prediction accuracy of their previous work and successfully identified in more depth the features and associated hardware components that have the most impact on network congestion. They concluded that receive buffers on intermediate nodes, network links and injection FIFOs are primary indicators of network congestion, while dilation or hop count are less important factors. Decent rank correlation results of various mappings were observed for predictions of datasets of high node count in one case and production applications in the other, using training set for smaller node count and communication kernels respectively. Finally, it was noted that the importance and the effectiveness of features vary greatly depending on the application's code, and a feature selection attempt was made to identify a subset of features that performs well for a variety of applications.

2.4 Comparison of prediction models

In the previous section, we have presented all the state-of-art prediction methods. Now, we will rank them based on four criteria and state where our work stands. The four dimensions that we will examine are accuracy, prediction overhead, whether or not the technique is platform independent and application applicability (i.e. whether it is application independent or it needs to be tuned for every new one).

Accuracy

Bhatele et al. [25] have presented extremely accurate results at some cases when the training and testing set consisted of various task mappings of a specific application for a fixed message size and node count. The accuracy is decent even when mixed datasets from different applications and configurations are used. The achieved accuracy, though, is in reality deceptive, because they actually run the applications whose performance they want to predict so as to gather the desired features. On that grounds, we cannot give credit to this work for ahead-of-execution performance prediction. We should, yet, underline the great accuracy of this work in identifying the causes of network congestion and eventually the features that determine the communication time. All the other methods, previously presented, do not provide an accurate absolute prediction of communication time. Naturally, some attempts like [3] achieve better results than the basic models, but as specified earlier, all these models provide some insights for better understanding of applications, algorithms and systems and each and every one of them neglects some parameters of performance loss or take rough estimates of them. Our work's accuracy is definitely better than the LogP family models and bandwidth-latency ones and approaches sometimes the accuracy of [25].

Prediction Overhead

All the analytical models that were described need some real runs on the target machine for parameter estimation. However, these runs could be prior to the execution of the application. Thus, the overhead is not on the critical path. On the other hand, Bhatele et al. use hardware performance counters data for some features and need actual runs of the application in question. As a result, this method involves a massive overhead, which makes it highly impractical. Our prediction scheme occurs ahead of execution (after node allocation and process mapping) causing no additional performance cost.

Platform Independence

The analytical models can be applied to any platform. Nevertheless, the parameters should be assessed every time (not so cumbersome thanks to tools like Netgauge). The techniques presented in [25] and [24] apply only on machines where hardware counters exist and are accessible. Moreover, the scheme is validated only on one platform using an isolated allocation. As far as our work is concerned, we use features that are related to application's communication pattern and generic interconnect characteristics. We applied our methodology with successful results to two machines of different network architecture with trivial modifications on platform-related features. On that account, our technique could be regarded as machine independent.

Application Independence

For LogP family and bandwidth-latency models a new model should be built for each application. The LogGOPSim using the application trace assists in predicting any application with the LogGOP model. Gahvari et al. proposed a performance model for a specific application. The supervised techniques ([25],[24]) are application independent, since they are primarily based on performance counters and need no knowledge of the communication pattern. In fact, they make predictions of various communication kernels and production applications. Be as it may,

in order to achieve optimal accuracy, their methods involve exhaustive search for finding the best subset of features for every application. Our attempts focus only on stencil communication patterns with two different forms (3D,4D) and various configurations (number and message sizes, node count etc.). For now therefore, we cannot be considered application independent. In future work, we plan to test our methodology on other applications to figure out where we really stand on this metric.

All in all, our technique is highly accurate with minimal prediction overhead and applicability to many platforms with little tuning. This consists therefore, a prediction scheme that it is unique in its qualities, to the best of our knowledge.

Chapter 3

Methodology

3.1 Machines description

Our prediction scheme focuses on high-level network and mapping features that are easily modified and adapted to each target architecture. We avoided low-level architectural features, such as latency and bandwidth, and obscure system-level parameters, like routing and communication algorithms, so as to avoid tiresome tuning when moving from one platform to another. Nonetheless, a good understanding of the underlying architecture of our experiment's execution platforms was vital for evaluating the performance divergence and designing the prediction technique.

3.1.1 Vilje

Vilje is a supercomputer installation at the Norwegian University of Science and Technology. It is an SGI system that consists of 1404 Intel Xeon-E5 dual eight-core nodes, interconnected with Infiniband FDR. The machine topology is an enhanced hypercube, where redundant links are added to available switch ports at the lower dimensions of the hypercube. Each switch connects to 18 nodes and each rack is a 3D hypercube with 8 switches and 144 compute nodes. The MPI library built in the system is a component of the SGI Message Passing Toolkit.

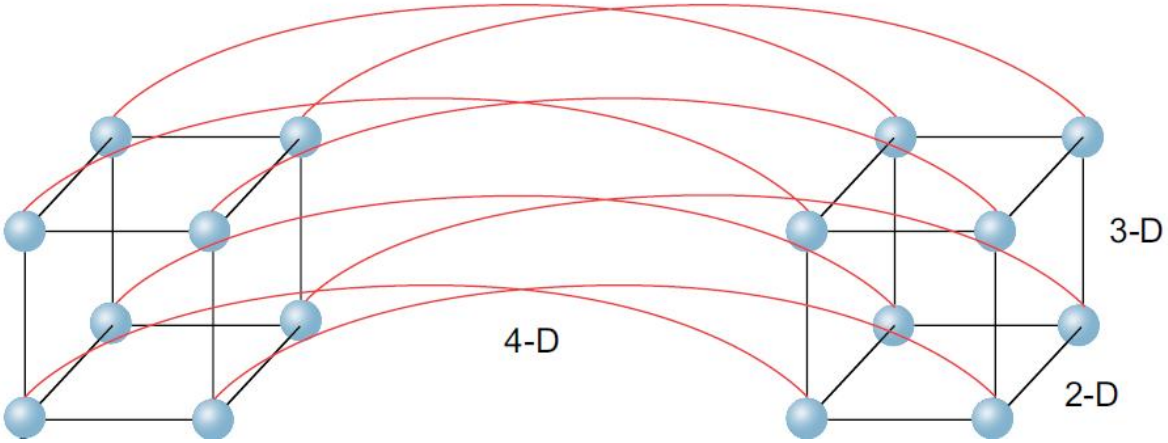


Figure 3.1: A 4D hypercube of two racks. Each vertex has 18 compute nodes.

Some policies and characteristics of this machine hinder to some point our attempts to build an accurate predictor. The configuration of the PBS scheduler does not permit specific node selection and hence, we were unable to control the allocated partition. Moreover, we were often given sparse allocations with nodes unevenly distributed on various switches and spread over different racks. This resulted in changing levels of interference from concurrent jobs and diverse process mappings of our application. Inevitably, great variability of communication performance, between different executions, was observed. This fact, in combination with limited core hour budget, high system utilization, long waiting hours and constraints faced by an average user impeded our efforts.

Furthermore, it seems from the available documentation that the used Infiband routing is deterministic and selects minimal paths without any congestion control. In particular, packets are first routed minimally to the correct position in higher dimension before continuing routing in the next dimension. This static routing will inevitably lead to unexpected stalls at intermediate switches, highly contented by our application or other external jobs.

3.1.2 Cray XC30 (Piz Daint)[1]

The Cray XC30 uses the Cray-developed Aries interconnect with Dragonfly network topology. It provides great performance in terms of bandwidth, latency and message rate, achieving in

[1]Most of the information came from [26].

the same time scalable global bandwidth.

The basic network component is the blade which consists of four nodes (dual socket Intel Xeon nodes) and an Aries router. A set of sixteen blades forms a chassis. All the Aries routers within a chassis are directly connected with each other. Six chassis are packaged in a group. Within each group a network connects, with electrical cables, each Aries of a chassis to all its peers in the other chassis of the group (Fig. 3.2).

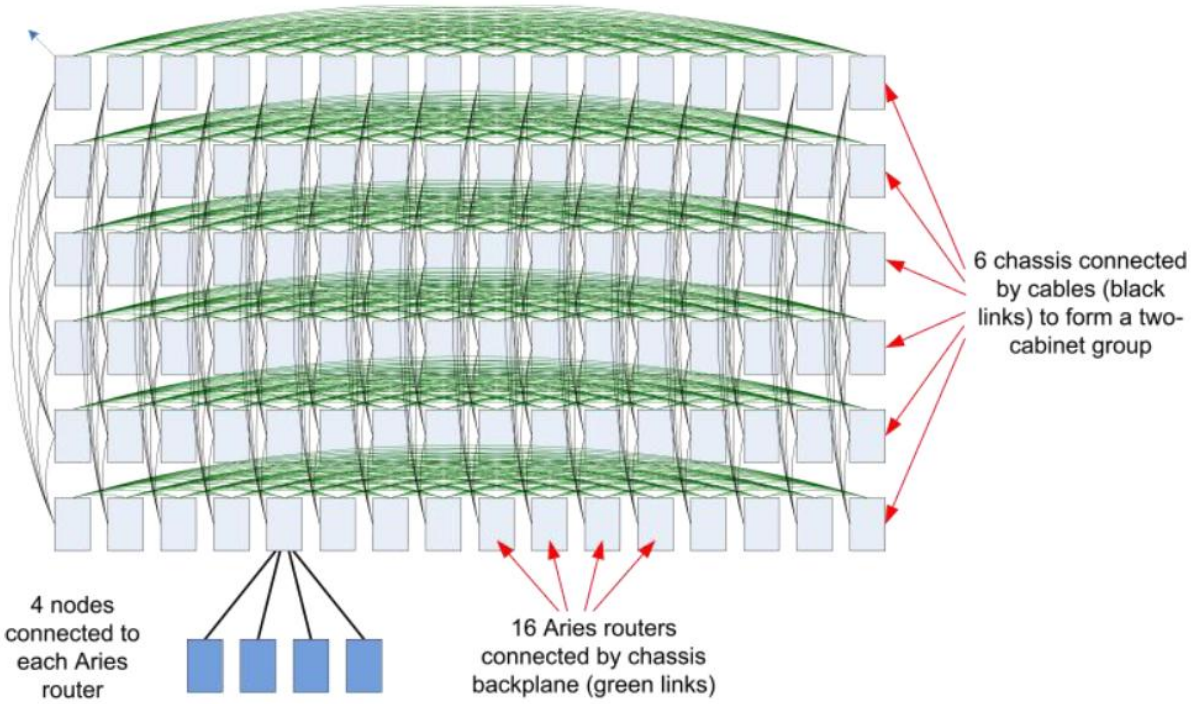


Figure 3.2: IntraGroup connections in Cray XC30 [26]

The groups are connected with an all-to-all topology network of optical links (Fig. 3.3).

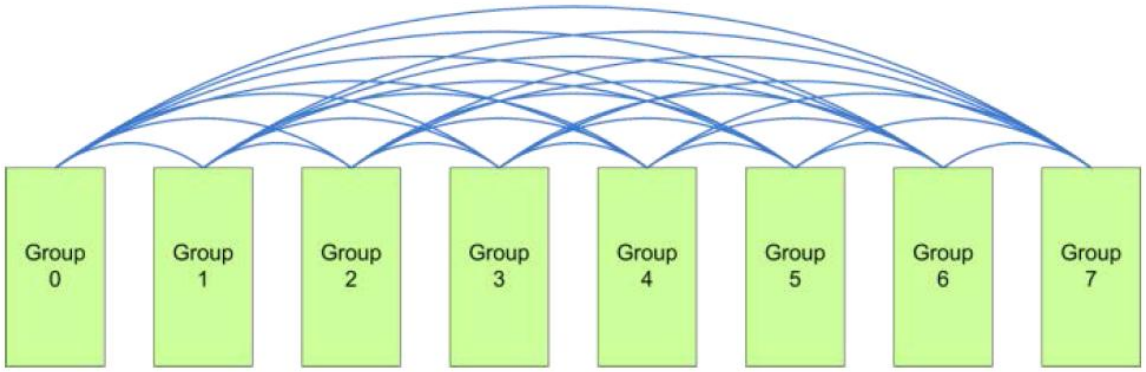


Figure 3.3: InterGroup network in Cray XC30 [26]

Aries supports a sophisticated packet-by-packet adaptive routing mechanism. Adaptive routing

selects between minimal and non-minimal paths based on load, spreading non-uniform traffic evenly over the set of available links in the system and avoiding, as a result, congestion.

When traffic is balanced, the links are equally loaded on minimally routed traffic. Nevertheless, in case there are insufficient links to achieve full bandwidth with minimal routing (e.g. in the worst case scenario where each group i sends all packets to group i+1), then non-minimal routing is applied. Non-minimal routes within a group involves routing minimally from the source Aries to a randomly selected intermediate router and then minimally to the target Aries. Similarly, global traffic is redirected through a random intermediate group in case of congestion.

Gahvari et al. [4] verified that XC30 interconnect handles congestion phenomena well and no penalties due to congestion were required, which was not the case for other networks, like Infiniband [3]. The only penalty needed was related to dilation, as both non-minimal paths and inter-group communication were worse latency-wise than minimal routes and intra-group communication respectively.

The Cray XC30, as it seems, has sufficient intra-group and global bandwidth to support full injection rate for all the nodes, thanks to adaptive routing. For that reason, communication heavy applications will probably be injection bandwidth bound, rather than network bound.

Furthermore, the constraints that we presented in the previous chapter, are expected to be less apparent on this platform. Adaptive routing allows the Cray XC network to handle a diverse set of traffic patterns at full speed by avoiding any hot spots in the network. Consequently, job to job interference would be less severe compared to other machines. In addition, the OS noise is supposed to be minimum considering the lightweight operating system that runs on compute nodes.

Nonetheless, even though sparse allocations and inconvenient process mappings on this system will not hurt performance significantly, they will certainly yield worse results than denser allocations with better process mappings.

Finally, Cray XC network provides highly optimized implementations of standard programming

environments, such as MPI.

3.2 Machine learning

3.2.1 Why use machine-learning?

Analytical methods aim to find a representative mathematical relationship between a metric, that needs to be calculated, and a set of parameters. Usually, a domain expert defines this relationship and the parameters are adjusted based on observed data. In the case of communication time, this is a strenuous task, supposing that high accuracy is required. The great variety of application traits, network-specific features, run-time configurations and MPI implementations, not to mention interference from nearby jobs and OS noise, differentiate greatly the end result. We are convinced that it is almost impossible to handle the whole spectrum and create a highly accurate, generic and analytical performance model. Yet, even for a significantly reduced scope (e.g. particular application and system) it is proven to be a challenging task. In any case, the number and the complexity of factors that can degrade performance is too high. On that account, the next logical step is to infer a model directly from data. Hence, we try to extract information, concerning the effect of the aforementioned features on communication costs, by running benchmarks on the target platform. This supervised learning approach will allow us to hide obscure network effects. Specifically, we avoid quantifying congestion on network links by capturing its consequences through the benchmarking procedure. Moreover, we hope that some of the effects of OS noise will be captured by the benchmark and there will be no need to specify explicitly in our model supplementary features related to this problem.

3.2.2 Why use decision trees?

We chose to use decision tree as our base estimator. Decision trees aim to predict the value of a target variable by learning simple decision rules based on given data. They are simple to interpret and understand since they can easily be visualized. Moreover, they indicate, in an

automated way, the features that affect communication time the most. These features will be the top few nodes on which the tree is split. Even if we use redundant features, a decision tree will make an automatic feature selection for us based on some information gain criteria. Furthermore, no form of normalization, to overcome scale differences, is needed given the tree structure, something that is not true when fitting a regression model. Decision trees are also not sensitive to outliers, because the splitting occurs based on proportion of samples within the split ranges and not on absolute values. The most important advantage of decision trees compared with other methods is, however, the fact that nonlinear relationships between parameters do not affect tree performance. In many regression models, highly nonlinear relationships between variables will result in poor results. To the contrary, decision trees do not require any assumptions of linearity in the data. Especially for our problem, this asset makes a great difference. In particular, we have noticed that many of the features, that we have selected, do not score so well on Pearson correlation coefficient (examines linear relationship with communication time), while they score a lot better on Spearman's rank correlation coefficient (examines monotonic relationship).

For visualization purposes, we present a decision tree example for determining whether somebody should go play tennis or not.

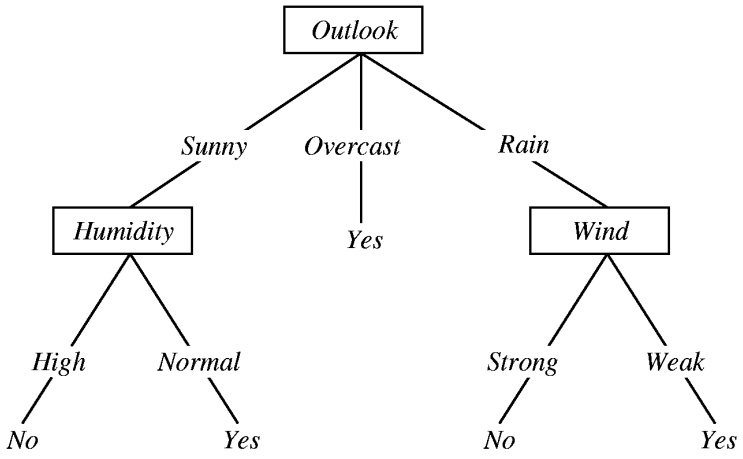


Figure 3.4: Decision tree example for tennis playing [27]

3.2.3 Why use ensemble methods?

Despite their advantages, decision trees tend to overfit the training data and end up giving very poor predictions, especially without limiting tree growth with proper pruning. An alternate choice is to combine many weak base estimators (e.g. decision trees) and come up with a stronger ensemble prediction with generalizability and robustness. By averaging different predictive models, it is likely, from a statistical point of view, to get better results and reach a globally optimum prediction - for the provided features of course - avoiding overfitting.

Table 3.1 clearly shows that using multiple decision trees improves impressively the achieved accuracy. The measurements were taken from Vilje for small sized messages using identical metrics. The evaluation metrics are thoroughly explained on section 4.4.

Table 3.1: Benefits of using ensemble methods

Number of decision trees	*Average relative error*	*Max relative error*	*Min relative error*	R^2 *score*	*RCC*	*Mean absolute error*
1	40.15%	202.22%	-74.54%	0.5487	0.8219	0.0080
100	30.60%	115.42%	-56.18%	0.8521	0.9017	0.0049

3.2.4 Ensemble techniques

There are two dominant categories of ensemble methods : averaging and boosting methods. The first one, involves building several estimators independently and then averaging their predictions for reduced variance. The other one builds base estimators subsequently. Each subsequent weak learner concentrates on the points missed by the previous ones in the sequence and thus reduces the bias of the so far combined estimator. Averaging methods work best with strong and complex models (e.g. fully developed decision trees), in contrast with boosting methods which usually work best with weak models (e.g. shallow decision trees).

3.2.4.1 Averaging methods

Two popular averaging methods, based on decision trees, are the random forests [28] and the extremely randomized trees [29].

In the former method, every tree is built using a sample drawn with replacement from the training set. Moreover, when splitting a node, instead of choosing the best split among all the features, the best split is inferred from a random subset of the features. This randomness results in significant reduction of variance compared with single decision trees.

In extremely randomized trees, not only do we use a random subset of features to find the best split, but additionally split thresholds are drawn at random for each candidate feature. To be more precise, the cut-off is chosen, at random, between the min and max value of each feature of all the samples that reach the node. The split feature is, then, selected as the best feature to split on based on the previous randomized threshold selection. This alteration further reduces variance with a slight increase in bias. In order to minimize this increase in the bias, extremely randomized trees method use the whole training set (rather than a bootstrap replica as in random forests) to create the trees. In the end, if the randomization level is properly adjusted and averaging over a sufficiently large ensemble of trees occurs, the variance will almost vanish while bias will only slightly increase with respect to standard tree. Contrarily, excessive randomization will increase bias significantly and will loosen the dependence of the output predictions on the learning sample.

3.2.4.2 Boosting methods

A very popular boosting algorithm is AdaBoost [30]. It tries to fit a sequence of weak learners on repeatedly modified data. Particularly, at each iteration we apply weights to the training samples depending on the correctness of the last prediction. If a sample was mispredicted, it will get a higher weight for the next step, while if it was predicted accurately then it will get less weight. By having each weak learner focusing on data overlooked by the previous ones, we receive a good prediction in the end.

3.3 Examined features

We examined a variety of features that could give some insight on communication performance variability. These features involved application's communication profile characteristics, user defined allocation size, process mapping information and network specific traits. It should be pointed out that all these metrics are known at runtime and can be extracted prior to execution. We evaluated their minimum, average and maximum values.

Allocation size metrics : They are explicitly requested by the user.

- **Nodes** (N) : The number of nodes defines the size of the allocation and influences parameters like dilation, path diversity and communication volume.
- **Processes per Node** (PPN) : The number of processes hosted on the same node controls both intranode and internode communication volume. High number of processes could lead to delays due to contention for locally shared network components.

Application-based metrics :

- **Message Size** (S) : The latency varies depending on the message size, and for different message sizes protocol switches occur.
- **Number of Messages per Process** (M) : Multiple messages coming from the same process could be overlapped or serialized.
- **Process Traffic** ($PT = S * M$) : Process mapping captures local contention effects and gives indications for presence of congestion on the network.

Process Mapping aware (and architecture unaware) metrics :

- **Node Traffic** (NT) : The total amount of bytes injected into the network from all the processes that reside on a node. This feature is highly correlated with communication time given its ability to identify congestion on injection links.

- **Messages Injected per Node** (NI) : Number of messages injected into the network by the processes hosted on a node. It indicates potential overlapping or serialization of messages during their injection into the network.
- **Intranode Traffic** (INT) : The total amount of bytes exchanged between processes hosted on one node. This feature could capture contention issues on the memory bus/controllers or NUMA channels.
- **Intranode Injection** (INI) : Same as intranode traffic, but measures number of messages instead of bytes.
- **Total Communication Volume** ($V = NT*N$) : The quantity of bytes flowing through the network during a communication phase. It could reveal network stress limits.
- **Total Injected Messages** ($VI = NI * N$) : Total number of messages traversing the network.

Topology and network aware metrics

Piz Daint :

- **Aries Traffic** (AT) : The sum of node traffic (measured in bytes) deriving from the nodes connected to an Aries router (up to four nodes).
- **Aries Injection** (AI) : The sum of messages sent by the nodes connected to an Aries router.
- **Intra-Aries Traffic** ($IntraAT$) : The amount of bytes exchanged between the nodes connected to an Aries router.
- **Intra-Aries Injection** ($IntraAI$) : The number of messages exchanged between the nodes connected to an Aries router.
- **Inter-Aries Traffic** ($InterAT$) : The outgoing traffic (measured in bytes) from an Aries router.

- **Inter-Aries Injection** ($InterAI$) : The number of outgoing messages from an Aries router.

- Similarly to Aries Traffic and Aries Injection, we used **Chassis Traffic** (CT), **Group Traffic** (GT) and **Chassis Injection** (CI), **Group Traffic** (GI) to quantify the volume of bytes or the number of messages sent from the processes of a chassis or a group.

- **Intra-Chassis Traffic** ($IntraCTX$) and **Intra-Chassis Injection** ($IntraCIX$) : The number of bytes and messages exchanged between Aries routers within a chassis.

- **Inter-Chassis Traffic** ($InterCT$) and **Inter-Chassis Injection** ($InterCI$) : The outgoing number of bytes and messages from a chassis.

- **Intra-Group Traffic** ($IntraGTX$) and **Intra-Group Injection** ($IntraGIX$) : The number of bytes and messages exchanged between different chassis within a group. These metrics together with Intra Chassis Traffic and Intra Chassis Injection could denote intra-group congestion.

- **Inter-Group Traffic** ($InterGT$) and **Inter-Group Injection** ($InterGI$) : The outgoing number of bytes and messages from a group. Useful for detection of overwhelmed global links and switch of routing policy (from minimal to non-minimal).

- **Maximum Single Group Traffic** ($MaxSGT$) : The maximum number of bytes exchanged between two groups. This metric can point out excessive inter-group traffic that could stress an optical global link and lead to non-minimal routing, and thus increased latency.

- **Maximum Single Group Injection** ($MaxSGI$) : Equivalent to MaxSGT but refers to number of messages instead of bytes.

- **Maximum Number of Chassis per Group** ($MaxCG$) : It suggests possible intra-group congestion.

- **Number of Groups** ($Groups$) : This feature reveals the size of allocation and implies the value of maximum dilation.

We have not included a distance related feature, since we have per-packet adaptive routing and even the packets that consist a single message could take totally different paths. Besides, metrics like Inter-Group traffic give us an idea of the maximum dilation.

Vilje : The network features for Vilje do not essentially differ from the ones for Piz Daint. Instead of having Aries routers, in Vilje, we have Infiniband network switches and instead of groups, we have racks. In Vilje, we we do not have a network components inside a rack, except for switches, while in Piz Daint, there is an intermediate network level which is chassis, a collection of routers smaller than a group (see Machines description section for more details).

- **Similarly to Aries related metrics, we have Switch Traffic** (ST), **Switch Injection** (SI), **Intra-Switch Traffic** ($IntraST$), **Intra-Switch Injection** ($IntraSI$), **Inter-Switch Traffic** ($InterST$), **Inter-Switch Injection** ($InterSI$). It should be noted also, that, in this case, up to 18 nodes (instead of 4) can introduce traffic to each switch.
- **Intra-Rack Traffic** ($IntraRTX$) and **Intra-Rack Injection** ($IntraRIX$) could suggest intra-rack congestion and contributes similarly to the combination of **Intra-Group Traffic** ($IntraGTX$), **Intra-Chassis Traffic** ($IntraCTX$) ,**Intra-Group Injection** ($IntraGIX$) and **Intra-Chassis Injection** ($IntraCIX$) in Daint.
- Equivalently to **Inter-Group Traffic** ($InterGT$) and **Inter-Group Injection** ($InterGI$) we have **Inter-Rack Traffic** ($InterRT$) and **Inter-Rack Injection** ($InterRI$), since we have racks instead of groups in Vilje.
- **Average Nodes per Switch** ($avgNSW$) : The average number of utilized nodes connected to a single switch. It could indicate the chances of extra contention for the switch.
- **Average Switches per Rack** ($avgSWR$) : The average number of switches used inside a rack. Useful for detection of intra-rack interference.

Even though the features are similar in both cases, the ones that we end up using for each platform differ greatly.

3.4 Benchmarking

The goal of benchmarking is to extract an incisive dataset that would train properly our performance model and successfully encapsulate the various network effects of the target machine. We devised our benchmark to resemble a generic application communication phase, where data streams from multiple processes flow concurrently throughout the network. We run various configurations so as to sweep the space of parameters formed by the metrics in our study. Our benchmark is built upon the WICON [6] benchmark, where random node pairs, with a single process hosted on each node, communicate in a ping-pong fashion simultaneously. WICON's goal is to measure contention and suffices for exploring the effects of message size and total communication volume. To capture the effect of many processes per node and increased node traffic, we enhanced the benchmark so that all processes hosted on a node communicate simultaneously with the processes hosted on the paired node. To break the symmetry of this scheme and stress the network, we switched from random node pairs to random rank pairs, as a worst-case communication pattern scenario. To assess the impact of the number of messages and process traffic, we paired each process with M other randomly chosen ones, resulting in a scheme where each process sends M messages of the same length to M random processes and receives a reply. We also switched from blocking to non-blocking communication, as the latter allows overlapping at the system level between consecutive message transmissions and is a common practice for most MPI applications. In the resulting benchmark, each process performs simultaneous non-blocking ping-pong with one or more randomly selected processes, where messages are of equal size. We execute this scheme for various message sizes, number of messages and processes per node on several node allocations. We evaluate our benchmark three times, to acquire diverse values for switches and racks. These parameters are beyond user control and we can only record the values reported by the system for different executions, thus we cannot get sufficient variability. The communication times reported from the benchmark are computed as follows: the core ping-pong operations are executed for a few hundred iterations, with an MPI barrier between each, to synchronize the processes and allow for more accurate time measurements. To avoid extreme outliers from OS noise, each process disposes

25% of the iterations with the highest reported communication times, so the maximum time for each process is the third quantile of all iterations. The reported communication time from the benchmark is the maximum time of all processes.

Chapter 4

Constructing and Tuning Performance Models

4.1 Comparison of methods

We tested random forests, extremely randomized trees and AdaBoost and compared their performance. We tuned each method with extensive grid search, so that we could find the best parameters for each one. We have tested them in many different subsets of our dataset and we came to the conclusion that extremely randomized trees slightly outperform the other two methods. For instance, for datasets with large messages ($message\,size \geq 16KB$) (tested on Vilje with the same features) we see from Table 4.1 that extremely randomized trees yield marginally better results than AdaBoost and random forests in every metric that we considered in our analysis. The biggest asset of the extremely randomized trees was the limitation of the range of the relative errors compared with the other two techniques.

Table 4.1: Comparison of the ensemble methods

Method	*Average relative error*	*Max relative error*	*Min relative error*	R^2 *score*	*RCC*	*Mean absolute error*
Extremely randomized trees	22.97%	71.03%	-65.04%	0.8783	0.9487	0.1986
Random forests	23.82%	77.22%	-65.33%	0.8688	0.9456	0.2091
Adaboost	23.60%	80.82%	-67.12%	0.8607	0.9448	0.2086

From a theoretical viewpoint, extremely randomized trees may sometimes perform better than

random forests. They add extra randomness in the ensemble, so that the base learners make mistakes which are less correlated to each other, leading to greater benefits from the averaging procedure. If the number of trained decision trees is sufficient, it appears that the extra randomized trees will give a more general solution. This assumption is validated by our experiments.

As far as boosting techniques are considered, our data showed that AdaBoost was a little bit worse than extremely randomized trees. On the other hand, Bhatele et al. [25] concluded that both extremely randomized trees and gradient boosted regression tree gave similar results. Anyhow, it was safe to assume that the other methods would not yield better predictions in comparison with extremely randomized trees. For that reason, we chose, ultimately, to use extremely randomized trees to devise our prediction scheme. All the experimental results, presented in the next section, were obtained utilizing this technique.

We used the Python-based *scikit-learn* (version 0.16.1) package [31] for our analysis, which provides the `ExtraTreesRegressor`, `RandomForestRegressor` and `AdaBoostRegressor` classes.

4.2 Tuning of machine learning methods

We will only focus on extremely randomized trees since it is, after all, the utilized method. Briefly for the other techniques; the tuning of random forests does not differ notably from the one of extremely randomized trees and regarding AdaBoost, the best performance was achieved when we utilized a fair number of estimators, relatively small learning rate and not exhaustively developed decision trees.

In [29], three basic tuning parameters were presented and commented on, and these are the parameters that we considered too. The first one is the number of estimators. We decided to use a large amount of trees (at least 100), so as to minimize variance and generally reduce some of the negative effects of randomization.

The second parameter is the maximum number of features which are considered at each split. In

the original paper [29], the use of all the available features is proposed for regression problems, in which high levels of bias are not tolerated and a moderate randomization level is desired. However, in our case, some features are highly correlated with each other and contain redundant information. Furthermore, some features, like switch or node traffic, have high correlation with communication time and tend to dominate, leading to biased trees. Therefore, in some cases with many features, we excluded, at each split, one feature at random, increasing, in this way, the chances of the remaining ones to be selected.

The last parameter determines the minimum number of samples required to split an internal node, and thus controls the depth of the decision trees. Normally, we want trees with considerable depth for smaller bias. How much splitting is in order is imposed by the noise in our training dataset. It is not preferable to derive predictions from outliers. In our case, a large training dataset is available and some points with abnormal execution times are present. Instead of pruning the dataset by removing outliers, we avoid them by setting this parameter equal to 5. We do not expect to lose in accuracy given the size of our training set. Besides, this specific value was proposed in [29] as well.

We validated our thoughts on parameter tuning by doing exhaustive search of a large enough parameter space.

4.3 Predictor Classes

The metrics vary from general to machine-specific ones. By considering more specific features, we lose in generality, but we gain in prediction accuracy and understanding of the factors that determine communication time. In the interest of examining this transition from the general and mediocrely accurate to the specific and highly accurate, we devised three classes of predictors.

Class A (Mapping-agnostic and Topology-agnostic) : The first one uses only features that stems from the application's profile and the user specified allocation size. This predictor will give a rough estimate of the execution time and can answer to dilemmas of whether it

is worth running an application or whether the execution will take too long. It could also be useful for ranking algorithms based on the predicted communication time, facilitating in this way algorithmic design.

Class B (Mapping-aware and Topology-agnostic) : The second class uses information gained from knowing the process mapping, in addition to the metrics of class A. The supplementary features give some estimates on the overall traffic in the network and yield some hints as far as congestion is concerned. In some cases, where traffic is really high, this predictor successfully identifies some delays and provides accurate predictions. This predictor class is suitable for user-level decisions regarding the allocations and configurations used.

Class C (Mapping-aware and Topology-aware) : The final predictor class, along with class B predictor's features, also encapsulates network-specific traits. It takes into account some network components that could become bottlenecks for performance and also evaluates the way these components are organized and connected. Given that this predictor class is the most accurate one and is related to each system, it could be valuable for system-level scheduling.

4.4 Evaluation Metrics

The success of a prediction scheme can be determined by many different evaluation metrics. We have used four metrics to evaluate our regression model.

The first one is the Rank Correlation Coefficient (RCC) which rates how well we predicted the ordering between different configurations in terms of performance. If $m_1, m_2, ..., m_n$ are the measured execution times of n samples and $p_1, p_2, ..., p_n$ are the predicted execution times, then

$$RCC = (\sum_{0 \leq i < n} \sum_{0 \leq j < i} concordant_{ij}) / \frac{n(n-1)}{2}$$

where

$$concordant_{ij} = \begin{cases} 1, & \text{if } m_i > m_j \text{ and } p_i > p_j \\ 1, & \text{if } m_i < m_j \text{ and } p_i < p_j \\ 0, & \text{otherwise} \end{cases}$$

The next three metrics compare absolute values. We used relative error to examine the performance of our model regardless of the absolute deviation of each prediction from the measured value. The minimum and maximum values will give us an estimation of how bad a prediction could eventually be, while the average absolute value will give as a mean estimation of accuracy. This measurement is computed as follows:

$$relative\,error = \frac{predicted\,time - measured\,time}{measured\,time}\ 100\%$$

Moreover, we use R^2, the coefficient of determination from statistics:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2}{\sum_{i=0}^{n_{\text{samples}}-1} (y_i - \bar{y})^2}$$

where $\hat{y}_i$ is the predicted value of the i-th sample, y_i is the corresponding measured value, n_{samples} is the number of samples and

$$\bar{y} = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} y_i$$

The last metric that we considered was the mean absolute error of predictions. This metric is not presented extendedly later on due to its high correlation with R^2. It was sometimes taken into account in feature selection.

4.5 Partitioning

We decided to partition our datasets based on the message size. This decision derived from the fact that features exhibit different correlation with communication time for subsets with different message sizes. By splitting the dataset, on the one hand, we achieve higher accuracy, while on the other hand, we get a better understanding of the different factors that affect communication time in each dataset. The partitioning, thus, moderates the basic drawback of machine learning techniques, which is the difficulty to interpret the final product of prediction.

4.5.1 Piz Daint

We chose to create three partitions for Piz Daint. The first partition consisted of configurations that involved small sized messages ($size \leq 2KB$), the second one involved medium sized messages ($2KB < size \leq 8KB$), while the last one concentrated on large sized messages ($size > 8KB$).

Small messages do not produce high communication volume and thus do not create congestion problems. As shown in [4], [6], small messages tend to exhibit great variability up to 30% in latency owing to dilation. Consequently, it seems that the communication time is determined primarily by the length of the paths that the packets follow. Medium messages consist an intermediate state. They can sometimes cause congestion, but their latency depends, to some extent, on dilation too. In the last partition, we examine transfer of messages that exceed 8KB. Large sized messages seem not to be affected by dilation as pointed in [6]. The main cause of performance variability, here, is the different levels of congestion caused by our application or other concurrent jobs. The most severe and critical congestion problems appear at the injection links rendering applications injection bound rather than network bound. These differences between the partitions are reflected by the used features presented in the next section.

4.5.2 Vilje

The biggest cause of performance degradation, in this platform, is congestion. Applications could suffer from delays on the injection points or on switches. To each switch, we have eighteen compute nodes connected. If all these nodes choose to send concurrently bytes to each other or to other switches, then the switch will not be able to handle all this traffic and significant stalls will take place. Some of the compute nodes in each switch could be used by other users and hence extra traffic could appear, hindering the quick transportation of even small messages. Switches should also handle incoming traffic to the compute nodes and forward traffic from one switch to another.

Furthermore, the routing protocols are deterministic and not adaptive, as in Piz Daint. As a consequence, there is no special care for avoiding congestion. Small messages would take the shortest path but they would probably end up stalling at some intermediate switch. Having said that, we suppose that latency due to distance will be occasionally overlapped by severe delays due to congestion.

The negative consequences of congestion that affect even smaller messages, minimizing eventually the impact of dilation in communication time, led us not to split the small messages to medium sized and smaller ones, like in Daint. We created thus just two partitions : small messages ($size \leq 4KB$) and large messages ($size > 4KB$). These two datasets exhibit quite different behaviour. Small messages tend to cause less congestion phenomena on links than large messages. Therefore, there is no need for many features that measure byte traffic. On the other hand, a strong determinant of communication time, in this area, is the number of messages that need to be processed at the network components. The higher the number of messages that we sent is, the bigger are the chances for some of them to delay at some intermediate switch owing to high congestion. As a result, features that measure number of messages and not bytes, would be more important. It should be noted that message and packet count is the exact same thing for small messages, since InfiniBand transmits data in packets of up to 4 KB. This is not the case for large messages, which are split in many packets and the presence of message size features would be necessary to convert number of messages to number of pack-

ets. For large messages, the dominant factors that regulate execution time are the degree of contention for network components, like switches or links and the injection bandwidth. Delays owing to injection bandwidth are easy to be captured given that they are caused by our application's traffic and not by external jobs as it is the case sometimes for higher network levels. Finally, a major difference between the two partitions is the MPI message passing protocol that is used. For small messages, the eager protocol is applied, an asynchronous protocol that allows a send operation to complete without acknowledgement from a matching receive, while for large messages, the rendez-vous protocol is utilized, a synchronous protocol which requires an acknowledgement from the receiver before sending the data.

4.6 Feature Selection

In this section we will present the feature selection procedure. Roughly, the procedure began by noticing the correlation of the features to the communication time for both the testing and training dataset. For correlation evaluation we used the Spearman's rank correlation coefficient[1]. This correlation calculation assesses how well the relationship between two variables can be described using a monotonic function.

Based on these correlations, we found the features that could contribute to an accurate prediction; in other words, the features that demonstrated some positive correlation with communication time. The features that were always included were the ones that had the higher correlation with communication time. Some other features with high correlation were either not included in the end because of providing redundant information, or selected with proper tuning of the ensemble method (lower max features parameter). Apart from them, some features with small correlation with communication time were also selected. They may not justify, in most occasions, differences on communication time. However, by not correlating well with the other already included metrics, they could give some additional information for specific data points. Hence, they were vital for enhancing the absolute value prediction.

[1] `http://en.wikipedia.org/wiki/Spearman's_rank_correlation_coefficient`

After the careful selection of useful features in every case, we performed an exhaustive search over these features so as to find the optimal subset, including minimum, average and maximum feature's values. The selection of the best subset was strenuous given the variety of used metrics and the absence of a subset that outperformed the others in any given metric. Consequently, we tried to figure out which subset performed almost optimally in all the metrics. It was not uncommon for some subsets to provide great R^2 and mean absolute error but relatively low RCC scores. They consisted of few metrics, and gave a good average prediction that did not produce high errors. However, they could not distinguish essentially different configurations. These subsets were generally not preferred in our selection. On top of that, we preferred mostly maximum values over average ones so as to identify network hotspots.

From the ranking of features, produced automatically by the extra randomized trees method, we can observe the metrics that contributed the most to the end result. As expected, the metrics that scored well on Spearman correlation coefficient got the top ranking positions. The ones with lower correlation scores ended up in the last positions with contributions that were even lower than 5%. Yet, the small changes and the node splits at the low parts of the decision trees, that they caused, made our model more accurate.

In the following subsections, we present the features that were eventually selected in every predictor class and in every data subset for the two used platforms.

4.6.1 Piz Daint

Class A

For the class A predictor, we wanted to create a generic prediction scheme which could be applied to any given machine and will solely be based on the application's communication profile and basic allocation information. The metrics that we used give decent accuracy no matter the target platform. In particular, we chose PPN and N to get an estimate of the given allocation, and $minS$, $maxS$, $avgM$ to get an idea of the total communication volume and encapsulate the MPI protocol switches (eager to rendez-vous).

Class B

In this class, we were able to include, except for user defined and application-specific metrics, process mapping-aware features. For medium and large messages, we pretty much chose the same metrics as congestion was the primary factor regarding performance in both cases. Node traffic and injection, and intra-node traffic metrics gave the best results. On the other hand, communicating small messages does not suffer from congestion. Therefore, these metrics were not used in this occasion. The only available metric that gave an idea of the allocation and could give a decent rough estimate of communication time was the number of nodes. Thus, N was the only feature that was used in class B predictor for small sized messages.

Class C

Small messages : In this dataset, we included, first of all, metrics that reveal the size and the shape of the allocation. These metrics are $N, MaxCG$ and $Groups$, and they give a good insight concerning the maximum dilation. For a more accurate estimation of the maximum dilation, we included three supplementary metrics. These additions were motivated by the fact that in Cray XC30 relatively great latency differences were noticed among intra-Aries[2], intra-group but inter-Aries, and inter-group traffic. The $maxInterAI$ indicates whether we have inter-Aries traffic or all the messages are exchanged within the Aries routers. The $avgInterGT$ and $avgInterGI$ declares if inter-group communication occurs and could indicate switch to non-minimal routing due to large volume of global traffic.

Medium messages : The most important feature was the $avgNT$. High values of this metric indicated congestion hazard. Three additional features ($maxIntraAT$, $maxIntraCIX$, $maxIntraGIX$) encapsulated congestion effects within a blade, a chassis and a group respectively. The fifth selected attribute was $maxSGT$. Its addition was necessary in order to take into account latency differences due to increased dilation. If it is non zero, then inter group

[2]communication between compute nodes connected to the same Aries router

communication is present and the observed latency could be affected. Furthermore, if this metric reach a high value, then a non-minimal route will be selected and the latency could sustain a further increase.

Large messages : For large messages, no features related to inter-group communication were needed. The dominant features were NT and AT. Actually, the class B predictor performed really well and the addition of network-aware metrics increased only slightly the accuracy of the prediction. Besides, according to the overview of Cray XC30, intra and inter group congestion is unlikely to happen, whereas the dominant performance degradation factor would be the injection bandwidth. On that grounds, we selected a bunch of metrics that captured congestion on the first network components ($maxNT, maxNI, avgAI, maxInterAT$). Similarly to medium messages, metrics related to congestion in other parts of the network were also chosen ($maxIntraAI, maxIntraCIX, maxIntraGTX$). Finally, we included $maxINT$, too, so as to catch contention for resources inside the compute node.

In table 4.2, we can concisely observe the used features for Piz Daint.

Classes	Partitions	Features
Class A	-	$PPN, N, minS, maxS, avgM$
Class B	$S \leq 2KB$	N
	$2KB < S \leq 8KB$	$avgM, avgNT, maxNI, maxINT$
	$S > 8KB$	$avgM, maxNT, maxNI, maxINT$
Class C	$S \leq 2KB$	$N, maxInterAI, avgInterGT, avgInterGI, maxCG, Groups$
	$2KB < S \leq 8KB$	$avgNT, maxIntraAT, maxIntraCIX, maxIntraGIX, maxSGT$
	$S > 8KB$	$avgM, maxNT, maxNI, maxINT, avgAI, maxInterAT$ $maxIntraAI, maxIntraCIX, maxIntraGTX$

Table 4.2: Selected features for Piz Daint

4.6.2 Vilje

Class A

We wanted for class A predictor to be generic and machine agnostic. Hence, we employed the exact same features with Daint's class A prediction scheme.

Class B

Small messages : The $avgNI$ will express the number of messages injected in the network and quantify its impact on delays , while V and VI will reveal the total communication volume and determine the chances of congestion on the network. We added PPN and $maxINI$ for detection of contention for local resources.

Large messages : The most important feature for predicting the communication time was the $avgNT$. This metric showed impressive correlation with execution time and was the main contributor to the construction of the decision trees, meaning that it was the attribute based on which most top tree splits happened. In addition to node traffic, some metrics that measured number of messages were also involved. $maxNI$ will indicate how many messages need to be processed and introduced to the network, $maxINI$ will identify contention for local resources depending on the number of messages and VI will give an indication of the total communication volume. Finally, we added $minS$ and $maxS$ to encapsulate MPI protocol switches and to translate number of messages to packet count.

Class C

Small messages : From the class B attributes, we kept all the metrics expect for VI. We introduced instead two features that gave more elaborate information concerning the number of messages traversing the various network levels. The first one was $maxSI$. It diagnoses stalls as a result of extreme number of messages sent concurrently by the utilized nodes of a switch. The other one was $maxIntraRIX$. It can detect delays owing to routing an excessive number of messages through intermediate switches inside a single rack.

Large messages : Class B metrics $avgNT$, $maxNI$, $maxINI$, $minS$ and $maxS$ were included in this class as well. Additionally, we added $maxINT$ as it was better combined with the new metrics than with the class B ones, and gave a marginally better prediction. Apart from these metrics, we boosted our predictor with network specific features. The most

useful and important one was $maxST$. This one hints for delays due to congestion on switch links or because of processing an enormous number of packets from the switch. $maxST$ and $avgNT$ played fundamental roles on the final structure of our model in this particular partition. Furthermore, we introduced $maxIntraST$ to capture congestion on the links between the switch and its nodes. Instead of VI, we used $avgInterSI$ which expresses the number of messages each switch injects into the network. In the interest of specifying the volume in bytes of the traffic exchanged among the switches we included $maxIntraRTX$ for intra-rack switch communication. Finally, since interference could be an issue in Vilje, we selected $avgNSW$ to evaluate the chances of having interfering jobs residing on nodes connected to switches that we use and $avgSWR$ to reveal the density of our allocation.

Table 4.3 briefly presents the features that we choose for each class and partition on Vilje.

Classes	Partitions	Features
Class A	-	$PPN, N, minS, maxS, avgM$
Class B	$S \leq 4KB$	$PPN, avgNI, maxINI, V, VI$
	$S > 4KB$	$minS, maxS, avgNT, maxNI, maxINI, VI$
Class C	$S \leq 4KB$	$PPN, avgNI, maxINI, maxSI, maxIntraRIX, V$
	$S > 4KB$	$minS, maxS, avgNT, maxNI, maxINT, maxINI, maxST, avgInterSI$ $maxIntraST, maxIntraRTX, avgNSW, avgSWR$

Table 4.3: Selected features for Vilje

Chapter 5

Experimental Results

5.1 Testing set

To evaluate the accuracy of our prediction scheme, we experimented with one parallel application kernel, the 3D-Jacobi solver. For this application, the processes are arranged in a virtual 3D-cartesian grid (p_x, p_y, p_z) and the original 3D domain (N_x, N_y, N_z) is decomposed into smaller 3D subdomains $(N_x/p_x, N_y/p_y, N_z/p_z)$. The 3D-Jacobi solver exposes a 3D-halo communication pattern, where each process exchanges a 2D-face with each of the six neighbouring processes in the 3D grid. The message sizes are obtained from the size of the 3D-domain and the domain decomposition. In any configuration, the applications exchange messages of $N_xN_y/p_xp_y, N_yN_z/p_yp_z, N_xN_z/p_xp_z$ double precision floating point elements. We predict the communication time for various problem sizes, core counts and execution configurations, from 16 up to 8192 processes. 3D-Jacobi has been executed three times in Piz Daint and four times in Vilje, to ensure that our prediction models adequately capture the variations in communication time due to varying process mapping. We added some additional runs for small sized messages, given that they are more prone to performance variability.

We supplementary tested a few configurations for a 4D Halo communication kernel to add some points with increased communication volume. This kernel uses a 4D grid of MPI processes to exchange eight messages with two neighbours in each dimension.

Our testing set comprises, in the end, of 674 for Vilje and 476 for Daint communication time measurements with their predictions. It should be noted that at some data points the node count exceeds the node count used in the benchmarking procedure. For example, for Daint the testing set includes configurations with 1024 nodes, while our training set includes for large messages up to 128 nodes and for small sizes, where the size of allocation is important, up to 512 nodes.

The computational load of both 3D-Jacobi solver and 4D-Halo is almost perfectly balanced across the MPI processes, allowing us to focus on their communication phases.

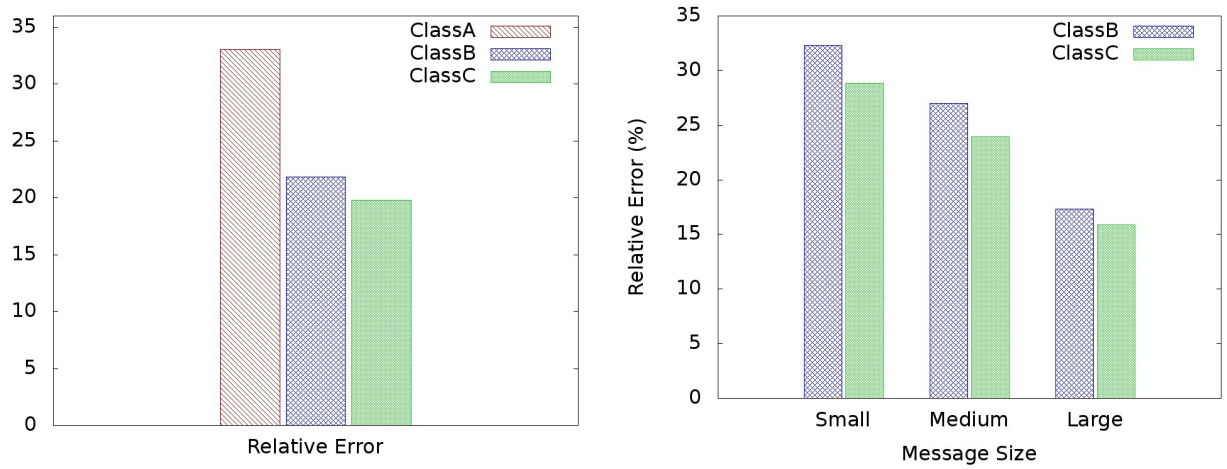
5.2 Piz Daint

5.2.1 Evaluation of prediction accuracy

We will present the achieved scores for three different metrics for all our class predictors and compare their performances. In short, we begin with a predictor with good RCC score, but low R^2 score and high average relative error, and end up with a highly accurate predictor with very high RCC and R^2 scores and remarkably low average relative error.

5.2.1.1 Average absolute relative error

Figure 5.1a presents the differences in average absolute relative error for the three classes. Class A yields a fair but not satisfactory average performance. It can easily be pointed out that there is an improvement in average absolute relative error by the addition of process mapping aware metrics. The percentage drops from 33.07% to 21.85% for class B. Class C decreases slightly further this metric to 19.78%.



(a) Error for all classes

(b) Error for all the partitions of classes B and C

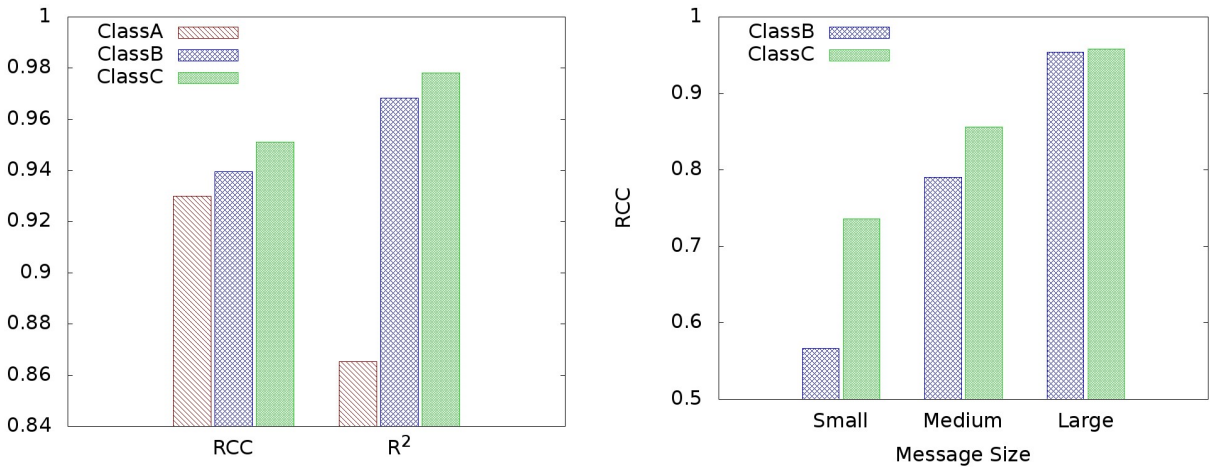
Figure 5.1: Average absolute relative error plots

It is interesting to elaborate more on the differences between class B and class C prediction accuracy. In Figure 5.1b, we notice that the biggest reduction in relative error occurs for small and medium sized messages. This was expected since in these areas network-specific metrics play an important role. On the other hand, smaller decrease is experienced for large messages, the prediction of which is not greatly boosted by the supplementary features. Furthermore, large messages consist the majority of our dataset and consequently the overall reduction in relative error is not so apparent in Figure 5.1a.

5.2.1.2 *RCC* and R^2

Figure 5.2a demonstrates the improvements in *RCC* and R^2 when moving to more advanced classes. The most impressive fact is the high difference in R^2 between class A and class B. R^2 is one of the most accurate metrics to evaluate absolute values predictions. Therefore, this big alteration is justified by the fact class A use some elementary metrics and class B use more elaborated ones. In spite of not predicting successfully the exact communication times, class A predictor can rank satisfactorily the various configurations, achieving good RCC scores. As far as class B and class C are concerned,the latter seems to outperform the former in both *RCC* and R^2. In fact, class C manages to achieve an almost perfect R^2 score. This very high score indicates that the misranked configurations, that prevented *RCC* of reaching a higher score,

differ only marginally in execution time and their ranking is thus not so important.



(a) Comparison of classes based on RCC and R^2 (b) RCC scores for all partitions of classes B and C

Figure 5.2: RCC and R^2 scores

In Figure 5.1b, it can be observed how the RCC score augments from class B to class C predictor in each partition. It is obvious that RCC score for small sized messages is considerably ameliorated by the supplementary machine-specific metrics. This is only natural, as these metrics are the ones that count the most and determine the communication time for small messages. The differences of class B and C, as far as prediction for small messages is concerned, will be more clarified with the error scatterplots in the next subsection. For large messages, class B predictor does a great job ranking different configurations. Thus, minor improvements from moving to class C are observed in this case. This small difference in performance between class B and C for large messages derives from the fact that communicating large messages is usually bounded by the injection bandwidth rendering higher level network attributes unimportant.

5.2.1.3 Topology aware prediction accuracy

Figure 5.3 illustates why class C predictor displays better RCC score than the other classes. This figure shows the measured time and the predictions from all the classes for three configurations that have the exact same volume in bytes and messages and allocation size. The only thing that changes is the shape of the allocation, namely how the allocated nodes are distributed to groups and switches. We see from Figure 5.3 that class A and class B are unable to

capture the differences in time for these three configurations. However, class C is equipped with some useful features, ideal for distinguishing topologically different data points. The presented configurations refer to small sized message communication.

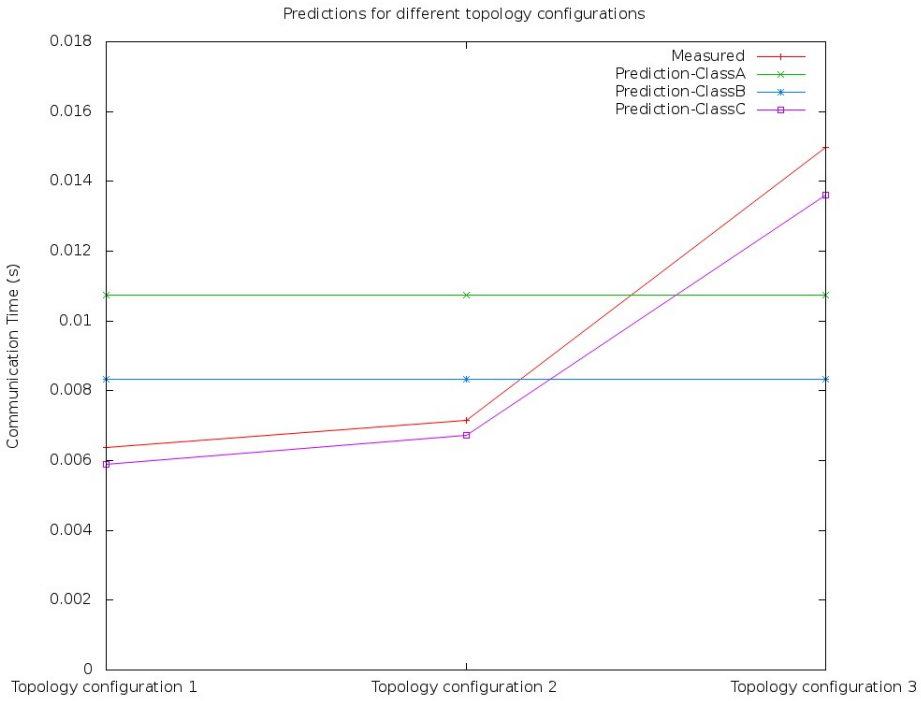


Figure 5.3: Comparison of prediction accuracy for different topology configurations

5.2.2 Error Distribution

In this subsection, we can observe how the relative errors are distributed in each predictor class and as a result notice the benefits from considering more and more specific metrics.

Class A

In Figure 5.4 we can see that even though class A predictor occasionally gives low relative errors, some data points are completely mispredicted, producing excessive relative errors ranging from -84.19% to 211.87%.

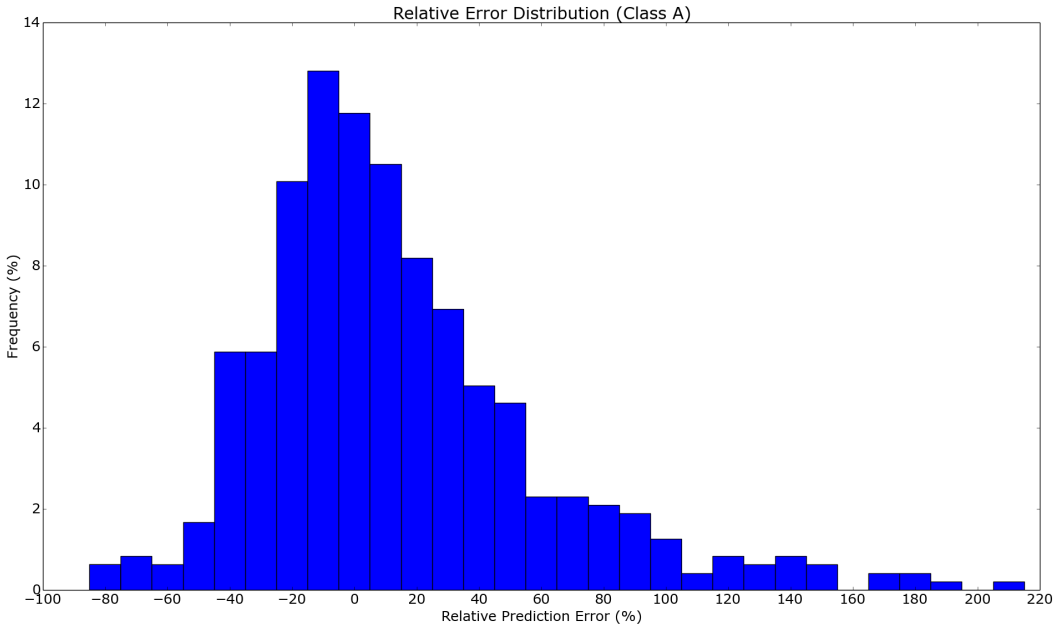


Figure 5.4: Class A error distribution

Class B

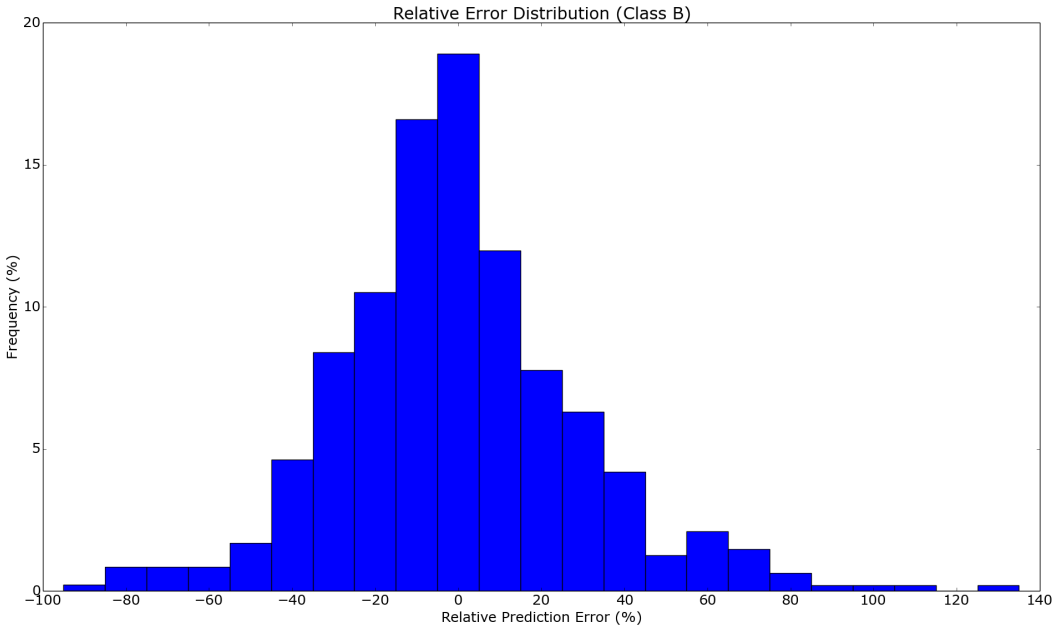


Figure 5.5: Class B error distribution

With class B predictor, the number of points with relative error that lies between -40% and 40% are significantly increased compared with class A predictor. Nevertheless, errors that exceed 100% still exist.

Class C

In this case, we achieve great prediction accuracy. 22% of the training set is extremely accurately predicted with $\pm 5\%$ relative error, while environ 50% of our predictions yielded $\pm 15\%$ error. Moreover, the range of relative errors is further limited between -84.79% and 85.78%.

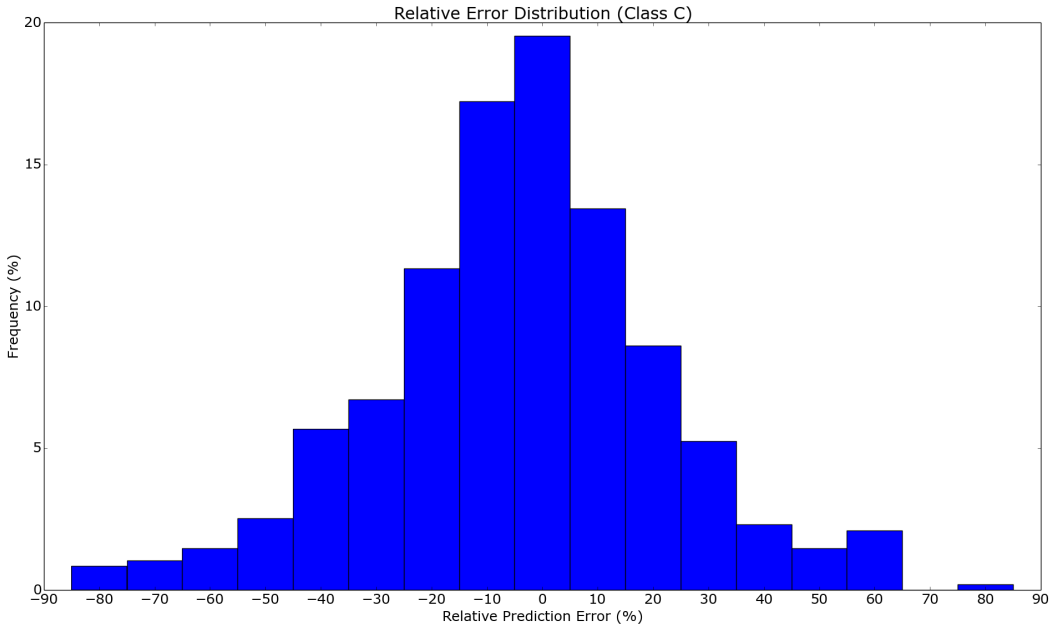


Figure 5.6: Class C error distribution

We will elaborate more on class C by presenting a error scatterplot for each partition.

Large messages : From Figure 5.7, it seems that the vast majority of predictions lie within the $\pm 30\%$ area. In addition, there is a balance regarding how many under- and over- predictions are made. The average relative error is 15.91%, while the maximum and minimum values are 63.47% and -79.19% respectively. This area yields the most accurate results. This success stems from the fact that our metrics capture effectively the congestion phenomena related to

communicating large messages on a dragonfly topology. Some inaccurate predictions, in this area, seem to happen for small execution times. This prediction inadequacy could be caused by the fact that our benchmark includes configurations with node count up to 128 nodes for large messages, while the testing set includes configurations with 1024 nodes.

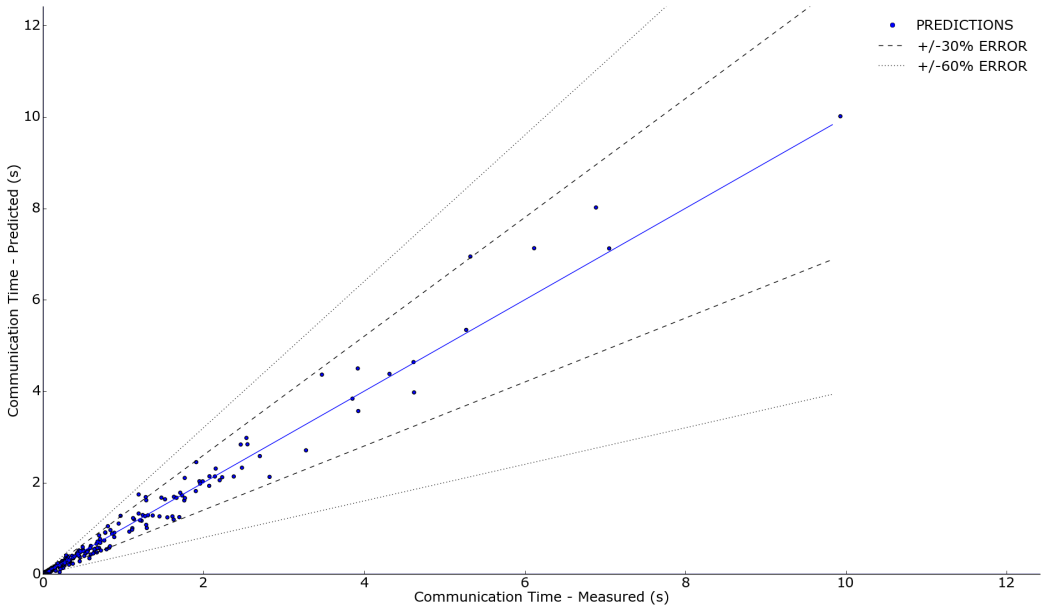


Figure 5.7: Class C error scatterplot for large messages

Medium messages : In Figure 5.8, we see how well distributed are the errors for the medium sized messages too. The average relative error is 23.94%, and the errors vary from -66.84% to 77.65%. The two data points that have measured communication time more than 0.04 s and are mispredicted with error greater (in absolute value) than -60%, possibly consist two outliers, whose execution time was affected by OS noise or excessive interference from concurrent jobs.

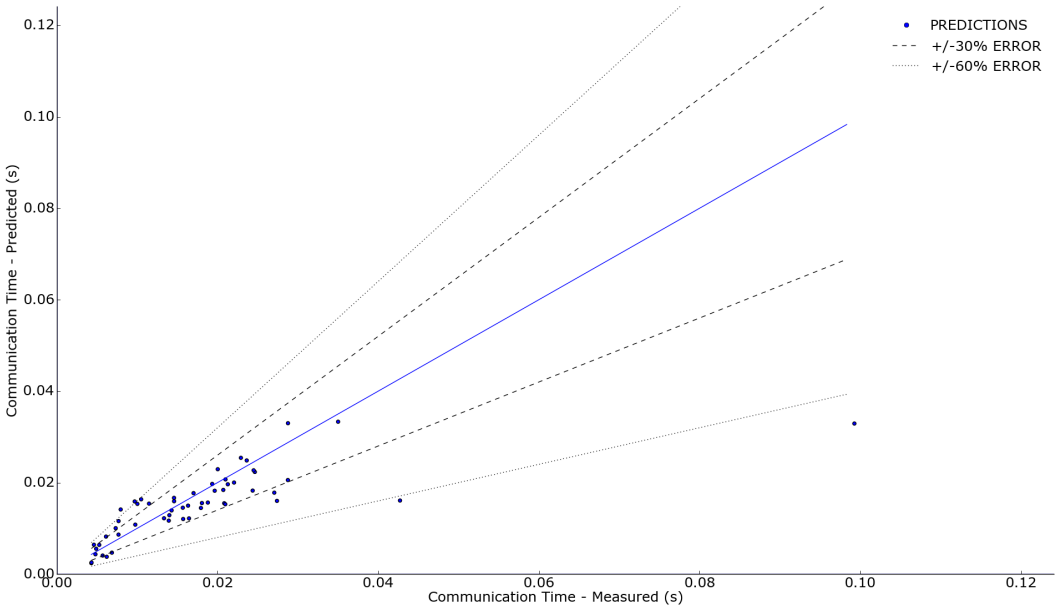


Figure 5.8: Class C error scatterplot for medium messages

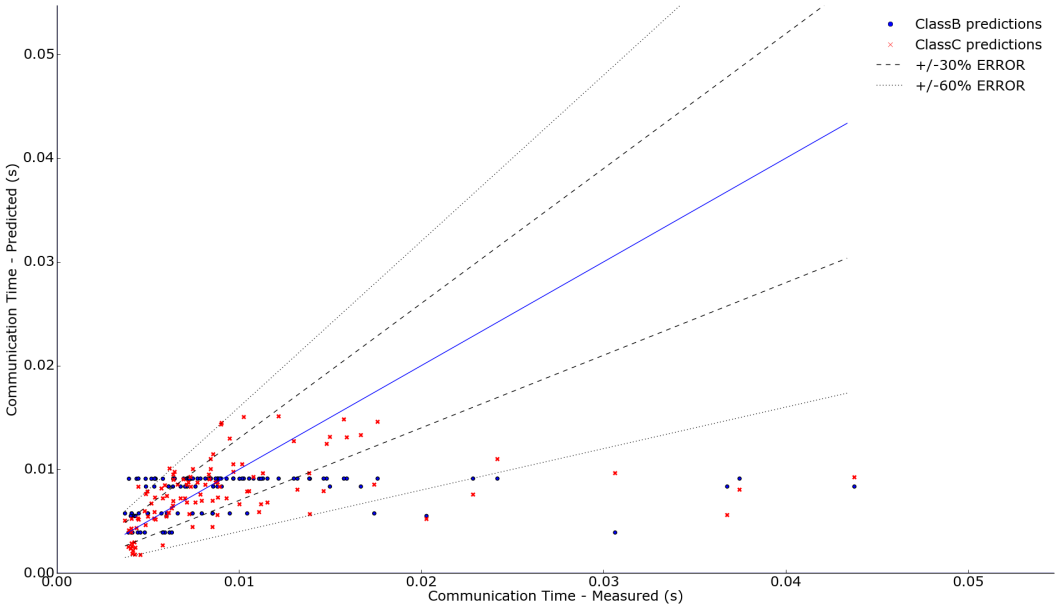


Figure 5.9: Class B and C error scatterplot for small messages

Small messages : For small sized messages, we present a scatterplot that includes both class B and C predictions. In this way, the benefits from using features related to the underlying architecture are clear. With class B, the same value is predicted for many data points with

greatly different measured execution times, whereas with class C, predictions are more balanced and distinguish different configurations. Similarly to medium messages, we observe that there are possibly some outliers that damage considerably our prediction accuracy. The presence of these outliers is validated in the next subsection. The average relative error is 28.83%, the maximum value is 85.79% and the minimum is -84.79%(owing to outliers)

5.2.3 Performance variability

The following figures clearly illustrate that there are huge differences in execution times between executions of the exact same configuration. This validates with great certainty our previous assumptions for some outliers in our testing set. 30% of these configurations exceed 100% difference in measured time, while four configurations reach or exceed 200%.

The used configurations were taken from the testing set of small sized messages. Given that the transfer time of small messages is highly correlated with dilation, we can assume that possibly in one execution, interference forces messages to take non-minimal routes while in the other execution (less busy network), messages take the shortest paths achieving the smallest possible latency. In addition, small messages need less time to be communicated and thus are more susceptible to OS noise.

From Figure 5.11, it appears that we suffer from high relative errors when high performance distortion occurs, while on the contrary, low prediction errors are observed for data points with minimal differences. We can conclude that our prediction scheme is hindered by events and factors that are external to the running application and hence are highly unpredictable.

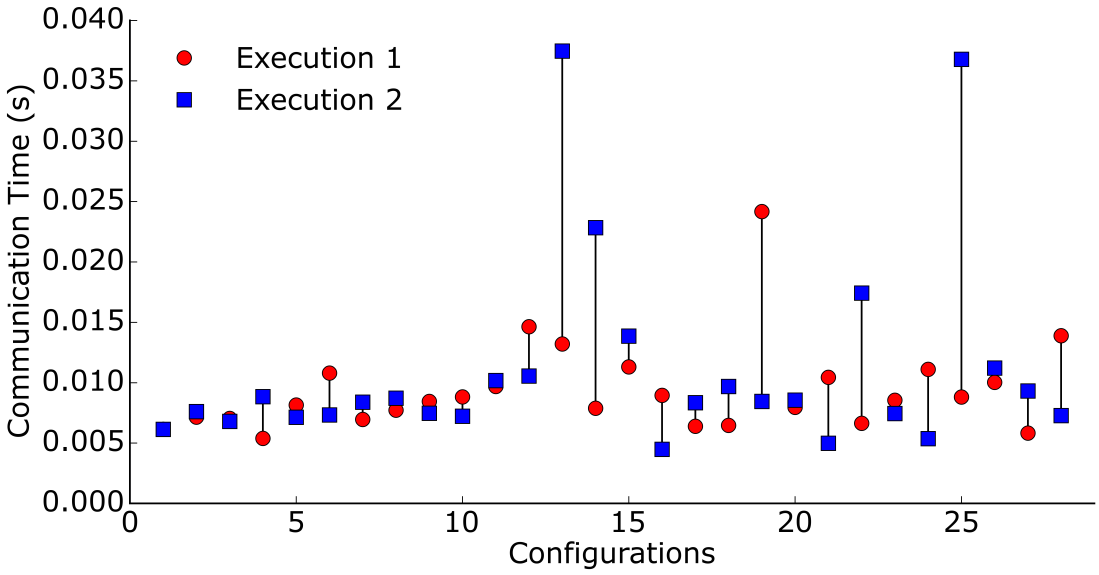


Figure 5.10: Performance differences for two executions of identical configurations

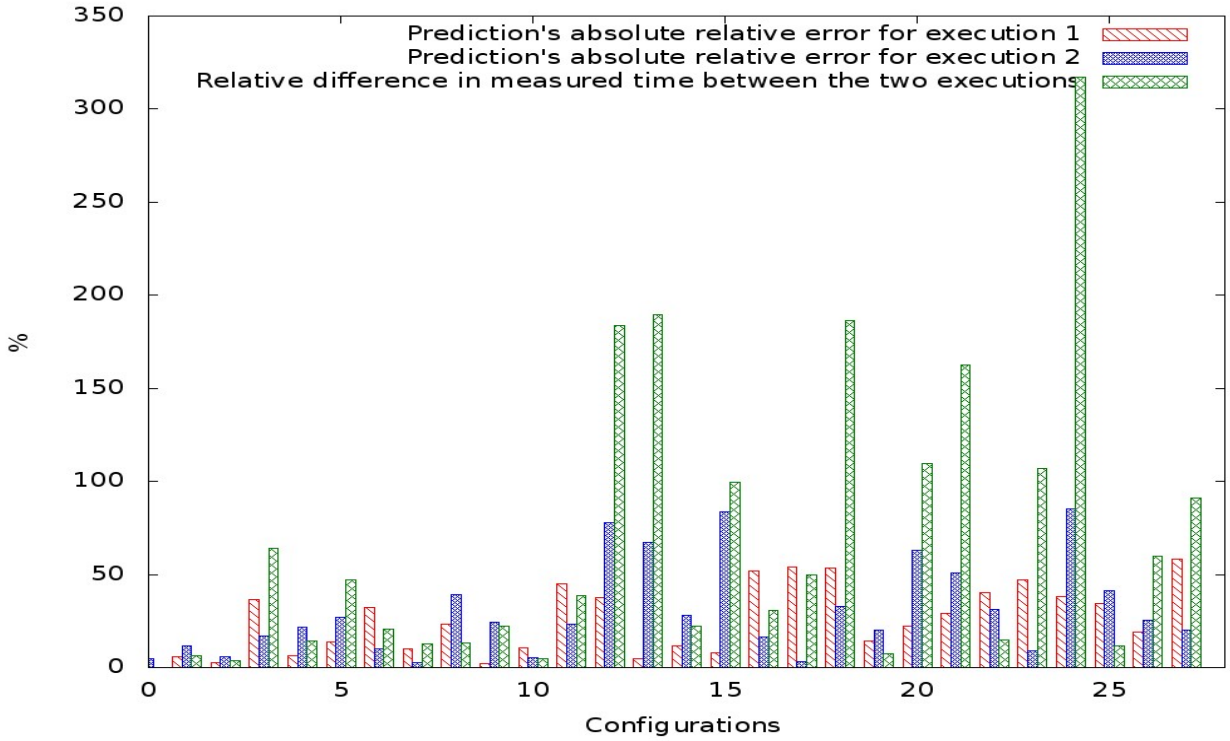


Figure 5.11: Performance variability percentages combined with relative errors of our prediction

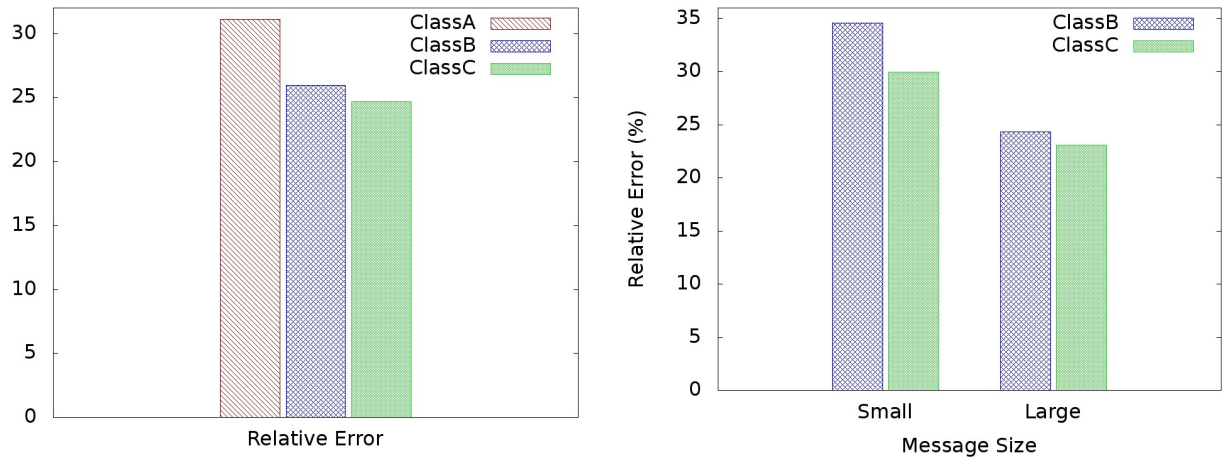
5.3 Vilje

5.3.1 Evaluation of prediction accuracy

In the following sections, we will demonstrate and comment the performance results from the devised for Vilje prediction scheme, and explain why our predictions, though satisfactory, were not as successful as those for Piz Daint.

5.3.1.1 Average absolute relative error

From Figure 5.12a, it is apparent that class B and C reduce the average absolute relative error significantly compared with class A. Between the two more detailed classes, the difference is rather small. A big difference in mean absolute relative error occurs for small sized messages (Figure 5.12b). This is expected, since small messages, in contrast with large messages, do not cause congestion problems to the injection links, but they can cause congestion at higher network levels, if many messages are accumulated at specific network components, like switches. Yet, large message samples are much more in number than small message ones, and thus improvements for small messages are not clearly reflected on the whole testing set results.



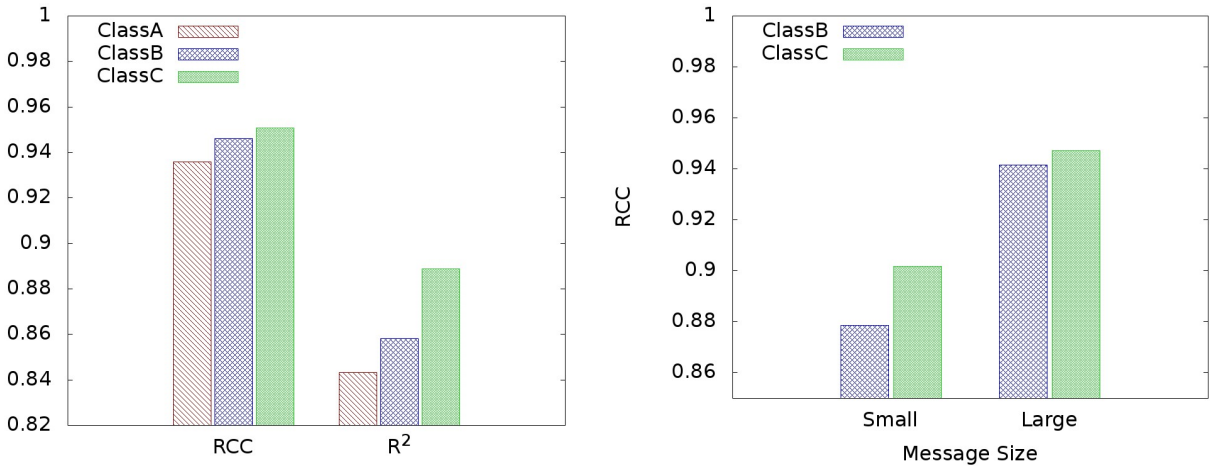
(a) Error for all classes

(b) Error for all the partitions of classes B and C

Figure 5.12: Average absolute relative error plots

5.3.1.2 *RCC* and R^2

Figure 5.13a reveals very important information regarding the gains of having advanced predictor classes. Without doubt, all the predictors perform very well concerning the *RCC* metric. The difference among the classes are noticeable but not so significant. *RCC* is primarily ameliorated for small sized messages (Figure 5.12b), as it was the case for Piz Daint as well, even though here the improvement is not so impressive. In regard to R^2, we can see that the score is improved substantially class by class and the major improvement occurs between class B and class C. It can be assumed that the more specific the metrics are, the more accurate the prediction in absolute values would be.



(a) Comparison of classes based on RCC and R^2 (b) RCC scores for all partitions of classes B and C

Figure 5.13: RCC and R^2 scores

It could be useful at this point to compare these results with Piz Daint. The scores for *RCC* are similar to the ones for Piz Daint. However, the R^2 scores are obviously worse for Vilje. Our metrics managed to achieve 0.978 R^2 in Piz Daint, while here they don't even reach 0.9. We can conclude that it is rather difficult to predict the absolute communication time, while we can easily distinguish and rank configurations and give a good estimation of the execution time with good relative errors.

We believe that the problem is the excessive interference and the congestion issues that come with it and lead to great performance variability. As opposed to Piz Daint, where only four

nodes were connected to each Aries router, in Vilje, we have eighteen nodes connected to each switch. Consequently, it is very likely that we would not occupy all the eighteen nodes connected to a switch and other jobs will reside in the remaining nodes causing extra contention for the switch. For instance, our measurement for the traffic of a given switch (value of metric ST) could be only the 50% of the actual traffic, leading us to wrong estimations of the delays that are caused by this switch. Furthermore, the routing protocol in Infiband is deterministic. It finds the shortest path and sticks to it without changing the route in case of congestion. Without sufficient congestion control and adaptive routing, unexpected and prolonged stalls from congestion will come up, something that was not the case for Piz Daint. On top of all these, the allocation that we were usually given was sparse. Such an allocation involved many network components and as a result there were increased chances of competing with external applications for some of them.

But why interference affected the R^2 score much more than the RCC? We can only guess that large groups of the samples in the testing set suffer from similar external noise, and our judgement regarding the comparison of the configurations is not drastically impeded. However, even if interference is uniform in many cases, it is extremely difficult to be quantified. Our benchmarking process was performed separately at a different moment, when different jobs were present nearby causing effects of different intensity compared with the testing set executions. Therefore, we are unable to properly capture the consequences in communication time of the interference in the 3D-Jacobi runs, and it is only natural, due to different levels of interference between the training and the testing set, to predict with a small offset from the measured time.

5.3.2 Error Distribution

Class A

In Figure 5.14, we can observe that most errors are concentrated around 0%. However, many predictions exceed 50% in both the positive and the negative side and the worst predictions come with some extreme errors that reach -78.51% and 157.34%.

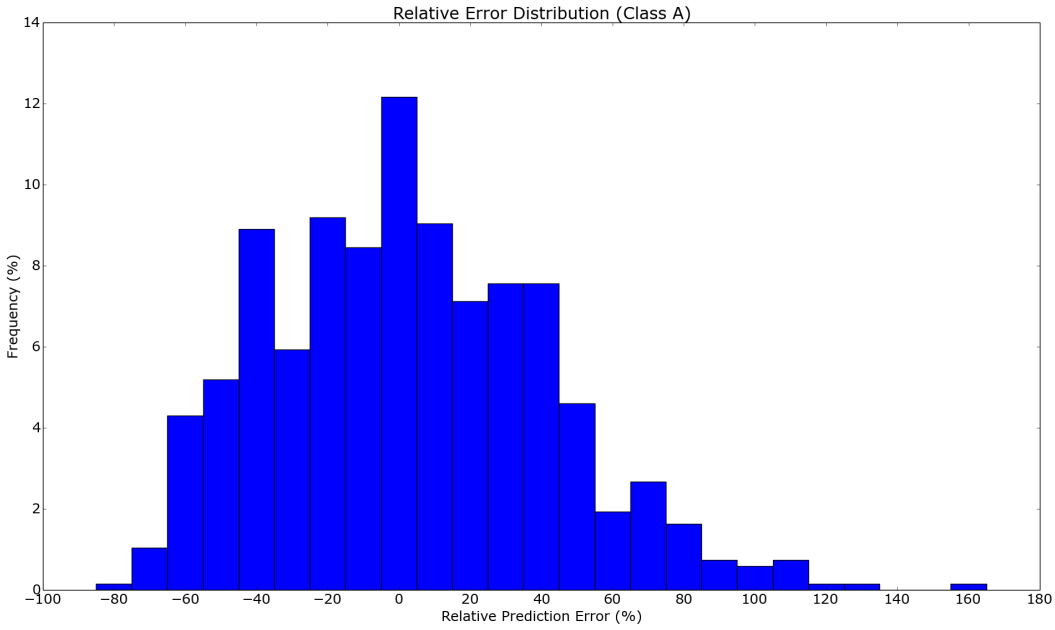


Figure 5.14: Class A error distribution

Class B

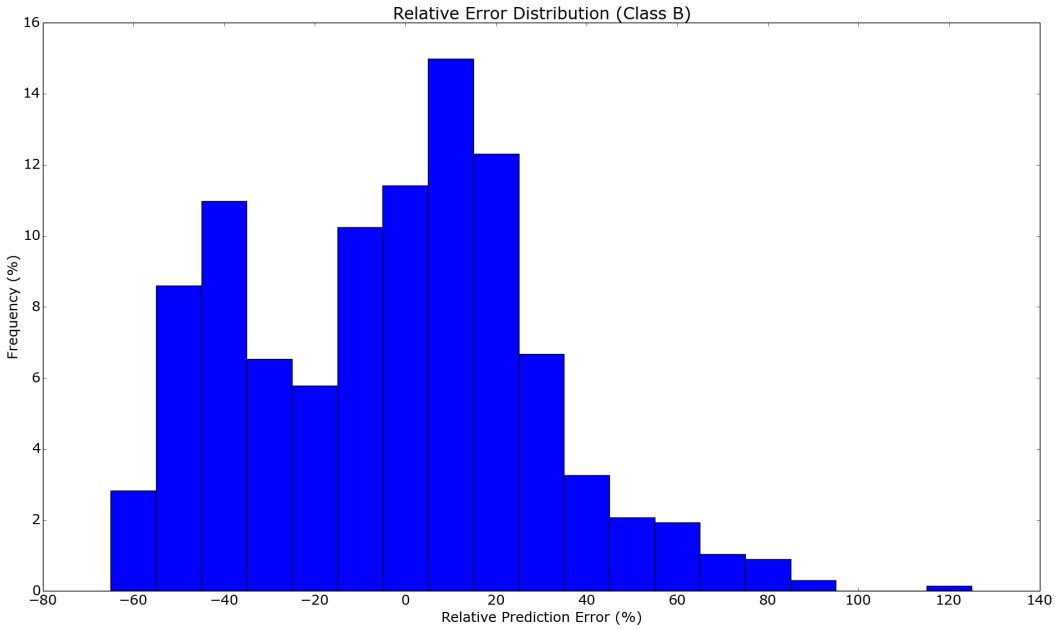


Figure 5.15: Class B error distribution

For class B, we see that, first of all, the worst negative errors are limited to -64.20%, from being -78.51% for class A. The highest errors from the positive side are also considerably reduced. In class A we had many points mispredicted with errors exceeding 100% and going up to 157%, while with class B predictor, we only have one point with error more than 100% and specifically 122.33%. Except for limiting the range, in class B the number of samples predicted with error within the $\pm 15\%$ is notably augmented to almost 40%.

Class C

Figure 5.16 shows that the prediction accuracy is a little bit improved compared with class B. The maximum positive error is reduced from 122.33% to 115.42%. The percent of predictions within $\pm 5\%$ is increased by 4%. Furthermore, the number of predictions with negative errors greater in absolute value than -45% is limited, and generally the negative errors are pushed closer to 0%. 42% of the samples' errors lie now between $\pm 15\%$.

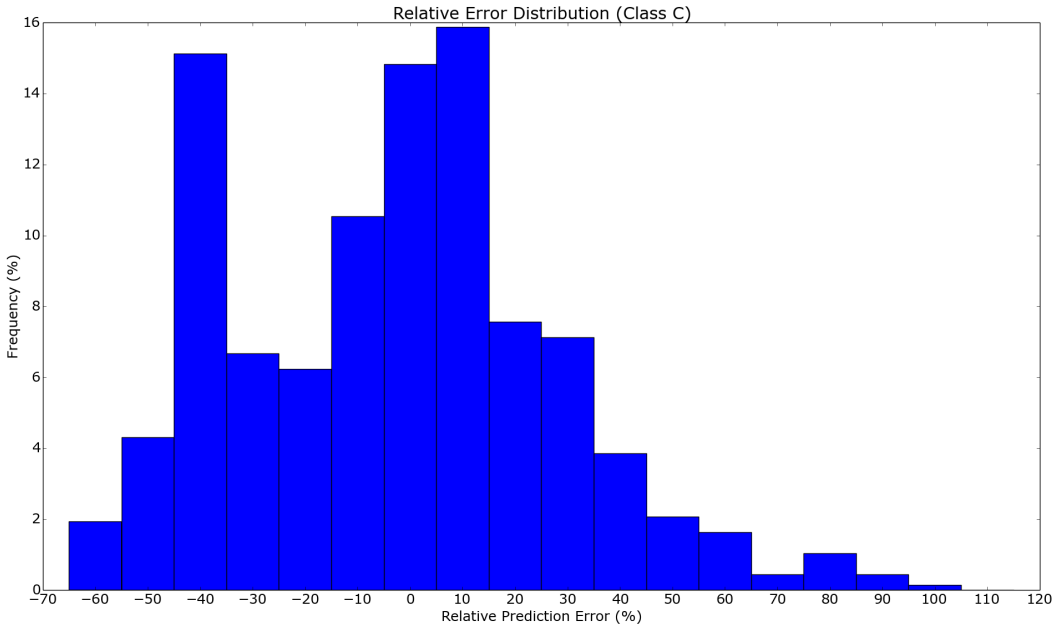


Figure 5.16: Class C error distribution

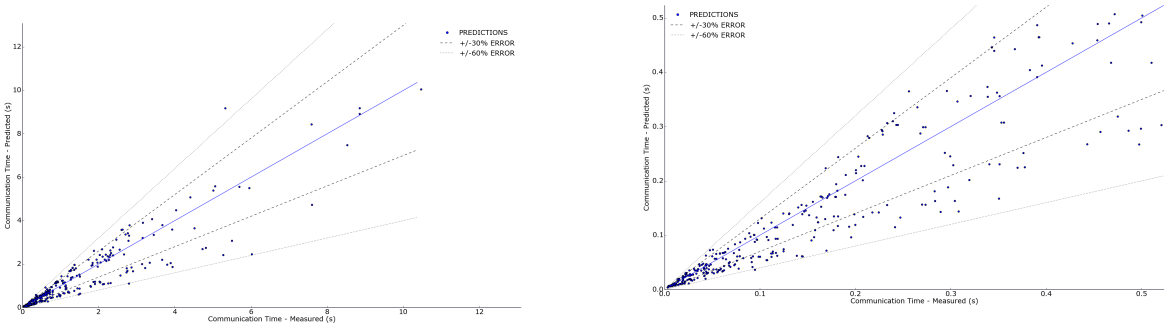
It is apparent from the error distribution of class B and C that our prediction scheme tends to produce more under-predictions than over-predictions and, apart from that, the under-

predictions exhibit much more excessive errors. Indicative of that are the high peaks that exist between -35% and -55% for class B and between -35% and -45% for class C. This flaw of our prediction scheme actually stems from the properties of the underlying system. As we previously stated, this platform is prone to congestion because of interference. This congestion refers either to link congestion or contention for switches from nodes connected to it or from messages using them as intermediate or final hops. All these stalls cannot be measured accurately and we are forced to make a rough estimate of them. Actually, from benchmarking we take an estimation of the execution time that is a little bit worse than the average case scenario; for each configuration, we drop the highest 25% (in execution time) so as to avoid extreme outliers and take the maximum value from the remaining measurements. Thus, we are unable to catch heavily performance-wise deteriorated data points of the testing set. Furthermore, our dataset from benchmarking is very large and our predictions are determined by an abundance of benchmarked configurations with similar traits, producing an additional moderating effect. Thus, it is reasonable to end up making a lot of underestimations.

After mentioning all these, we strongly believe that high levels of interference harm considerably our prediction scheme, while moderate levels are more expected and can be more accurately predicted. We evaluated experimentally this idea in section 5.2.3.

In the purpose of clarifying more the prediction results of the class C predictor, we will additionally present some error scatterplots for each partition.

Large messages : From Figure 5.17, it is clear that almost all the testing set with large sized messages was predicted with errors within the $\pm 60\%$ range. We can also observe that the overpredictions do not usually exhibit errors greater than 35%, while the underpredictions, that are more in number, tend to have in many occasions errors that lie between -30% and -60%.



(a) Whole large messages dataset

(b) Large messages configurations with small execution time

Figure 5.17: Class C error scatterplot for large messages

Small messages : For small sized messages, we can see from Figure 5.18, that many predictions have errors within the $\pm 30\%$ range. However, there exists an area, which consists of configurations with execution time less than 0.015 sec, where we have overpredictions with errors that exceed 60%. In this case, it appears that we have overestimated the delays owing to the processing and routing of packets from the switches or due to intranode contention. It is also possible that time measurements errors occurred given the extremely small execution times.

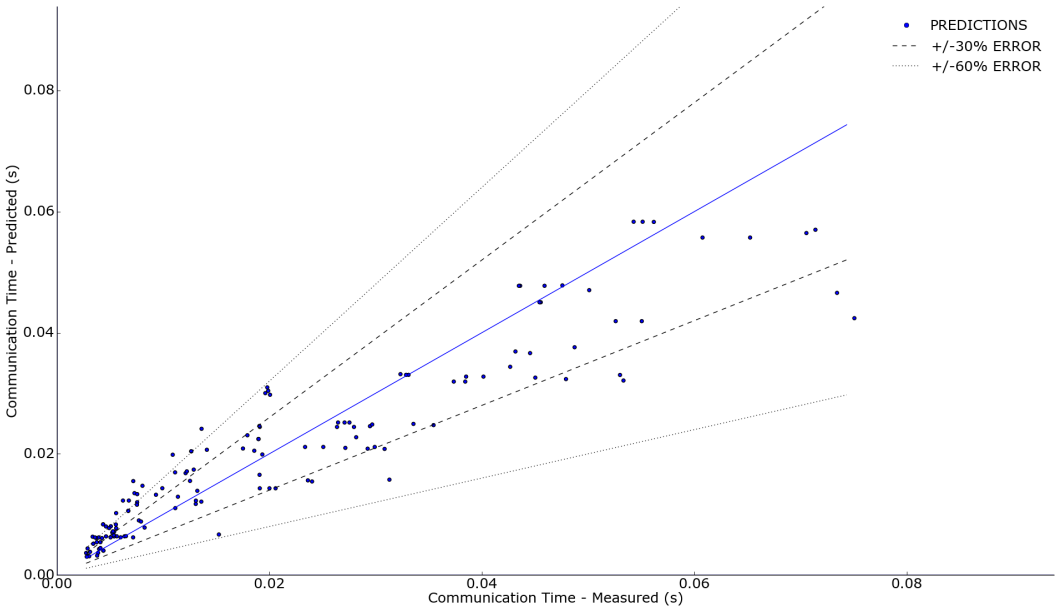


Figure 5.18: Class C error scatterplot for small messages

5.3.3 Performance variability

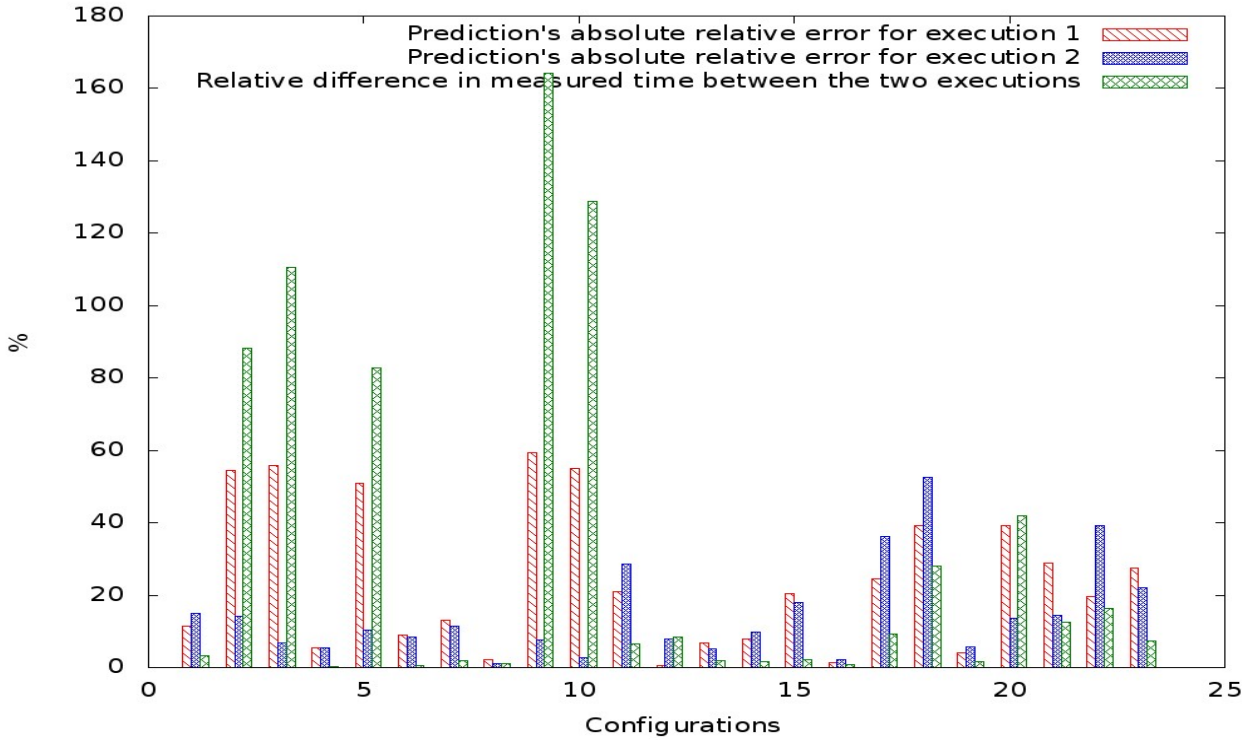


Figure 5.19: Performance variability percentages combined with relative errors of our prediction

The above figure presents differences between two executions of identical configurations. The differences in measured time are not so big and many as they were in Piz Daint. However, in this case, the configurations include both small and large messages and thus a comparison between the two machines is not so useful. Even though, some great variances exist for five configurations accompanied with high relative errors, and some less high ones with some high errors as well, we are unable to conclude that performance variability between different runs of the same configuration is the root of our predictor's shortcomings. We can presume instead that the noise from the operating system and nearby runs, if existent, is quite uniform in the aforementioned configurations. We will elaborate more on this last idea. The configurations that we examined in Figure 5.19 either apply to small allocations, like sixteen nodes, where nodes in two different runs are likely to be identically connected (e.g. all connected to a single switch), or to occasions when two executions were chronically close enough to each other so that the same system partition was available at both instances. In the first case, we do not expect high penalties due to interference because of the limited amount of network resources used, while

in the second case, we anticipate the same external applications running and producing similar interference effects and minimum variability between the two executions. The points, in Figure 5.19, that have indeed high variability could be subject to OS jitter or varying communication intensity and timing of the concurrent jobs.

After coming almost empty-handed from the previously examined configurations, in our attempt to justify some deficiencies of our prediction model in this machine, we compared the different runs of the 3D-Jacobi application. We tried to spot possible outliers and data points with big negative relative errors. Each of these runs could indicate different degrees of interference from jobs nearby. We noticed that the majority of the second execution's configurations displayed higher execution time compared with the corresponding configurations(same communication volume and allocation size) of the other executions. This increased time was not clearly justified by the topological characteristics of this run, as in most cases it had neither the sparsest nor the densest allocation and generally none of our metrics could explain this behaviour. Consequently, we concluded that during this run some communication intensive applications degraded severely the performance of the majority of the configurations. Such a run could be a reason for the plethora of under-predictions with high errors. Therefore, we tested our class C predictor without including the seemingly noisy second run. The results are presented in table 5.1. We have included for comparison the scores that we got when we used the entire testing set.

Table 5.1: Prediction accuracy depending on the used training set

Testing set	*Average relative error*	*Max relative error*	*Min relative error*	R^2 *score*	*RCC*	*Mean absolute error*
whole	24.69%	115.42%	-61.79%	0.8890	0.9507	0.1562
reduced	22.90%	109.68%	-57.13%	0.9340	0.9568	0.1181

It is apparent that all the metrics are improved with the exclusion of the second run from the testing set. The most significant enhancement concerns the prediction of absolute communication time values. We have a 24.4% reduction in mean absolute error and the R^2 score reaches at last a satisfactory level and reduces the gap from the achieved score in Piz Daint.

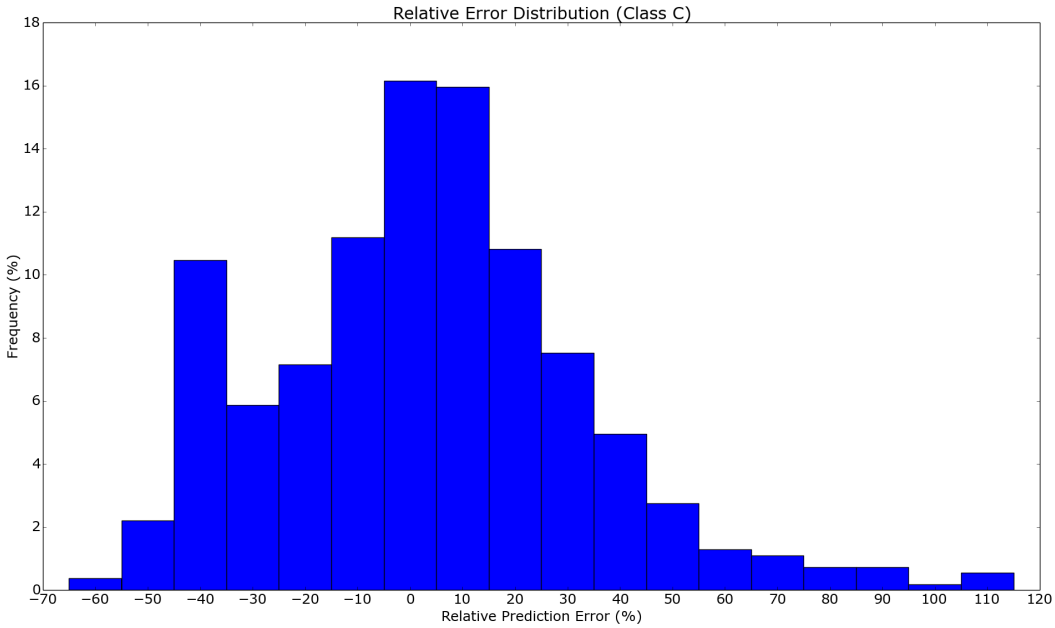


Figure 5.20: Class C error distribution for reduced testing set

In Figure 5.20, we can observe the error distribution of our class C predictor for the reduced testing set. We notice that the percentage of predictions with negative error lower than -35%, that was previously 22% (for the whole dataset), is now reduced to just 12.5%. In addition, the percentage of samples with error within the range $\pm 25\%$ is increased by 6%, reaching eventually 61% of the total used testing data points.

The aforementioned results indicate that our predictor is highly accurate when it encounters less deteriorated by interference application runs. Anyhow, even with aggressive interference, we managed to provide a pretty good estimation.

5.4 Use case

The utility of a prediction scheme is evaluated based on its ability to predict critical points in decision-making scenarios. Thus, we will examine such a scenario to further grade the quality of our predictor. We will test if our model finds the configuration with the minimum execution time, problem that it is very common for a user who wants to make the best choice and avoid configurations that do not scale well.

We speculate that a perfect predictor for computation time exists and we use our predictors for communication time. We tested for four different problem sizes of the 3D-Jacobi solver on both our machines. In addition, we tested the prediction of the configuration with the minimum communication time for the halo 4D communication kernel (no computation part) on our two platforms.

In table 5.2, we present the results. In the first column, we can observe the used system, the tested application and the problem size (for 3d Jacobi). In the next two columns, we present the observed and the predicted optimal configuration and in the last column we demonstrate, if the prediction failed, the relative error between the observed minimum execution time and the execution time of the predicted optimal configuration.

Test Case	Observed Conf. (NxPPN)	Predicted Conf. (NxPPN)	Result	Pred. vs Obs. relative error
Daint-3d-Jac-256^3	1024 x 4	512 x 8	False	11.4%
Daint-3d-Jac-512^3	1024 x 8	1024 x 8	True	-
Daint-3d-Jac-1024^3	1024 x 4	1024 x 8	False	8.5%
Daint-3d-Jac-2048^3	1024 x 4	1024 x 4	True	-
Vilje-3d-Jac-256^3	512 x 2	512 x 2	True	-
Vilje-3d-Jac-512^3	512 x 16	512 x 16	True	-
Vilje-3d-Jac-1024^3	512 x 16	512 x 16	True	-
Vilje-3d-Jac-2048^3	512 x 16	512 x 16	True	-
Daint-4d-Halo	1024 x 1	1024 x 1	True	-
Vilje-4d-Halo	256 x 1	256 x 1	True	-

Table 5.2: Minimum time configuration prediction

We successfully predict the optimal configuration (among 28 configurations for Daint and 30 for Vilje) for 8 out of 10 test cases. We miss out two test cases on Piz Daint. The first one

refers to the small problem size for which the message sizes for large allocations are very small and are, hence, very susceptible to performance variability (examined in previous section) due to external factors. The second test case involves a large problem size and all the sent messages are large sized messages. For large messages, we have benchmarked data points that include up to 128 nodes. Thus, predictions for 1024 node count are sometimes inevitably problematic. In particular, for this test case, the smallest message size is 16KB which is in fact the smallest large sized message and the one more affected by our limited benchmark data. Anyhow, for the mispredicted configurations the relative error between the execution time of the measured and predicted optimal configuration is 8.5% and 11.4%. These percentages indicate that even if the predicted configurations is not optimal, it could be considered suboptimal. We conclude, thus, that our prediction models are successful in identifying the minimum time configurations with a very low and usually zero error margin.

Chapter 6

Conclusion

In this thesis, we created a prediction scheme for the communication of MPI applications with iterative kernels and point-to-point communication patterns. We defined a set of descriptive features that stems from application's communication pattern, user defined allocation demands, process mapping and system architecture traits. Three different classes of predictors were devised going from the simple and generic to the more elaborate and accurate. These classes could be utilized by users, system schedulers and algorithm designers according to the needs of each entity in question. Our model was trained with a dataset from a benchmarking process that evaluated the relation between the features and communication time.

We constructed prediction methods for two different systems and tested them for two communication kernels with satisfactory results. Our prediction attempts revealed some interesting characteristics of the two used machines.

For Piz Daint, we achieved great absolute communication prediction. Large messages do not cause many congestion problems to the network due to adaptive routing and sufficient global bandwidth, and performance is primarily limited by the injection bandwidth. These delays are easily captured even by the class B predictor. The transfer of small messages is determined in a decisive degree by the number of hops and the traversed network levels. Moreover, communication costs for medium messages are influenced both by the injection rates and dilation. There are two areas which exhibited the worst results. The first one includes the smallest sized large

messages, the prediction for which is negatively affected by the reduced benchmark as far as node count is concerned. The second area refers to small messages, whose transfer time depends highly on dilation. Unluckily, different levels of interference caused different message routing (sometimes minimal and sometimes non-minimal) and led to great performance variability that impeded our prediction efforts.

For Vilje, congestion is the major reason for performance degradation for all message sizes. Static routing and insufficient bandwidth do not allow handling of the delays derived from highly contented network resources. Congestion phenomena could likely be caused by our application's traffic. Small messages could create congestion only on high network levels (e.g. intra-rack congestion), while communicating large messages could overwhelm injection links and stress the network as well. These phenomena are, unfortunately, highly aggravated by interfering jobs given Vilje's network architecture and the sparsity of given allocations. External jobs add additional traffic and congestion to the network forcing even small messages to stall sometimes at intermediate switches. We made a lot of under-predictions and did not predict absolute values so accurately because of the exclusion of some outliers from the benchmark and generally due to dissimilar levels of interference between the testing and the training set. However, by removing a noisy application run from the testing set, we achieved notable accuracy.

Our predictor was also evaluated for a decision-making scenario, that concerns users of large-scale systems, and was successful in identifying critical points.

We plan to extend our model for irregular communication patterns and test our methodology on additional applications and systems. In this way, we will explore our scheme's applicability and generalization ability.

Acknowledgements

This research was supported in part with computational resources at the Norwegian University of Science and Technology (NTNU) provided by NOTUR (http://www.notur.no) and in part by a grant from the Swiss National Supercomputing Center (CSCS) under project ID g83. We would like to thank the people at NTNU and CSCS for granting us access to Vilje and Piz Daint.

Bibliography

[1] Mohamed Sayeed, Hansang Bae, Yili Zheng, Brian Armstrong, Rudolf Eigenmann, and Faisal Saied. Measuring high-performance computing with real applications. *Computing in Science and Engineering*, 10(4):60–70, 2008.

[2] Abhinav Bhatelé, Lukasz Wesolowski, Eric Bohm, Edgar Solomonik, and Laxmikant V. Kalé. Understanding application performance via micro-benchmarks on three large supercomputers: Intrepid, ranger and jaguar. *Int. J. High Perform. Comput. Appl.*, 24(4):411–427, November 2010.

[3] Hormozd Gahvari, Allison H. Baker, Martin Schulz, Ulrike Meier Yang, Kirk E. Jordan, and William Gropp. Modeling the performance of an algebraic multigrid cycle on hpc platforms. In *Proceedings of the International Conference on Supercomputing*, ICS '11, pages 172–181, New York, NY, USA, 2011. ACM.

[4] Hormozd Gahvari, William Gropp, Kirk E. Jordan, Martin Schulz, and Ulrike Meier Yang. Algebraic multigrid on a dragonfly network: First experiences on a cray XC30. In *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation - 5th International Workshop, PMBS 2014, New Orleans, LA, USA, November 16, 2014. Revised Selected Papers*, pages 3–23, 2014.

[5] Pavan Balaji, Anthony Chan, Rajeev Thakur, William Gropp, and Ewing L. Lusk. Toward message passing for a million processes: characterizing MPI on a massive scale blue gene/p. *Computer Science - R&D*, 24(1-2):11–19, 2009.

[6] Abhinav Bhatele and Laxmikant V. Kalé. Quantifying network contention on large parallel machines. *Parallel Processing Letters*, 19(4):553–572, 2009.

[7] T. Hoefler and M. Snir. Generic Topology Mapping Strategies for Large-scale Parallel Architectures. In *Proceedings of the 2011 ACM International Conference on Supercomputing (ICS'11)*, pages 75–85. ACM, Jun. 2011.

[8] Bertrand Putigny, Benoit Ruelle, and Brice Goglin. Analysis of MPI shared-memory communication performance from a cache coherence perspective. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops, Phoenix, AZ, USA, May 19-23, 2014*, pages 1238–1247, 2014.

[9] S. Ramos and T. Hoefler. Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 97–108. ACM, Jun. 2013.

[10] Lei Chai, Albert Hartono, and Dhabaleswar K. Panda. Designing high performance and scalable mpi intra-node communication support for clusters. In *CLUSTER*. IEEE Computer Society, 2006.

[11] Abhinav Bhatele, Kathryn Mohror, Steven H. Langer, and Katherine E. Isaacs. There goes the neighborhood: Performance degradation due to nearby jobs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 41:1–41:12, New York, NY, USA, 2013. ACM.

[12] Matthew J. Koop, Miao Luo, and Dhabaleswar K. Panda. Reducing network contention with mixed workloads on modern multicore, clusters. In *CLUSTER*, pages 1–10. IEEE Computer Society, 2009.

[13] Alex D Breslow, Leo Porter, Ananta Tiwari, Michael Laurenzano, Laura Carrington, Dean M Tullsen, and Allan E Snavely. The case for colocation of hpc workloads. *Concurrency and Computation: Practice and Experience*, 2013.

[14] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. Characterizing the influence of system noise on large-scale applications by simulation. In *Proceedings of the*

2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[15] Fabrizio Petrini, Darren J. Kerbyson, and Scott Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of asci q. In *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, SC '03, pages 55–, New York, NY, USA, 2003. ACM.

[16] Roger W. Hockney. The communication challenge for mpp: Intel paragon and meiko cs-2. *Parallel Computing*, 20(3):389 – 398, 1994.

[17] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPOPP '93, pages 1–12, New York, NY, USA, 1993. ACM.

[18] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. LogGP: Incorporating long messages into the logP model - one step closer towards a realistic model for parallel computation. In *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '95, pages 95–105, New York, NY, USA, 1995. ACM.

[19] Fumihiko Ino, Noriyuki Fujimoto, and Kenichi Hagihara. LogGPS: A parallel computational model for synchronization analysis. In *Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*, PPoPP '01, pages 133–142, New York, NY, USA, 2001. ACM.

[20] Csaba Andras Moritz and Matthew I Frank. LoGPC: Modeling network contention in message-passing programs. In *ACM SIGMETRICS Performance Evaluation Review*, volume 26, pages 254–263. ACM, 1998.

[21] T. Hoefler, A. Lichei, and W. Rehm. Low-Overhead LogGP Parameter Assessment for Modern Interconnection Networks. In *Proceedings of the 21st IEEE International Parallel & Distributed Processing Symposium, PMEO'07 Workshop*. IEEE Computer Society, Mar. 2007.

[22] T. Hoefler, T. Schneider, and A. Lumsdaine. LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 597–604. ACM, Jun. 2010.

[23] Torsten Hoefler, William Gropp, Rajeev Thakur, and Jesper Larsson Träff. Toward performance models of mpi implementations for understanding application scaling issues. In *Proceedings of the 17th European MPI Users' Group Meeting Conference on Recent Advances in the Message Passing Interface*, EuroMPI'10, pages 21–30, Berlin, Heidelberg, 2010. Springer-Verlag.

[24] Nikhil Jain, Abhinav Bhatele, Michael P. Robson, Todd Gamblin, and Laxmikant V. Kale. Predicting application performance using supervised learning on communication features. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '13. IEEE Computer Society, November 2013 (to appear). LLNL-CONF-635857.

[25] Abhinav Bhatele, Andrew R. Titus, Jayaraman J. Thiagarajan, Nikhil Jain, Todd Gamblin, Peer-Timo Bremer, Martin Schulz, and Laxmikant V. Kale. Identifying the culprits behind network congestion. In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium (to appear)*, IPDPS '15. IEEE Computer Society, May 2015. LLNL-CONF-663150.

[26] Greg Faanes, Abdulla Bataineh, Duncan Roweth, Tom Court, Edwin Froese, Bob Alverson, Tim Johnson, Joe Kopnick, Mike Higgins, and James Reinhard. Cray cascade: A scalable hpc system based on a dragonfly network. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 103:1–103:9, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[27] Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.

[28] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001.

[29] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Mach. Learn.*, 63(1):3–42, April 2006.

[30] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.