ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Τεχνικές παραλληλοποίησης διεργασιών σε συστήματα cloud

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΘΕΟΔΩΡΟΥ Κ. ΝΙΚΟΛΑΚΟΠΟΥΛΟΥ

Επιβλέπουσα: Θεοδώρα Α. Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Νοέμβριος 2015

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Τεχνικές παραλληλοποίησης διεργασιών σε συστήματα cloud

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΘΕΟΔΩΡΟΥ Κ. ΝΙΚΟΛΑΚΟΠΟΥΛΟΥ

Επιβλέπουσα: Θεοδώρα Α. Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 9η Νοεμβρίου 2015.

……..	……..	……..
Θεοδώρα Βαρβαρίγου	Εμμανουήλ Βαρβαρίγος	Βασίλειος Λούμος
Καθηγήτρια Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2015

…….
Θεόδωρος Κ. Νικολακόπουλος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η ανάπτυξη της πληροφορικής και η εισβολή της σε κάθε σύγχρονη τεχνολογία έχει ως αποτέλεσμα την παραγωγή τεράστιου όγκου δεδομένων με συνεχώς αυξανόμενους ρυθμούς, αυτό που χαρακτηρίζουμε ως big data. Η ανάγκη για αποθήκευση και ανάλυση των big data οδήγησε στην ανάπτυξη νέων τεχνολογιών, με βασικότερη ίσως από αυτές το cloud computing. Τα συστήματα cloud μάς επιτρέπουν την κατανεμημένη αποθήκευση και επεξεργασία δεδομένων. Η προσέγγιση αυτή πλεονεκτεί λόγω του παραλληλισμού και της εντοπιότητας που μπορεί να προσφέρει, αλλά δημιουργεί ταυτόχρονα νέες προκλήσεις λόγω της κατανεμημένης φύσης της. Ο μεγάλος αριθμός και η ποικιλία των χαρακτηριστικών των διαθέσιμων υπηρεσιών cloud καθιστά την επιλογή της κατάλληλης υπηρεσίας εξαιρετικά δύσκολη για τον χρήστη. Υπάρχει επομένως ανάγκη για δημιουργία αυτοματοποιημένων marketplace για την αναζήτηση και σύγκριση των υπηρεσιών αυτών.

Σκοπός της παρούσας διπλωματικής είναι η μελέτη του marketplace για υπηρεσίες cloud 4CaaSt και η εύρεση τρόπων βελτίωσης της αποδοτικότητάς του με τεχνικές παραλληλοποίησης. Η επίλυση του προβλήματος της επιλογής κατάλληλων υπηρεσιών cloud στο 4CaaSt γίνεται σε τέσσερα στάδια, καθένα εκ των οποίων ξεκινά να εκτελείται αφού ολοκληρωθεί το προηγούμενο. Στην εργασία αυτή προτείνονται δύο τεχνικές για την μετάδοση των δεδομένων από κάθε στάδιο στο επόμενο κατά τη στιγμή της παραγωγής τους και πριν την ολοκλήρωση του σταδίου. Οι τεχνικές αυτές στηρίζονται στη χρήση ουρών μηνυμάτων. Η πρώτη προτείνει τη χρήση μίας ουράς, στην οποία θα προωθούνται τα δεδομένα από κάθε στάδιο, και αυτή θα αναλαμβάνει να τα στείλει στο επόμενο. Η δεύτερη προτείνει τη χρήση ξεχωριστής ουράς για κάθε ζεύγος διαδοχικών σταδίων. Αυτές οι τεχνικές θα επιτρέψουν στα τέσσερα στάδια να εκτελούνται σε παράλληλα σε μεγάλο βαθμό.

Για να συγκριθούν ως προς την αποδοτικότητά τους οι τεχνικές αυτές, τόσο μεταξύ τους όσο και με την αρχική ιδέα της σειριακής υλοποίησης, αναπτύχθηκε ένα απλοποιημένο μοντέλο του 4CaaSt, με έμφαση στα τέσσερα στάδια της επίλυσης του προβλήματος αναζήτησης. Έγιναν μετρήσεις για διαφορετικά πλήθη ανταλλασσόμενων μηνυμάτων μεταξύ των σταδίων, καθώς και για διαφορετικές διάρκειες κάθε σταδίου. Από τη συγκριτική μελέτη των αποτελεσμάτων διαπιστώθηκε ότι και οι δύο τεχνικές βελτιώνουν την αποδοτικότητα του marketplace, με την τεχνική των ξεχωριστών ουρών να υπερτερεί.

Λέξεις-κλειδιά

Big data, message queues, cloud computing, παράλληλη επεξεργασία, marketplace, ZeroMQ, 4CaaSt, BaseX.

Abstract

The rapid evolution of information technology and its integration to modern technologies results to the production of immense volumes of data in an exponentially increasing rate and lead to the introduction of the term big data. The new needs for storing and processing big data inspires the development of new techniques and technologies, the most important among them being cloud computing. Cloud computing systems enable distributed data storage and processing. This approach offers the advantages of parallel processing and locality, but also creates new challenges due to its distributed nature. The big number of the available cloud services, together with the wide variety in their characteristics, make the selection of the most appropriate service difficult for the user. Automated marketplaces for cloud services are to be developed for that reason.

In this thesis we study the proposed 4CaaSt marketplace for cloud services and we seek ways of improving its performance using parallelization techniques. 4CaaSt resolves the problem of proposing the most appropriate cloud services package in for steps, each of them has to wait for the previous one to end before starting. Using message queues, we propose two techniques for data transferring from each step to the next, as soon as data are created and well before the whole step finishes. The first technique uses a single message queue, to which data are sent once created, and then forwarded to the next step. The second uses separate queues for data transferring between each pair of consecutive steps.

In order to compare the efficiency of these techniques with one another, as well as against the initial implementation of serial processing, a simplified model of the 4CaaSt marketplace has been created, emphasizing upon the aforementioned four steps. Measurements have been taken for different numbers of send messages as well as for different workloads on each step. Studying the measurement results we conclude that both these techniques improve the efficiency of the marketplace; the separate queues model is even more efficient than the single queue one.

Λέξεις-κλειδιά

Big data, message queues, cloud computing, parallel processing, marketplace, ZeroMQ, 4CaaSt, BaseX.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά την επιβλέπουσα καθηγήτρια κ. Θεοδώρα Βαρβαρίγου για την καθοδήγησή της στην εκπόνηση της παρούσας διπλωματικής εργασίας. Θα ήθελα επίσης να ευχαριστήσω τα μέλη του Εργαστηρίου Κατανεμημένης Γνώσης και Πολυμεσικών Συστημάτων Ε.Μ.Π. για την εξαιρετική συνεργασία που είχαμε και ιδιαιτέρως τον υποψήφιο διδάκτορα κ. Βρεττό Μουλό για την πολύτιμη βοήθειά του και καθοδήγησή του σε όλα τα στάδια της διπλωματικής αυτής. Ακόμη, θα ήθελα ευχαριστήσω και να εκφράσω την ευγνωμοσύνη μου στους γονείς μου, Κωνσταντίνο και Αλεξάνδρα, για την αμέριστη στήριξή τους όλα αυτά τα χρόνια.

Πίνακας περιεχομένων

Πίνακας σχημάτων

Πίνακας πινάκων

1 Εισαγωγή

Η ραγδαία αύξηση του ρυθμού παραγωγής δεδομένων μάς κάνει πλέον να μιλάμε για την εποχή των big data. Οι νέες τεχνολογίες όχι μόνο δημιουργούν πολύ περισσότερα δεδομένα, αλλά μας επιτρέπουν να τα αποθηκεύουμε ευκολότερα και οικονομικότερα, καθώς και να τα επεξεργαζόμαστε ταχύτερα. Ένας σχετικά σύγχρονος κλάδος της πληροφορικής που γνωρίζει ραγδαία ανάπτυξη αυτήν την περίοδο, λόγω και της υψηλής «συμβατότητάς» του με τα bid data, είναι το cloud computing. Υπάρχουν σήμερα πολυάριθμες υπηρεσίες που προσφέρονται σε περιβάλλον cloud, με πολλά διαφορετικά χαρακτηριστικά και για πολλούς διαφορετικούς τομείς. Αυτό καθιστά το πρόβλημα της επιλογής της κατάλληλης υπηρεσίας για το εκάστοτε πρόβλημα εξαιρετικά δύσκολη για έναν και μόνο άνθρωπο. Ως εκ τούτου, είναι απαραίτητο να δημιουργηθούν marketplace τα οποία να μπορούν να επιλύουν αυτοματοποιημένα αυτό το πρόβλημα για το χρήστη.

Στην παρούσα διπλωματική μελετάμε ένα τέτοιο marketplace, το 4CaaSt, το οποίο προτάθηκε για να παρέχει εξελιγμένους τρόπους επίλυσης του προβλήματος αυτού. Στόχος είναι η πρόταση τεχνικών βελτίωσης της αποδοτικότητάς του, παραλληλίζοντας την εκτέλεση των σταδίων επίλυσης του προβλήματος. Για το σκοπό αυτό χρησιμοποιούμε ουρές μηνυμάτων (message queues), οι οποίες αναλαμβάνουν την μετάδοση των δεδομένων από κάθε στάδιο επίλυσης του προβλήματος στο επόμενο, καθώς και την προσωρινή αποθήκευσή τους, όταν το επόμενο στάδιο είναι απασχολημένο.

Το κείμενο της παρούσας διπλωματικής οργανώνεται ως εξής: στο πρώτο κεφάλαιο γίνεται μια αναφορά στον κλάδο των big data, τα χαρακτηριστικά τους και τις σχετικές με αυτά τεχνολογίες. Στο δεύτερο κεφάλαιο εστιάζουμε στο cloud computing και στα ιδιαίτερα χαρακτηριστικά του. Στο τρίτο κεφάλαιο ασχολούμαστε με τις ουρές μηνυμάτων (message queue) και μελετάμε διάφορες τεχνικές χρήσης τους, με έμφαση στην ουρά ZeroMQ. Στο τέταρτο κεφάλαιο αναλύουμε τα χαρακτηριστικά που πρέπει να έχει ένα marketplace για υπηρεσίες cloud και μελετάμε το 4CaaSt. Στο πέμπτο κεφάλαιο περιγράφεται η απλοποιημένη εκδοχή marketplace που υλοποιήθηκε στα πλαίσια αυτής της διπλωματικής. Στο έκτο κεφάλαιο εκθέτονται τα αποτελέσματα των μετρήσεων και παρουσιάζονται τα αποτελέσματα της μελέτης.

2 Big Data

Η σύγχρονη τεχνολογία έχει παρεισφρήσει σχεδόν σε κάθε πτυχή της ανθρώπινης δραστηριότητας, από την ιδιωτική ως τη δημόσια, από την προσωπική ως την κοινωνική, από τη διασκέδαση ως την εργασία. Και οι σύγχρονες τεχνολογίες λειτουργούν συνήθως χρησιμοποιώντας κάποιο είδος υπολογιστικού συστήματος· παράγουν λοιπόν δεδομένα.

Καθώς οι ηλεκτρονικοί υπολογιστές και τα μέσα αποθήκευσης δεδομένων εξελίσσονται, γίνονται φθηνότερα και συνεπώς όλο και πιο προσβάσιμα τόσο σε εταιρείες όσο και στο κοινό. Αυτό έχει ως αποτέλεσμα ο ρυθμός με τον οποίο παράγονται δεδομένα να αυξάνεται εκθετικά με την πάροδο του χρόνου. Ως ένα απλοϊκό, πλην όμως εντυπωσιακό, παράδειγμα αξίζει να σημειωθεί ότι κατά το έτος 2011, κατά τη διάρκεια ενός σαρανταοκταώρου, παράγονταν παγκοσμίως δεδομένα όγκου 1.8ZB (1,800,000,000,000,000,000,000 ή $1.8 \cdot 10^{21}$ byte). Ο όγκος αυτός αντιστοιχεί περίπου στο σύνολο τον δεδομένων που παρήγαγε ολόκληρη η ανθρωπότητα από την εμφάνισή της έως το 2003! [1] Ως εκ τούτου, η σύγχρονη πληροφορική καλείται να αντιμετωπίσει την πρόκληση της επεξεργασίας δεδομένων των οποίων ο όγκος είναι σχεδόν ασύλληπτος για τον ανθρώπινο εγκέφαλο και ο ρυθμός παραγωγής συνεχώς αυξάνεται· καλείται να αντιμετωπίσει τα λεγόμενα «Big Data».

Ανάμεσα σε δεδομένα μεγάλου όγκου βρίσκεται συχνά «κρυμμένη» πολύτιμη πληροφορία. Πληροφορία που δεν θα μπορούσε να εξαχθεί ούτε από έναν και μόνο συμβατικό υπολογιστή, αλλά ούτε και με χρήση των συμβατικών τεχνικών που παραδοσιακά χρησιμοποιούνται στην επεξεργασία δεδομένων –τουλάχιστον σε εύλογο χρονικό διάστημα. Επίσης, σε πολλές περιπτώσεις, η πληροφορία αυτή δεν θα μπορούσε να εξαχθεί παρά μόνο με την επεξεργασία τεράστιου όγκου δεδομένων, συχνά διαφορετικής μορφής, παλαιότητας και πηγής. Αυτό είναι προσδίδει ενδιαφέρον στις τεχνολογίες των big data και προσφέρει στη σύγχρονη πληροφορική νέες προκλήσεις.

2.1 Ορισμός

Παρ’ ότι τα big data τυγχάνουν ευρέως ενδιαφέροντος και εντατικής έρευνας κατά την τελευταία δεκαετία, δεν υπάρχει προς το παρόν κάποιος καθολικά αποδεκτός ορισμός.

Κάποιοι από τους προτεινόμενους ορισμούς εστιάζουν στα χαρακτηριστικά των big data, όπως ο ακόλουθος: «Big data: ένας τεράστιος όγκος από δομημένα και μη δεδομένα, τόσο μεγάλος που είναι δύσκολο να τα επεξεργαστούμε χρησιμοποιώντας παραδοσιακές βάσεις δεδομένων και τεχνολογίες λογισμικού.» Υπάρχουν αρκετοί ακόμα ορισμοί αυτού του είδους, οι οποίοι κυρίως αναφέρονται στα λεγόμενα «5V» των big data, οι οποίοι δεν παρατίθενται εδώ, καθώς θα ασχοληθούμε αναλυτικά με τα χαρακτηριστικά των big data στη συνέχεια.

Άλλοι ορισμοί εστιάζουν στο πώς οι συμβατικές τεχνικές και τεχνολογίες αδυνατούν να τα επεξεργαστούν τα big data. Η Apache ορίζει τα big data ως «σύνολα δεδομένων τα οποία οι γενικοί υπολογιστές δεν θα μπορούσαν να συλλέξουν, διαχειριστούν και επεξεργαστούν σε αποδεκτό χρονικό διάστημα». Η IDC (International Data Corporation – Διεθνής Εταιρεία Δεδομένων) αναφέρεται στις «τεχνολογίες big data [οι οποίες] περιγράφουν μία νέα γενιά τεχνολογιών και αρχιτεκτονικών, σχεδιασμένων να εξάγουν οικονομικά αξία από πολύ μεγάλους

όγκους ποικίλων δεδομένων, κάνοντας δυνατή την υψηλής ταχύτητας σύλληψη, ανακάλυψη ή/και ανάλυσή τους». [2] Τέλος, το NIST (National Institute of Standards and Technology – Εθνικό Ινστιτούτο Προτύπων και Τεχνολογίας των Η.Π.Α.), ορίζει τα big data ως εξής: «Big data καλούνται τα δεδομένα των οποίων ο όγκος, η ταχύτητα σύλληψης ή η αναπαράσταση περιορίζει την δυνατότητα χρήσης παραδοσιακών σχεσιακών μεθόδων προς διεξαγωγή αποτελεσματικής ανάλυσης, ή τα δεδομένα τα οποία μπορούν να επεξεργασθούν αποτελεσματικά χρησιμοποιώντας οριζόντιες τεχνολογίες zoom.»

2.2 Χαρακτηριστικά

Τα χαρακτηριστικά των big data μπορούν να αναλυθούν με δύο προσεγγίσεις: τα προαναφερθέντα 5V (ή 3V ή 4V κατ' άλλους) και το θεώρημα HACE. Όπως θα γίνει κατανοητό στη συνέχεια, οι δύο αυτές προσεγγίσεις έχουν πολλά κοινά και σε καμία περίπτωση δεν αντιβαίνει η μία στην άλλη.

2.2.1 Τα 5V

Ως 5V σωρευτικά αναφέρονται ο όγκος (volume), η ταχύτητα (velocity), η ποικιλία (variety), η αξία (value) και η αξιοπιστία (veracity) των big data, από το πρώτο γράμμα των όρων στην αγγλική γλώσσα. Ορισμένοι κάνουν λόγο για 3V, δηλαδή όγκο, ταχύτητα, και ποικιλία, ενώ άλλοι για 4V, προσθέτοντας στα τρία προηγούμενα την αξία.

2.2.1.1 Ο όγκος (volume)

Ο (μεγάλος) όγκος είναι το κυρίαρχο χαρακτηριστικό των big data. Είτε πρόκειται για terabyte, είτε για petabyte ή για exabyte, ο τεράστιος όγκος δεδομένων παίζει πρωτεύοντα ρόλο στις δυσκολίες που αντιμετωπίζονται κατά την επεξεργασία, μεταφορά και αποθήκευσή τους.

2.2.1.2 Η ποικιλία (variety)

Η ποικιλία χαρακτηρίζει τα big data σε πολλαπλά επίπεδα. Κατ' αρχάς, στις περισσότερες περιπτώσεις έχουμε πολλά διαφορετικά είδη δεδομένων. Κείμενο, φυσική γλώσσα, οπτικό υλικό από κάμερες διαχείρισης κυκλοφορίας, μετρήσεις από επιστημονικές παρατηρήσεις, δεδομένα τραπεζικής αποτελούν ένα ελάχιστο μερίδιο των διαφορετικών ειδών big data. Φυσικά, κάθε είδος δεδομένων αποθηκεύεται σε διαφορετικούς τύπους και δομές, οδηγώντας σε ακόμα μεγαλύτερη ποικιλία, καθώς πλέον έχουμε να κάνουμε με δομημένα, ημιδομημένα ή μη δομημένα δεδομένα. Ακόμη όμως και στην περίπτωση που τα big data που μας ενδιαφέρουν είναι όλα του ιδίου τύπου, μπορεί να προέρχονται από πολλές διαφορετικές πηγές, καθεμία των οποίον είναι πιθανόν να παράγει δεδομένα διαφορετικής δομής. Τέλος, το γεγονός ότι τα big data μπορεί να έχουν συλλεχθεί σε βάθος χρόνου έρχεται να προσθέσει ακόμα έναν παράγοντα πολυπλοκότητας, καθώς οι χρησιμοποιούμενες δομές από κάθε πηγή μπορεί να τροποποιούνται καθώς εξελίσσεται το αντίστοιχο λογισμικό.

2.2.1.3 Ταχύτητα (velocity)

Όταν αναφερόμαστε στην ταχύτητα ως χαρακτηριστικό των big data, εννοούμε δύο διαφορετικά πράγματα. Πρώτον, τον πολύ υψηλό ρυθμό παραγωγής τους. Δεύτερον, την υψηλή ταχύτητα που πρέπει να έχουν τα μέσα μεταφοράς τους, ούτως ώστε να μεταφέρονται σε αποδεκτά για την εκάστοτε χρήση χρονικά πλαίσια. Παραδείγματος χάριν, το grid του LHC (Large Hadron Collider – Μεγάλου Επιταχυντήρα Αδρονίων) του CERN (Centre Européen pour la Recherche Nucléaire – Ευρωπαϊκού Κέντρου Πυρηνικών Ερευνών), πρέπει να επεξεργάζεται δεδομένα όγκου 1PB ανά εικοσιτετράωρο. Αυτό σημαίνει ότι μεταξύ των εξυπηρετητών του διακινούνται μέχρι και 10GB ανά δευτερόλεπτο. [3]

2.2.1.4 Η αξία (value)

Αυτό που έλκει το ενδιαφέρον στην ανάλυση των big data είναι η αξία της πληροφορίας που «κρύβουν». Στο παρελθόν, οι περιορισμοί στην αποθήκευση των δεδομένων οδηγούσε στο να διατηρούνται μόνο τα δεδομένα υψηλής «πυκνότητας» ως προς την αξία τους. Σήμερα, οι αποθηκευτικοί αλλά και υπολογιστικοί πόροι που έχουμε στη διάθεσή μας μας επιτρέπουν να διατηρούμε και να αναλύουμε δεδομένα χαμηλής «πυκνότητας» ως προς την αξία. Αυτό μας δίνει τη δυνατότητα να εξάγουμε χρήσιμη πληροφορία που δεν θα μπορούσε να εξαχθεί με άλλο τρόπο.

Η αξιοποίηση της πληροφορίας που μπορεί να εξαχθεί από big data σχετικά με την υγεία για παράδειγμα, υπολογίζεται ότι μπορεί να επιφέρει μείωση κατά 8% στις ετήσιες δαπάνες του συστήματος υγειονομικής και φαρμακευτικής περίθαλψης των Η.Π.Α., ποσοστό που αντιστοιχεί σε 300 δισ. δολάρια ετησίως.

2.2.1.5 Η αξιοπιστία (veracity)

Τα big data περιέχουν δεδομένα τα οποία μπορεί να είναι αναξιόπιστα και γι' αυτό είναι σημαντική η ανάπτυξη τεχνικών για την αξιολόγηση και το φιλτράρισμα των δεδομένων. Η αναξιοπιστία αυτή μπορεί να οφείλεται σε διάφορους λόγους. Αφ' ενός, όπως προαναφέρθηκε, τα big data προέρχονται συνήθως από πλήθος διαφορετικών πηγών, πολλές φορές κάποιες από τις οποίες είναι αναξιόπιστες. Σε άλλες περιπτώσεις περιλαμβάνουν δεδομένα που προέρχονται από χρήστες του διαδικτύου, των οποίων η αξιοπιστία δεν είναι δυνατόν να ελεγχθεί. Άλλος ένας λόγος που μπορεί να οδηγήσει στη χρήση μη αξιόπιστων δεδομένων είναι τα μέσα και τα πρωτόκολλα μέσω των οποίων μετακινούνται τα δεδομένα, καθώς και το ποιος μεσολαβεί σε αυτή τη διαδικασία.

2.2.2 Το θεώρημα HACE

Με βάση το θεώρημα HACE, «τα big data ξεκινούν με ετερογενείς (**H**eterogeneous) και αυτόνομες (**A**utonomous) πηγές μεγάλου όγκου δεδομένων που δεν ελέγχονται κεντρικά, αλλά κατανεμημένα, και προσπαθούν να εξερευνήσουν πολύπλοκες (**C**omplex) και εξελισσόμενες (**E**volving) σχέσεις μεταξύ των δεδομένων αυτών». [4]

Παρατηρούμε ότι αυτή η –πιο πρόσφατη– θεώρηση καλύπτει τα δύο από τα 5V που αναλύθηκαν προηγουμένως, και συγκεκριμένα τον (μεγάλο) όγκο και την ποικιλία (ως ετερογένεια και

αυτονομία των πηγών), ενώ αναφέρεται ακροθιγώς και στην ταχύτητα (δεδομένου ότι μιλά για πηγές μεγάλου όγκου δεδομένων), αλλά και στην αξιοπιστία (δεδομένης της απουσίας κεντρικού ελέγχου των πηγών).

Επιπροσθέτως, όμως, εισάγει ρητώς και την έννοια της υψηλής πολυπλοκότητας των σχέσεων που αναζητούνται, καθώς και της ιδιότητάς τους να εξελίσσονται με την πάροδο του χρόνου. Παραδείγματος χάριν, στον κλάδο του CBM (Condition-based maintenance – Συντήρηση υπό συνθήκες), η ανάλυση δεδομένων που αφορούσαν στις συνθήκες που προηγήθηκαν βλαβών κατά το παρελθόν είναι σε θέση να μας δώσει πολύτιμες πληροφορίες για την αποφυγή ιδίων φαινομένων πριν καν αυτά εκδηλωθούν ή έστω πριν οδηγήσουν σε κρίσιμες καταστάσεις. Μια τέτοια ανάλυση απαιτεί την εξαγωγή πολύπλοκων σχέσεων που συνήθως δεν μπορούν να προβλεφθούν θεωρητικά, μεταξύ πολλών διαφορετικών μεταβλητών, κάτι που απαιτεί τη συσχετισμένη ανάλυση των δεδομένων τους.

2.3 Η προέλευση και οι εφαρμογές των big data

Ο κλάδος των big data ασχολείται με τα δεδομένα που παράγονται από κάθε πτυχή της ανθρώπινης τεχνολογικής και μη δραστηριότητας και πληρούν τα προαναφερθέντα χαρακτηριστικά. Ενδεικτικά εδώ θα αναφερθούν ορισμένοι μόνο από τους σημαντικότερους κλάδους που παράγουν big data.

Στις θετικές επιστήμες, σχεδιάζεται και εκτελείται πλήθος πειραμάτων και παρατηρήσεων που παράγουν τεράστιο όγκο δεδομένων. Ίσως το πιο ευρέως γνωστό παράδειγμα είναι από τον κλάδο της Βιολογίας και αφορά στην ανάλυση του ανθρώπινου γονιδιώματος (Human Genome Project), όπου ογκώδη δεδομένα προερχόμενα από την αλληλούχηση των βάσεων του ανθρώπινου DNA σε εργαστήρια ανά τον κόσμο έπρεπε να αναλυθούν και να συσχετισθούν. Αλλά και στις υπόλοιπες θετικές επιστήμες υπάρχουν πλήθος εφαρμογών που παράγουν και χρησιμοποιούν big data, όπως οι επιταχυντήρες στον κλάδο της Πυρηνικής Φυσικής, τηλεσκόπια στον κλάδο της Αστρονομίας, μετεωρολογικοί σταθμοί στον κλάδο της Μετεωρολογίας κ.ο.κ.

Στον κλάδο της ιατροφαρμακευτικής περίθαλψης δεδομένα από κλινικές διαδικασίες κατά την περίθαλψη ασθενών, δεδομένα από συμπτώματα και διαγνώσεις που μπορούν στη συνέχεια συσχετιζόμενα να διευρύνουν τις γνώσεις μας και να αυτοματοποιήσουν τη διαφορική διάγνωση, καθώς και δεδομένα από την εμφάνιση συγκεκριμένων ασθενειών, που έχουν μετέπειτα επιδημιολογική αξία.

Τα Οικονομικά και η Τραπεζική μάς δίνουν επίσης τεράστιους όγκους δεδομένων, όπως οι κινήσεις των χρηματιστηρίων διεθνώς, ισολογισμοί, τραπεζικές κινήσεις, κάθε είδος αγοραπωλησίας για την οποία τηρούνται ηλεκτρονικά αρχεία.

Μία από τις μεγαλύτερες πηγές προέλευσης big data είναι τα κοινωνικά δίκτυα. Περιέχουν δεδομένα σε πολλές μορφές, όπως ήχο, εικόνα, βίντεο, κείμενο εμπλουτισμένο με εικονίδια, σχέσεις μεταξύ ανθρώπων και τοποθεσιών όπως τις δηλώνουν οι χρήστες ή προκύπτουν αυτοματοποιημένα από το ίδιο το δίκτυο, γεωγραφικές τοποθεσίες

Άλλες πηγές big data είναι αρχεία τηλεφωνικών κλήσεων, δεδομένα από δίκτυα αισθητήρων, Internet of Things (IoT), δεδομένα από δορυφόρους, στρατιωτική παρακολούθηση, κυβερνητικές πληροφορίες αλλά και δεδομένα που αφορούν φυσικές καταστροφές.

2.4 Τεχνικές και τεχνολογίες για τα big data

Αν θέλουμε να είμαστε αποτελεσματικοί στον τομέα των big data, οι τεχνικές και οι μέθοδοι που χρησιμοποιούμε πρέπει να διαφοροποιηθούν από αυτές που χρησιμοποιούμε όταν έχουμε να κάνουμε με παραδοσιακά δεδομένα. Θα δούμε στη συνέχεια με ποιους τρόπους συλλέγονται τα big data και πώς μεταφέρονται. Θα δούμε ακόμα και τις νέες τεχνικές που έχουν αναπτυχθεί προκειμένου να αποθηκεύονται αποτελεσματικά τα big data και τι διαφορές υπάρχουν σε σχέση με την αποθήκευση συμβατικών δεδομένων.

2.4.1 Η συλλογή των big data

Υπάρχουν πολλοί τρόποι και μέσα με τα οποία μπορούμε να συλλέξουμε big data. Κατ' αρχάς, οι περισσότερες ψηφιακές συσκευές αλλά και σχεδόν όλα τα προγράμματα παράγουν αρχεία καταγραφής της δραστηριότητάς τους (log file). Τα αρχεία αυτά συνήθως είναι απλά αρχεία κειμένου, αν και ορισμένες εφαρμογές χρησιμοποιούν βάσεις δεδομένων για την καταγραφή της δραστηριότητάς τους. Πέραν των log file, τα big data πολύ συχνά συλλέγονται απευθείας από διαφόρων ειδών αισθητήρες, οι οποίοι μπορεί να έχουν προβεί σε στοιχειώδη προεπεξεργασία των μετρήσεών τους. Παραδείγματα αισθητήρων που χρησιμοποιούνται ευρέως στη συλλογή big data είναι αισθητήρες θερμοκρασίας, υγρασίας, δονήσεων, GPS. Μεγάλου ενδιαφέροντος είναι τα big data που αφορούν την κίνηση δικτύων υπολογιστών και τα περιεχόμενα του παγκόσμιου ιστού (world wide web), οπότε βασικό μέσον συλλογής των big data είναι πακέτα λογισμικού καταγραφής πακέτων δικτύου καθώς και εφαρμογές που σαρώνουν σελίδες του Ιστού (crawler).

2.4.2 Η αποθήκευση των big data

Ο μεγάλος όγκος των big data και η προέλευσή τους ενίοτε από διαφορετικές πηγές κάνουν επιτακτική την ανάπτυξη κατανεμημένων συστημάτων αποθήκευσης. Για την εξασφάλιση των δεδομένων σε περίπτωση βλάβης κάποιου κόμβου απαιτούνται αντίγραφα ασφαλείας για όλα τα δεδομένα, πράγμα που δημιουργεί δυσκολίες στη διαχείριση της φυσικής θέσης των αντιγράφων αυτών και δυσκολεύει την επίτευξη συνέπειας (consistency) των δεδομένων μετά από κάθε μεταβολή, καθώς πρέπει να εξασφαλιστεί η ενημέρωση περισσότερων του ενός αντιγράφων.

Με κριτήριο το αν τα big data που εξετάζονται είναι δομημένα (structured), ημιδομημένα (semi-structured), ή μη δομημένα (unstructured), αποφασίζεται και ο τύπος της βάσης δεδομένων που θα χρησιμοποιηθεί. Στην περίπτωση των δομημένων δεδομένων, οι παραδοσιακές σχεσιακές βάσεις διαθέτουν ισχυρά και αποδοτικά εργαλεία για την ανάλυση των δεδομένων. Έχουν όμως ένα βασικό μειονέκτημα: οι διεργασίες εισόδου/εξόδου (I/O – Input/Output) γίνονται τυχαία (randomly) και όχι σειριακά (sequentially). Η τυχαία πρόσβαση, όμως, είναι εξαιρετικά πιο αργή στους σκληρούς δίσκους (HDD – Hard Disk Drive) σε σχέση με τη σειριακή. Η ταχύτητα τυχαίας πρόσβασης βελτιώνεται κατά πολύ με τη χρήση αποθηκευτικών μονάδων στερεάς κατάστασης

(SSD – Solid State Drive), αλλά ακόμα και σε αυτές η σειριακή πρόσβαση παραμένει ταχύτερη και συγκρίσιμη με την αντίστοιχη των σκληρών δίσκων.

Δεδομένου ότι η σειριακή πρόσβαση στο αποθηκευτικό μέσον αυξάνει την ταχύτητα, αλλά και επειδή το 90% των big data είναι μη δομημένα, η χρήση μη σχεσιακών βάσεων δεδομένων (NoSQL – Not only SQL ή, κατ' άλλους, Non SQL) παρουσιάζει εξαιρετικό ενδιαφέρον. Οι βάσεις δεδομένων NoSQL κατηγοριοποιούνται σε τέσσερις βασικές κατηγορίες, με βάση το πώς και το τι δεδομένα αποθηκεύουν: τις βάσεις δεδομένων κλειδιού-τιμής (key-value), τις βάσεις δεδομένων αρχείων (document), τις βάσεις δεδομένων που αποθηκεύουν κατά στήλη (column-family) και τις βάσεις δεδομένων γράφων (graph). [5]

Στις βάσεις key-value όλες οι καταχωρήσεις γίνονται με μία αλφαριθμητική ακολουθία ως κλειδί και έναν απλό τύπο ή κάποια πιο πολύπλοκη δομή δεδομένων για τιμή. Οι βάσεις αυτές είναι ταχύτατες, καθώς η πρόσβαση στο αποθηκευτικό μέσον είναι κατ' εξοχήν σειριακή. Έχουν όμως το μειονέκτημα ότι η αναζήτηση μπορεί να γίνει μόνο με βάση τα κλειδιά. Τέτοιες βάσεις δεδομένων είναι μεταξύ άλλων οι Dynamo, Voldemort και Berkeley DB.

Στις document database αποθηκεύονται επίσης ζεύγη κλειδιού-τιμής (key-value), με τη διαφορά όμως ότι στο πεδίο της τιμής βρίσκονται αρχεία που περιέχουν ημιδομημένα δεδομένα σε κάποια πρότυπη μορφή (XML, JSON κλπ.). Σε αυτές τις βάσεις επιτρέπεται η αναζήτηση τόσο βάσει το κλειδιού όσο και με βάσει τιμής. Τέτοιες βάσεις είναι μεταξύ άλλων οι CouchDB και MongoDB.

Στις βάσεις δεδομένων column-family, σε αντίθεση με τις δύο προηγούμενες, σε κάθε κλειδί μπορούν να αντιστοιχούν περισσότερα του ενός πεδία. Το χαρακτηριστικό αυτών των βάσεων είναι ότι η αποθήκευση των δεδομένων στο αποθηκευτικό μέσον γίνεται κατά στήλες. Αυτό εξυπηρετεί πολύ στην κατανεμημένη αποθήκευση, καθώς τα δεδομένα κάθε στήλης μπορούν να βρίσκονται και σε άλλο data center. Επίσης, η δομή των βάσεων αυτών είναι πολύ αποδοτική για προβλήματα ταξινόμησης και μετατροπής μεγάλου όγκου δεδομένων, καθώς και άλλες στατιστικές αναλύσεις. Η πιο σημαντική βάση αυτού του είδους είναι η Bigtable της Google.

Οι graph database χρησιμοποιούν σχεσιακούς γράφους που περιγράφουν σχέσεις μεταξύ ομαδοποιημένων ζευγών κλειδιού-τιμής. Είναι χρήσιμες στις περιπτώσεις που περισσότερη σημασία έχουν οι σχέσεις μεταξύ των δεδομένων και όχι οι τιμές τους και ως τέτοιες χρησιμοποιούνται συχνά στην αναπαράσταση και διάσχιση κοινωνικών δικτύων. Παραδείγματα τέτοιων βάσεων είναι οι InfoGrid, Neo4J και AllegroGraph.

2.4.3 Η μεταφορά των big data

Η μεταφορά των big data, γίνεται σε δύο επίπεδα: εντός του data center στο οποίο βρίσκονται, καθώς και μεταξύ των data center. Στη συνέχεια αναφέρονται συνοπτικά ορισμένες από τις χρησιμοποιούμενες υποδομές αλλά και τύποι δικτύων που αφορούν κατ' εξοχήν αλλά όχι αποκλειστικά τα big data.

Ως προς την υποδομή, σημαντικό ρόλο στην κάλυψη των αυξημένων αναγκών των big data παίζουν τα οπτικά δίκτυα πολυπλεξίας με επιμερισμό μήκους κύματος (WDN – Wavelength-Division Networks), όπου η επικοινωνία μεταξύ δύο τελικών κόμβων γίνεται σε συγκεκριμένο εύρος μηκών κύματος και με χρήση οπτικού μέσου και οπτικών μεταγωγέων (optical switch) [6],

καθώς και τα δίκτυα πολυπλεξίας με επιμερισμό συχνότητας (FDM – Frequency-division networks).

Ως προς τον τύπο δικτύου, πέραν των κλασικών τοπολογιών, σημαντικά πλεονεκτήματα έχουν τα δίκτυα δυναμικού κυκλώματος (DCN – Dynamic Circuit Network), τα οποία συνδυάζουν τη μεταγωγή πακέτου (packet-switching) με τη μεταγωγή κυκλώματος (circuit-switching). Τα δίκτυα αυτά επιτρέπουν σε εφαρμογές που απαιτούν προσωρινά ή μόνιμα υψηλό ρυθμό μεταφοράς δεδομένων να τον εξασφαλίζουν με τη χρήση μεταγωγής κυκλώματος, υποστηρίζοντας όμως ταυτοχρόνως την ευρέως διαδεδομένη μεταγωγή πακέτου. [7]

2.4.4 Η επεξεργασία των big data

Για την αποδοτική επεξεργασία των big data πρέπει να χρησιμοποιούνται κατάλληλες αρχιτεκτονικές, προσαρμοσμένες στις εκάστοτε απαιτήσεις. Μια γενικευμένη πλατφόρμα επεξεργασίας πρέπει να περιλαμβάνει ευρύ φάσμα αναλυτικών μεθόδων, αναλόγως τον τύπο των big data και τον όγκο τους. Όπως είδαμε, συνήθως τα big data είναι αποθηκευμένα σε περισσότερα του ενός data center και –λόγω του μεγάλου όγκου τους– είναι ασύμφορη η μεταφορά τους. Οπότε είναι απαραίτητη η χρήση αρχιτεκτονικών και τεχνικών που επιτρέπουν την κατανεμημένη επεξεργασία τους στο χώρο όπου είναι αποθηκευμένα. Αυτό απαιτεί και ένα αποτελεσματικό σύστημα συντονισμού της όλης διαδικασίας.

Στην επεξεργασία των big data, πέραν των εξειδικευμένων τεχνικών που έχουν αναπτυχθεί ειδικά για αυτά, χρησιμοποιούνται και παραδοσιακές τεχνικές μαζικής επεξεργασίας δεδομένων. Οι τεχνικές αυτές περιλαμβάνουν μεταξύ άλλων στατιστικές μεθόδους όπως η ομαδοποίηση, η παραγοντική ανάλυση, η παραγοντική ανάλυση, η ανάλυση συσχετίσεων κ.λπ. [1] Ακόμα, χρήσιμες για την επεξεργασία big data είναι και τεχνικές επεξεργασίας σήματος, τεχνικές αναγνώρισης προτύπων (pattern recognition), τεχνικές βελτιστοποίησης, παράλληλης επεξεργασίας, καθώς και μηχανικής μάθησης και νευρωνικών δικτύων. [8]

Το υψηλό ενδιαφέρον που παρουσιάζουν τα big data καθώς και τα ιδιαίτερα χαρακτηριστικά τους οδήγησαν στην ανάπτυξη νέων τεχνικών και πλατφορμών επεξεργασίας, βελτιστοποιημένες για ανάλυση τέτοιου είδους δεδομένων. Οι γνωστότερες από αυτές είναι οι MapReduce, MPI (Message Passing Interface), Apache Mahout, Storm, Google’s Dremel και Apache Drill. Παρακάτω αναλύονται οι σημαντικότερες από αυτές.

2.4.4.1 MapReduce

Η MapReduce είναι ένα προγραμματιστικό μοντέλο που χρησιμοποιείται για παράλληλη επεξεργασία μεγάλου όγκου δεδομένων διαφόρων τύπων. Χρησιμοποιείται από πολλές πλατφόρμες σχετικές με big data, όπως το Hadoop. Είναι κατάλληλο για εργασίες ευρετηριοποίησης μεγάλου Παρουσιάζει δύο πολύ σημαντικά πλεονεκτήματα: πρώτον ότι ο ίδιος ο χρήστης δεν χρειάζεται ουσιαστικά να γράψει παράλληλο κώδικα και δεύτερον το ότι μπορεί να τρέξει σε συμβατικούς υπολογιστές. Είναι κατάλληλο για εργασίες ευρετηριοποίησης, ομαδοποίησης δεδομένων, μέτρησης του αριθμού εμφανίσεων λέξεων σε κείμενα, ταξινόμησης. Ακόμη, μπορεί να χρησιμοποιηθεί για πράξεις σχεσιακής άλγεβρας, όπως επιλογή (select), προβολή (project), ένωση (union), τομή (intersection), συνενώσεις (join).

Η επεξεργασία των δεδομένων με MapReduce έχει δύο στάδια, για καθένα από τα οποία ο χρήστης απαιτείται να υλοποιήσει μία συνάρτηση: το Map και το Reduce. Στο στάδιο Map λαμβάνονται ως είσοδος ζεύγη κλειδιού-τιμής (key-value) και λαμβάνονται ως έξοδος ενδιάμεσα ζεύγη κλειδιού-τιμής. Τα ενδιάμεσα αυτά ζεύγη ομαδοποιούνται με βάση την τιμή κλειδιού και στο στάδιο Reduce δίνονται ως είσοδος ζεύγη κλειδιού-λίστας τιμών˙ κάθε κλειδί είναι μοναδικό και στην αντίστοιχη λίστα βρίσκονται οι τιμές από όλα τα ζεύγη που προέκυψαν από το Map και αφορούσαν το συγκεκριμένο κλειδί. Το στάδιο Reduce, με τη σειρά του, για κάθε ζεύγος κλειδιού-λίστας τιμών που λαμβάνει ως είσοδο θα δώσει στην έξοδο ένα ζεύγος κλειδιού-τιμής, όπου κλειδί θα είναι το ίδιο που της δόθηκε ως είσοδος και η τιμή θα είναι το αποτέλεσμα της επεξεργασίας της λίστας τιμών της εισόδου. [9]

Για την καλύτερη κατανόησή του θα δούμε ένα απλουστευμένο παράδειγμα MapReduce σε ψευδόγλωσσα για τη συχνότητα εμφάνισης λέξεων σε ένα σύνολο αρχείων κειμένου. Η συνάρτηση map θα παίρνει ως είσοδο ζεύγη κλειδιού-τιμής, όπου κλειδί θα είναι το όνομα του κάθε αρχείου κειμένου και τιμή το περιεχόμενό του, και θα έχει ως εξής:

```
HashTable<string, int> map(string filename, string text)
{
      HashTable<string, int> occurencies;
      foreach (string word in text)
            if (occurencies.Keys.Contains(word))
                  occurencies[key]++;
            else
                  occurencies.Add(key, 1);
      return occurencies;
}
```

Από κάθε κείμενο θα προκύψει ένας πίνακας κατακερματισμού (HashTable), στον οποίο κλειδιά θα είναι οι λέξεις του κειμένου και τιμές η συχνότητα εμφάνισης των λέξεων. Στη συνέχεια, όλοι οι πίνακες κατακερματισμού θα ενωθούν σε έναν, στον οποίον κλειδιά θα είναι όλες οι λέξεις όλων των κειμένων και σε κάθε κλειδί θα αντιστοιχεί μια λίστα με τη συχνότητα εμφάνισης σε κάθε κείμενο που εμφανίζεται η λέξη αυτή. Κάθε ζεύγος λέξης-λίστας συχνοτήτων θα δοθεί τώρα ως όρισμα στη reduce, η οποία έχει ως εξής:

```
int reduce(string word, List<int> occurrences)
{
      int total_occurencies = 0;
      foreach (int n in occurrences)
            total_occurences += n;
      return total_occurences;
}
```

Η reduce επιστρέφει το άθροισμα των φορών που συναντήθηκε κάθε λέξη συνολικά σε όλα τα κείμενα.

2.4.4.2 Storm

Το Storm αναπτύχθηκε από την Canonical και αποτελεί ελεύθερο λογισμικό. Είναι μία βιβλιοθήκη ανεπτυγμένη για τη γλώσσα προγραμματισμού Python που παρέχει λειτουργικότητα για αντιστοίχιση (mapping) σχεσιακών βάσεων δεδομένων με αντικείμενα.. [10] Υποστηρίζει μέχρι

στιγμής τις βάσεις δεδομένων SQLite, MySQL και PostgreSQL, ενώ διαθέτει κατάλληλη διεπαφή προγραμματισμού εφαρμογών (API – Application Programming Interface) για την υλοποίηση της διασύνδεσης με άλλες σχεσιακές βάσεις δεδομένων από τον χρήστη. Με τον τρόπο αυτό επιτρέπει την εύκολη συνδυασμένη επεξεργασία δεδομένων που προέρχονται από διαφορετικές βάσεις δεδομένων, αντιμετωπίζει έτσι έναν από τους παράγοντες ετερογένειας των big data.

2.5 Internet of Things (IoT)

Με τον όρο «Internet of Things» (IoT) αναφερόμαστε σε ένα παγκόσμιο δίκτυο διασυνδεδεμένων αντικειμένων, καθένα από τα οποία θα διαθέτει μοναδική διεύθυνση· το δίκτυο αυτό βασίζεται σε προτυποποιημένα πρωτόκολλα επικοινωνίας. [11]

Το IoT, εξαιτίας του πλήθους των συσκευών που χρησιμοποιεί, δημιουργεί πολλούς κινδύνους παραβίασης της ιδιωτικότητας. Ο μεγάλος αριθμός και η ευρεία τοποθέτηση των αισθητήρων και ετικετών RFID, καθώς και ο μεγάλος όγκος των δεδομένων που παράγονται είναι δύσκολο να ελεγχθούν· είναι λοιπόν εξαιρετικά πιθανόν να παρακολουθούνται άμεσα ή έμμεσα (συνάγοντας δηλαδή από άλλα δεδομένα) συνήθειες και συμπεριφορές που δεν επιθυμεί ο χρήστης. Αυτό συμβαίνει κυρίως κατά την παρουσία ενός ανθρώπου σε χώρους που καλύπτουν δίκτυα εγκατεστημένα και διαχειριζόμενα από τρίτους. Μπορεί όμως να συμβεί και εντός δικτύων που ανήκουν στον ίδιο το χρήστη, είτε ακουσίως, εξαιτίας ελλιπούς μελέτης πριν την εγκατάσταση, είτε εκουσίως, με κακόβουλη χρήση ξένων ως προς το δίκτυο αναγνωστών και αισθητήρων.

Δεδομένης της πολύ απλής φύσης των συσκευών που χρησιμοποιούνται, η κρυπτογράφηση και η πιστοποίηση κατά τη μετάδοση των συλλεγομένων δεδομένων είναι πρακτικά αδύνατη. Αυτό καθιστά ιδιαίτερα εύκολες τις επιθέσεις τύπου man-in-the-middle, καθώς στην πλειοψηφία των περιπτώσεων η μετάδοση γίνεται ασύρματα. Επιπλέον, λόγω του όλο και φθηνότερου κόστους αποθήκευσης των δεδομένων, είναι πολύ πιθανόν τα δεδομένα που συλλέγονται να διατηρηθούν για πολλά έτη και σε πολλά αντίγραφα, αφήνοντας έτσι τον χρήστη εκτεθειμένο σε βάθος χρόνου και πρακτικά χωρίς δυνατότητα διαγραφής των δεδομένων που τον αφορούν. Για το λόγο αυτό είναι απαραίτητο τα δεδομένα να αποθηκεύονται μόνο εφ’ όσον είναι απολύτως απαραίτητο και να διαγράφονται το συντομότερο δυνατόν, εκτός αν υποστούν επεξεργασία ανωνυμοποίησης.

2.6 Προκλήσεις και ανοικτά ζητήματα

Τα big data είναι ένα καινούργιο σχετικά αντικείμενο, οπότε μένουν αρκετά ανοιχτά ζητήματα, ειδικά όσον αφορά στο κομμάτι της θεωρίας. Κατ’ αρχάς, όπως ήδη αναφέρθηκε, η ακαδημαϊκή κοινότητα δεν έχει ακόμη καταλήξει σε έναν ορισμό του τι ακριβώς είναι big data. Επίσης, πρέπει να αναπτυχθούν κατάλληλα μοντέλα και να βρεθεί ένα σύστημα για τη θεωρητική τους ανάλυση. Επιπλέον, απομένει να αναπτυχθούν πρότυπα για την αξιολόγηση τόσο των ιδίων των big data όσο και της επεξεργαστικής ικανότητας των συστημάτων που τα επεξεργάζονται. Το τελευταίο είναι ιδιαιτέρως σημαντικό, ούτως ώστε να μπορούμε να τα γνωρίζουμε εκ των προτέρων την αποτελεσματικότητα ενός συστήματος στη φάση της σχεδίασης και όχι της λειτουργίας, όπως

γίνεται σήμερα. Τέλος, χρειάζεται να αναπτυχθούν νέες υπολογιστικές μέθοδοι, κατάλληλες προς τα ογκώδη και ως επί το πλείστον κατανεμημένα big data.

Εκτός από τη θεωρία, είναι πολλά τα θέματα που αναζητούν επίλυση και σε τεχνολογικό επίπεδο. Η μεγάλη ετερογένεια που παρουσιάζεται στα big data κάνει επιτακτική την ανάγκη ανάπτυξη τεχνικών για τη γρήγορη και αποτελεσματική μετατροπή από έναν τύπο δεδομένων σε άλλο. Ο μεγάλος όγκος, ο υψηλός ρυθμός παραγωγής των big data, καθώς και το γεγονός ότι είναι κατανεμημένα, μας αναγκάζουν να αναζητούμε τρόπους για ταχύτερη μεταφορά των δεδομένων αλλά και αποτελεσματικότερη ανάλυσή τους τη στιγμή που παράγονται (real-time) και στο σημείο όπου βρίσκονται. Επιπλέον, χρειάζονται νέες τεχνικές που θα βελτιώσουν τις ικανότητές μας να ξεχωρίζουμε, ακόμα και να διορθώνουμε τυχόν ανακριβή δεδομένα τη στιγμή της συλλογής τους και τεχνικές με τις οποίες μπορούμε να οργανώνουμε αποτελεσματικότερα τα big data.

Τέλος, σημαντικά είναι τα ανοικτά ζητήματα σε θέματα ασφαλείας των big data. Η ανάπτυξη κατάλληλων τεχνικών ανωνυμοποίησης (anonymization) και όχι μόνο είναι απαραίτητη για την εξασφάλιση της ανωνυμίας ευαίσθητων ή απόρρητων δεδομένων. Επίσης, χρειάζεται να εξασφαλιστεί ότι τα δεδομένα παρέχονται με τη συναίνεση αυτών τους οποίους αφορούν και ότι παραμένουν ασφαλή τόσο κατά τη μεταφορά τους, όσο κατά την αποθήκευσή της και την επεξεργασία τους. Για το σκοπό αυτό κρίνεται απαραίτητη η ανάπτυξη νέων τεχνικών κρυπτογράφησης, καθώς οι υπάρχουσες είναι εξαιρετικά αργές και μη αποδοτικές για μεγάλους όγκους δεδομένων. Η ασφάλεια των big data, όμως, έχει και έμμεσες συνέπειες, καθώς η μεταφορά, επεξεργασία και αποθήκευση δεδομένων τα οποία δεν είναι ακριβή, είναι αλλοιωμένα ή γενικά χαμηλής ποιότητας, αυξάνουν το κόστος σε δικτυακούς, αποθηκευτικούς και επεξεργαστικούς πόρους, αυξάνοντας εν τέλει το χρόνο και το κόστος που απαιτείται για την ανάλυσή τους. [1]

3 Cloud Computing

Οι αυξημένες ανάγκες για υπολογιστική ισχύ και για αποθήκευση δεδομένων καθιστούν τους προσωπικούς υπολογιστές (PC) ανεπαρκείς για πολλές διεργασίες. Απαιτούνται λοιπόν διατάξεις με πολύ περισσότερες δυνατότητες για την αποθήκευση και την επεξεργασία των δεδομένων. Οι διατάξεις αυτές όμως είναι πολύ πιο δαπανηρές και συνήθως δεν χρησιμοποιούνται στο μέγιστο των δυνατοτήτων τους για μεγάλο χρονικό διάστημα –παρά μόνο λίγες χρονικές στιγμές ημερησίως. Μία εταιρεία λοιπόν που χρειάζεται να επεξεργαστεί ή/και να αποθηκεύσει μεγάλο όγκο δεδομένων, θα έπρεπε να επενδύσει στην αγορά ενός συστήματος με ικανότητες που θα παραμένουν τον περισσότερο χρόνο ανεκμετάλλευτες.

Οι δυνατότητες που μας παρέχουν οι σημερινές τεχνολογίες δικτύωσης και επικοινωνίας υπολογιστών, τόσο σε τοπικό (δίκτυα), όσο και σε πιο απομακρυσμένο (διαδίκτυο) επίπεδο, δίνουν τη δυνατότητα στην εταιρεία του παραδείγματός μας να χρησιμοποιήσει ένα υπολογιστικό σύστημα που μπορεί να μην της ανήκει –και συχνά δεν βρίσκεται καν στις εγκαταστάσεις της– για όσο το χρειάζεται και δεσμεύοντας μόνο τους πόρους που της είναι απαραίτητοι. Τέτοια συστήματα ανήκουν στην κατηγορία των συστημάτων cloud computing.

3.1 Ορισμός

Με τον όρο cloud αναφερόμαστε σε μία «δεξαμενή» πολλών εύχρηστων και ευπρόσιτων εικονικοποιημένων πόρων (όπως υλικό (hardware), πλατφόρμες ή/και υπηρεσίες ανάπτυξης λογισμικού). Οι πόροι που χρησιμοποιούνται μπορούν να επαναρρυθμίζονται δυναμικά, ούτως ώστε να προσαρμόζονται σε μεταβαλλόμενες απαιτήσεις και να επιτρέπουν τη βέλτιστη χρήση τους. Ο χρήστης αυτής της «δεξαμενής» πόρων συνήθως χρεώνεται αναλόγως της χρήσης του και ο πάροχος της υποδομής εγγυάται γι’ αυτό μέσω ενός προσαρμοσμένου SLA (Service-level agreement – συμφωνία επιπέδου υπηρεσίας). [12]

3.2 Τα χαρακτηριστικά του cloud computing

3.2.1 Αυτόματη απόδοση πόρων

Τα συστήματα cloud έχουν αναπτυχθεί ώστε η απόδοση πόρων, η εύρεση και η χρήση των διαθέσιμων υπηρεσιών να γίνεται από τους χρήστες χωρίς να απαιτείται ανθρώπινη παρέμβαση από πλευράς του παρόχου.

3.2.2 Ευρεία πρόσβαση μέσω του δικτύου

Η πρόσβαση στο σύστημα cloud γίνεται μέσω δικτύου. Είναι δυνατή η χρήση πολλών μέσων για το σκοπό αυτό, όπως εφαρμογές για κινητά, προγράμματα-πελάτες για προσωπικούς υπολογιστές, σελίδες ιστού για χρήση από φυλλομετρητή (browser) κλπ.

3.2.3 Εικονικοποίηση πόρων (resource virtualization)

Οι πόροι που χρησιμοποιούνται στα συστήματα cloud δεν είναι ορατοί από τον χρήστη. Αντιθέτως, όπως θα δούμε και αργότερα, μεσολαβεί ένα αφαιρετικό επίπεδο, το οποίο παρουσιάζει στο χρήστη εικονικούς πόρους, καθένας εκ των οποίων μπορεί να αντιστοιχεί σε μέρος μόνο ενός φυσικού πόρου ή και σε περισσότερους φυσικούς πόρους ταυτοχρόνως. Αυτό επιτρέπει τη συλλογική παρουσίαση στο χρήστη πόρων και υπηρεσιών που μπορεί να βρίσκονται/εκτελούνται στο ίδιο μηχάνημα ή σε μηχανήματα που βρίσκονται σε διαφορετικές τοποθεσίες (ακόμα και σε άλλη ήπειρο). Τέλος, η αντιστοίχιση των εικονικών πόρων στους φυσικούς γίνεται με δυναμικό τρόπο, αναλόγως των απαιτήσεων των χρηστών.

3.2.4 Ελαστικότητα (elasticity) και επεκτασιμότητα (scalability)

Βασικό χαρακτηριστικό των συστημάτων cloud είναι η δυνατότητά τους να προσαρμόζονται στις αυξομειώσεις του φόρτου εργασίας που δέχονται, με την αυτόματη διάθεση ή περιορισμό των χρησιμοποιούμενων πόρων (επεξεργαστική ισχύς, δίκτυο, μνήμη, αποθηκευτικός χώρος). Η χρήση ακριβώς τόσων πόρων όσων χρειάζονται είναι εξαιρετικά σημαντική τόσο για τον πάροχο, ο οποίος εξοικονομεί έτσι πόρους για να τους διαθέσει σε άλλους χρήστες, όσο όμως και για τους ίδιους τους χρήστες, καθώς όπως είδαμε η τιμολογιακή πολιτική συνήθως εξαρτάται από τους πόρους που χρησιμοποιούνται και τη διάρκεια χρήση τους.

Η φύση του cloud επιτρέπει (με σωστό σχεδιασμό) την επεκτασιμότητα των υπηρεσιών που «τρέχουν» σε αυτό χωρίς να απαιτούνται σημαντικές αλλαγές στην αρχιτεκτονική τους. Αυτό απαιτεί βασικό πλεονέκτημα στα στάδια ανάπτυξης εταιρικών χρηστών, καθώς δεν απαιτείται πλέον ανασχεδιασμός των εφαρμογών τους καθώς αυξάνονται οι απαιτήσεις.

3.2.5 Παρακολούθηση των αποδιδόμενων πόρων

Τα συστήματα cloud παρακολουθούν αυτοματοποιημένα τη χρήση των πόρων τους (είτε πρόκειται για υλικό, είτε για κίνηση δικτύου, είτε για λογισμικό) καθώς και το μέρος από αυτή που αντιστοιχεί σε κάθε χρήστη. Αυτό είναι απαραίτητο τόσο για την αυτόματη απόδοση και απελευθέρωση πόρων όσο και για την τιμολόγηση και επίβλεψη της πολιτικής χρήσης. [13]

3.3 Είδη παρεχόμενων υπηρεσιών

Ανάλογα με το είδος των παρεχόμενων υπηρεσιών, τα συστήματα cloud κατατάσσονται σε τρεις βασικές κατηγορίες: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) και Software as a Service (SaaS). [13]

3.3.1 Infrastructure as a Service (IaaS)

Σε αυτόν το είδος παροχής υπηρεσιών cloud αποδίδονται στο χρήστη πόροι, όπως εικονικά μηχανήματα (virtual machine), αποθηκευτικός χώρος, ευρυζωνικό δίκτυο κ.λπ. Ο χρήστης μπορεί πάνω σε αυτά να εγκαταστήσει το δικό του λειτουργικό σύστημα, εφαρμογές κ.ά. Πρέπει να τονιστεί εδώ ότι συνήθως ο χρήστης έχει πρόσβαση σε εικονικούς πόρους, έτσι δεν μπορεί να

διαχειριστεί απευθείας τους φυσικούς πόρους του συστήματος, παρά μόνο να του αποδοθούν προς χρήση μέσω των εικονικών πόρων που χρησιμοποιεί.

3.3.2 Platform as a Service (PaaS)

Σε αυτήν την περίπτωση ο πάροχος δίνει στο χρήστη συγκεκριμένο λειτουργικό σύστημα και ένα πακέτο από πλατφόρμες και εργαλεία στα οποία ο χρήστης μπορεί να εκτελεί (ή και να αναπτύσσει) τις εφαρμογές του. Οι υποστηριζόμενες γλώσσες προγραμματισμού καθορίζονται συνήθως από τον πάροχο και είναι προεγκατεστημένες στην παρεχόμενη υπηρεσία.

Τα τελευταία χρόνια μεταξύ άλλων έχει αρχίσει να γίνεται δημοφιλές και το dPaaS, όπου τη διαχείριση των αποθηκευμένων δεδομένων αναλαμβάνει ο πάροχος και όχι ο χρήστης.

3.3.3 Software as a Service (SaaS)

Εδώ ο χρήστης μπορεί να διαλέξει μεταξύ των έτοιμων εφαρμογών που έχει αναπτύξει ο πάροχος. Η επικοινωνία με αυτές τις εφαρμογές καθώς και η παραμετροποίησή τους γίνεται είτε με χρήση ιστοσελίδας (web-based), είτε με χρήση εφαρμογής-πελάτη (client application). Με αυτό το μοντέλο παροχής υπηρεσιών cloud οι χρήστες μπορούν να εξοικονομήσουν χρόνο και κόστος, καθώς δεν χρειάζεται να αναπτύξουν τις δικές τους εφαρμογές ή να εγκαταστήσουν εφαρμογές τρίτων στα συστήματά τους. Επίσης, οι δημιουργοί λογισμικού, εφ' όσον επιλέξουν το cloud ως μέσον για την παροχή των υπηρεσιών τους, μπορούν να ενημερώνουν και συντηρούν το λογισμικό τους ευκολότερα, καθώς και να αναπτύσσουν μία μόνο έκδοση, αυτή για το λειτουργικό σύστημα του cloud, και όχι για κάθε λειτουργικό ξεχωριστά.

Ενδιαφέρουσα υποκατηγορία του SaaS είναι το λεγόμενο Business as a Service (BaaS). Πρόκειται για παροχή υπηρεσιών που παραδοσιακά θα παρέχονταν ως SaaS για τον έλεγχο και τη διαχείριση του κεφαλαίου, των πόρων και των στόχων μιας εταιρείας. Σκοπός είναι να εκτελείται ο σχεδιασμός δράσης της επιχείρησης, να τίθενται μελλοντικοί στόχοι και να ελέγχεται ο βαθμός επίτευξής τους. [14]

3.4 Η αρχιτεκτονική των συστημάτων cloud

Η δομή των συστημάτων cloud μπορεί να χωριστεί νοητά σε τρία επίπεδα: αυτό των φυσικών πόρων, ένα επίπεδο αφαίρεσης και ελέγχου πόρων καθώς και το επίπεδο υπηρεσιών. [15]

3.4.1 Επίπεδο φυσικών πόρων

Στη βάση ενός συστήματος cloud βρίσκεται το υλικό (hardware), που περιλαμβάνει τους επεξεργαστές, τη μνήμη, τους χώρους αποθήκευσης, τα δίκτυα μαζί με όλη τους την υποδομή καθώς και άλλες υπολογιστικές διατάξεις. Αυτά μπορεί να βρίσκονται συγκεντρωμένα στον ίδιο χώρο ή σε πολλούς διαφορετικούς χώρους, ακόμα και σε διαφορετικές ηπείρους. Στο επίπεδο αυτό επίσης εντάσσονται κατά το NIST (National Institute of Standards and Technology – Αμερικανικό Ινστιτούτο Προτύπων και Τεχνολογίας) και οι εγκαταστάσεις που στεγάζουν το hardware, καθώς και ο εξαερισμός, η ηλεκτρική ισχύς κ.λπ.

3.4.2 Επίπεδο αφαίρεσης και ελέγχου πόρων

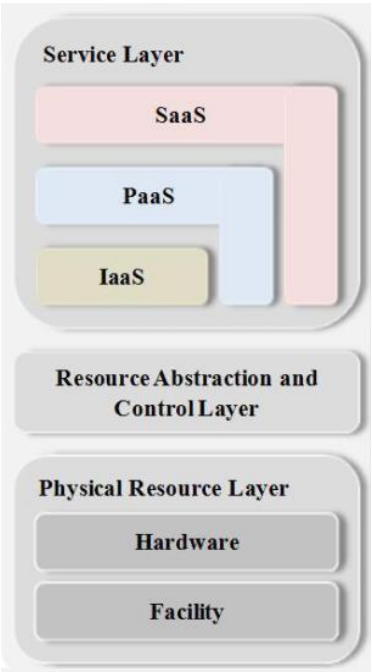
Το επίπεδο αυτό, όπως δηλώνει και το όνομά του, έχει διττό ρόλο: πρώτον την αφαιρετική παροχή των φυσικών πόρων του χαμηλότερου επιπέδου και δεύτερον τον έλεγχο της χρήσης τους.

Η αφαιρετική παροχή των φυσικών πόρων γίνεται αντιστοιχίζοντάς τους σε εικονικούς πόρους, όπως εικονικά μηχανήματα (virtual machines), hypervisor και εικονικές μονάδες αποθήκευσης. Ένας εικονικός πόρος μπορεί να αντιστοιχεί σε μέρος μόνο ενός φυσικού πόρου, σε ολόκληρο τον πόρο, ή σε περισσότερους από έναν πόρους.

Ο έλεγχος χρήσης αφορά κατ' αρχάς την απόδοση και απελευθέρωση των φυσικών πόρων αναλόγως με τις απαιτήσεις του χρήστη και την τιμολογιακή πολιτική στην οποία εντάσσεται. Ελέγχει όμως και την πρόσβαση στους πόρους αυτούς, ούτως ώστε να εξασφαλίζεται η ιδιωτικότητα τον δεδομένων που αποθηκεύονται εντός του συστήματος ή ακόμη εντός του ιδίου μοιραζόμενου φυσικού πόρου. Τέλος, καταγράφει ποιος χρησιμοποιεί κάθε πόρο και πόσο, ούτως ώστε να μπορεί στη συνέχεια ο χρήστης να χρεωθεί αναλόγως.

3.4.3 Επίπεδο υπηρεσιών

Σε αυτό το επίπεδο βρίσκονται οι παρεχόμενες από το σύστημα υπηρεσίες. Χρησιμοποιούν για την εκτέλεσή τους τους εικονικούς πόρους του προηγούμενου επιπέδου. Όπως φαίνεται και στο σχήμα, οι υπηρεσίες PaaS μπορούν να χρησιμοποιούν τυχόν υπάρχουσες υπηρεσίες IaaS του συστήματος για να εξασφαλίσουν λειτουργικό σύστημα, ή να είναι απευθείας δομημένες πάνω στο επίπεδο αφαίρεσης και ελέγχου. Το ίδιο ισχύει και για τις υπηρεσίες SaaS, που μπορεί να χρησιμοποιούν κάποιες εκ των IaaS και PaaS, αν δεν έχουν δομηθεί απ' ευθείας πάνω στο επίπεδο αφαίρεσης.



Σχήμα 1: Η αρχιτεκτονική ενός συστήματος cloud

3.5 Τύποι cloud

Αναλόγως με το ποιος έχει πρόσβαση σε ένα σύστημα cloud computing, μπορούμε να το κατατάξουμε σε μία από τις τέσσερεις ακόλουθες κατηγορίες.

3.5.1 Δημόσιο cloud

Εφ' όσον είναι διαθέσιμο στο ευρύ κοινό μέσω ενός δημόσιου δικτύου, τότε το cloud χαρακτηρίζεται ως δημόσιο.

3.5.2 Ιδιωτικό cloud

Όταν η πρόσβαση σε ένα σύστημα cloud είναι αποκλειστικά και μόνο διαθέσιμη στο χρήστη ή στους (εταιρικούς) χρήστες που αυτός επιλέγει, τότε πρόκειται για ένα ιδιωτικό cloud. Οι φυσικοί πόροι ενός ιδιωτικού cloud μπορεί να είναι εγκατεστημένοι τόσο στις εγκαταστάσεις του χρήστη (on-site) όσο και στις εγκαταστάσεις του παρόχου (outsourced).

3.5.3 Κοινοτικό (community cloud)

Υπάρχουν περιπτώσεις όπου το ίδιο σύστημα cloud το μοιράζονται πολλοί χρήστες, χωρίς όμως αυτό να είναι δημόσιο. Παραδείγματος χάριν ένας όμιλος εταιρειών που όλες τους έχουν δεδομένα ή υπηρεσίες που επιθυμούν να χρησιμοποιούν από κοινού. Σε αυτή την περίπτωση χρησιμοποιούν ένα κοινοτικό cloud, στο οποίο έχουν όλες τους πρόσβαση. Οι φυσικοί πόροι αυτού του cloud μπορεί να βρίσκονται κατανεμημένοι στις ίδιες τις εταιρείες (on-site) ή στις εγκαταστάσεις ενός παρόχου (outsourcing).

3.5.4 Υβριδικό cloud

Υβριδικό καλείται οποιοδήποτε cloud αποτελείται από δύο ή περισσότερα cloud των προαναφερθεισών κατηγοριών, τα οποία είναι διασυνδεδεμένα μεταξύ τους για την ανταλλαγή δεδομένων ή την εκατέρωθεν παροχή υπηρεσιών.

3.6 Προκλήσεις και ανοικτά ζητήματα

Καθώς το cloud computing αποτελεί μια σχετικά νέα τεχνολογία, έχει ακόμα να αντιμετωπίσει αρκετές προκλήσεις, κυρίως σε θέματα ασφαλείας, προτυποποίησης και νομικών θεμάτων. [16]

3.6.1 Ασφάλεια και ιδιωτικότητα

Βασικότερος ίσως παράγοντας επισφάλειας των συστημάτων cloud αποτελεί το γεγονός ότι σε πολλές περιπτώσεις περισσότεροι του ενός χρήστες μοιράζονται τους ίδιους φυσικούς πόρους. Είναι αναγκαίο λοιπόν να εξασφαλιστεί η ορθότητα των δεδομένων και η ιδιωτικότητά τους. Επίσης, όπως και άλλα συστήματα που χρησιμοποιούν δίκτυα, έτσι και τα συστήματα cloud κινδυνεύουν από επιθέσεις denial-of-service (DoS), καθώς και άλλους τρόπους υποκλοπής δεδομένων μέσω φυσικής πρόσβασης στους χώρους αποθήκευσης και επεξεργασίας τους.

3.6.2 Έλλειψη προτύπων

Αυτή τη στιγμή δεν έχουν θεσπιστεί καθολικώς αποδεκτά πρότυπα για τη δομή και την περιγραφή υπηρεσιών cloud. Ως εκ τούτου κάθε πάροχος ακολουθεί τον δικό του σχεδιασμό. Αυτό εγείρει δύο πολύ σημαντικά προβλήματα: πρώτον, παρεχόμενες υπηρεσίες από διαφορετικά cloud δεν μπορούν να συνδυαστούν χωρίς τη χρήση middleware· δεύτερον, οι εφαρμογές του χρήστη που σχεδιάζονται μπορούν να λειτουργήσουν μόνο στον πάροχο για τον οποίο έχουν σχεδιαστεί. Αυτό ειδικά το τελευταίο πρόβλημα κάνει τους χρήστες ιδιαίτερα διστακτικούς στο να μετακινήσουν τις εφαρμογές τους σε περιβάλλον cloud, διότι μετά η αλλαγή παρόχου θα σημάνει υλοποίησή τους σχεδόν εξ αρχής.

3.6.3 Νομικά και άλλα ζητήματα

Δεδομένης της κατανεμημένης φύσης των συστημάτων cloud, είναι πρακτικά πολύ δύσκολο να γνωρίζει ο χρήστης την ακριβή φυσική τοποθεσία όπου βρίσκονται τα δεδομένα του ή εκτελούνται οι εφαρμογές του. Αυτό εγείρει νομικά ζητήματα, καθώς η νομοθεσία μιας χώρας ως προς την ιδιωτικότητα των δεδομένων μπορεί να έρχεται σε αντίθεση με αυτή μιας άλλης· θα δημιουργηθεί επομένως νομικό θέμα αν τα δεδομένα που ανήκουν σε μια εταιρεία της πρώτης χώρας καταλήξουν σε φυσική τοποθεσία εντός της επικρατείας της δεύτερης. Ακόμα, το γεγονός ότι μια εταιρεία παροχής υπηρεσιών cloud μπορεί να κάνει outsourcing σε κάποια μέρη του συστήματος cloud της και αυτό να συνεχιστεί και σε δεύτερο και τρίτο επίπεδο, καθιστά την πιστοποίηση των προδιαγραφών των παρεχόμενων από την πρώτη εταιρεία υπηρεσιών εξαιρετικά δύσκολη υπόθεση.

Τέλος, πολλά από τα διαθέσιμα πακέτα λογισμικού δεν τιμολογούνται με βάση τη χρήση, αλλά ανά άδεια χρήστη, λογική ασύμβατη με την προοδευτική χρέωση των υπηρεσιών cloud.

4 Ουρές μηνυμάτων (Message Queues)

Η λειτουργία των κατανεμημένων συστημάτων απαιτεί την ανταλλαγή δεδομένων μεταξύ των υπηρεσιών που εκτελούνται σε αυτά. Ως εκ τούτου, οι υπηρεσίες αυτές χρειάζεται να βρουν τρόπους για την ασφαλή και γρήγορη αποστολή και λήψη των δεδομένων που επεξεργάζονται. Ο μεγάλος όγκος των δεδομένων, που είναι σήμερα βασικό στοιχείο πολλών εφαρμογών, ειδικά στο πεδίο των big data που εξετάζουμε, δημιουργεί δυσκολίες στη μετάδοση των δεδομένων για δύο βασικούς λόγους: πρώτον μία υπηρεσία μπορεί να παράγει δεδομένα με πολύ υψηλότερους ρυθμούς από αυτούς που μπορεί να τα μεταδώσει μέσω του δικτύου και δεύτερον μία υπηρεσία μπορεί να δέχεται δεδομένα με πολύ υψηλότερο ρυθμό από αυτόν που μπορεί να τα επεξεργαστεί.

Εγείρεται λοιπόν το θέμα της διαχείρισης και της προσωρινής αποθήκευσης των δεδομένων που απαιτούν επεξεργασία ή αναμένουν να αποσταλούν. Αυτό το ζήτημα με τη σειρά του ενέχει άλλη μία παράμετρο η οποία αυξάνει την πολυπλοκότητα της επίλυσής του: τι θα συμβεί με τα δεδομένα αυτά σε περίπτωση που η υπηρεσία αποτύχει. Προκειμένου να μην απαιτηθεί η εκ νέου επεξεργασία ή αποστολή τους, πράγμα που μπορεί να έχει σημαντικό κόστος σε χρόνο και υπολογιστικούς/δικτυακούς πόρους, πρέπει να εξασφαλιστεί ότι τα δεδομένα αυτά θα παραμείνουν αποθηκευμένα σε περίπτωση αποτυχίας και μάλιστα με τρόπο που να μας επιτρέπει να συνεχίσουμε από το σημείο που είχαμε μείνει. Προκειμένου να επιλυθούν τα ζητήματα αυτά, οι υπηρεσίες έχουν δύο επιλογές, είτε να υλοποιήσουν πολύπλοκα πρωτόκολλα επικοινωνίας προκειμένου να εξασφαλίσουν την απρόσκοπτη λειτουργία τους και τη διατήρηση των δεδομένων που στέλνουν ή λαμβάνουν, είτε να χρησιμοποιήσουν κάποιο είδος ενδιάμεσου λογισμικού μηνυμάτων (message-oriented middleware), όπως οι ουρές μηνυμάτων (message queue).

4.1 Ορισμός

Με τον όρο ουρές μηνυμάτων αναφερόμαστε σε λογισμικά που αναλαμβάνουν τη μετάδοση μηνυμάτων μεταξύ νημάτων (thread), διεργασιών (process) ή ακόμα και υπηρεσιών (service) που εκτελούνται στο ίδιο ή σε διαφορετικά μηχανήματα. Παρέχουν διεπιφάνεια προγραμματισμού εφαρμογών (API – Application Programming Interface) και συνήθως παρέχουν λειτουργικότητα εξασφάλισης των μηνυμάτων σε περίπτωση αποτυχίας του συστήματος καθώς και εγγυημένης παράδοσης των μηνυμάτων.

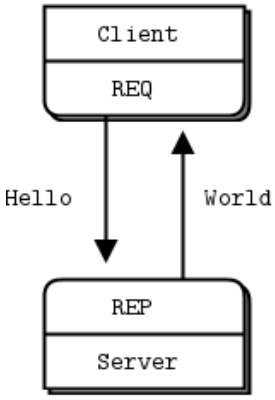
4.2 Αρχιτεκτονικές ουρών μηνυμάτων

Υπάρχουν διάφορες διατάξεις και λειτουργικότητες ουρών, προσανατολισμένες τόσο σε σύγχρονη όσο και σε ασύγχρονη επικοινωνία, καθώς και μεταξύ ενός ή περισσοτέρων αποστολέων/παραληπτών (1:1, 1:N, N:N). [17] [18] Θα δούμε αυτές τις περισσότερες από τις αρχιτεκτονικές μέσα από την υλοποίησή τους στην ουρά μηνυμάτων ZeroMQ (ØMQ), καθώς αυτή θα χρησιμοποιηθεί στην παρούσα διπλωματική.

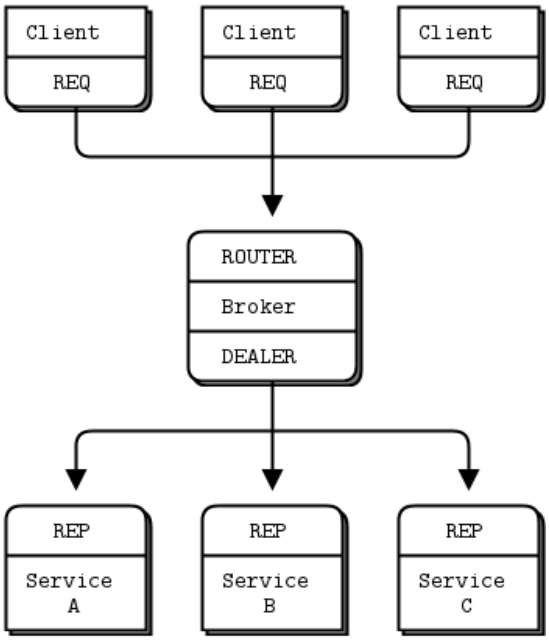
4.2.1 Σύγχρονη επικοινωνία

4.2.1.1 Το πρότυπο request–reply

Η σύγχρονη επικοινωνία στη ZeroMQ υλοποιείται με τη χρήση των socket `REQ` (request) και `REP` (reply). Το socket `REQ` είναι αυτό που πρέπει να στείλει πρώτο μήνυμα και να περιμένει την απάντηση από το socket `REP`. Σε περίπτωση που δεν λάβει μήνυμα απάντησης για το μήνυμα που έχει στείλει, δεν μπορεί να στείλει και δεύτερο μήνυμα. Αναλόγως, το socket `REP` δεν μπορεί να ξεκινήσει αυτό την επικοινωνία, παρά μόνο να απαντήσει στα μηνύματα του `REQ`. Το πρότυπο αυτό είναι κατάλληλο για εργασίες τύπου πελάτη–εξυπηρετητή (client server). Περισσότερα από ένα `REQ` socket (πελάτης) μπορούν να συνδεθούν στο ίδιο `REP` socket, το οποίο αναλαμβάνει και τη σωστή δρομολόγηση της απάντησης.



Σχήμα 2: Το πρότυπο request–reply



Σχήμα 3: Το επεκτεταμένο πρότυπο request–reply

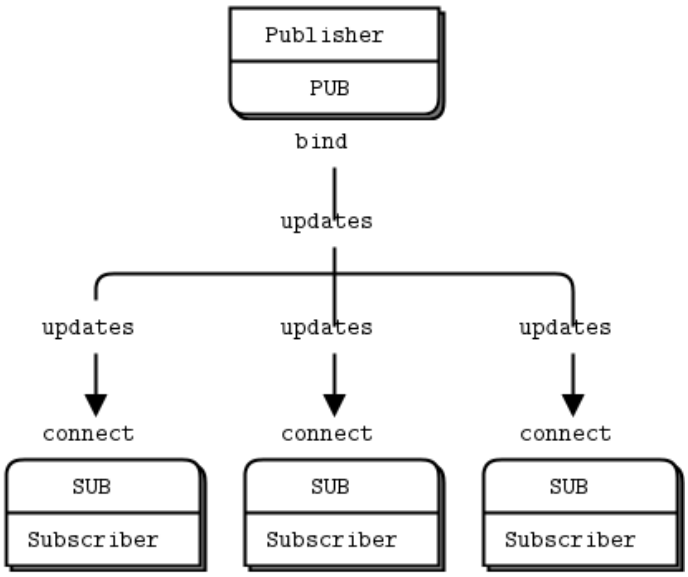
4.2.1.2 *Επέκταση του προτύπου request-reply*

Στην περίπτωση που χρειάζεται να προστεθούν περισσότερα μηχανήματα στην πλευρά του reply, προκειμένου για παράδειγμα να επιμεριστεί ο φόρτος εργασίας, μπορούν να χρησιμοποιηθούν τα socket `ROUTER` και `DEALER`, τα οποία μπορούν να υλοποιήσουν ασύγχρονη επικοινωνία. Το socket `ROUTER` δίνει τη δυνατότητα στην εφαρμογή που το χρησιμοποιεί να επιλέξει σε ποιο socket θέλει να στείλει το μήνυμα. Το socket `DEALER` προωθεί τα μηνύματα σε όλα τα συνδεδεμένα σε αυτό socket με λογική round-robin. Αυτό το πρότυπο είναι εξαιρετικά χρήσιμο στην περίπτωση ύπαρξης μεταβλητού αριθμού εξυπηρετητών, καθώς ο πελάτης μπορεί να συνδέεται στο μεσίτη (broker), αντί σε κάποιον από τους εξυπηρετητές, τους οποίους θα έπρεπε να αναζητά κάθε φορά.

4.2.2 Ασύγχρονη επικοινωνία

4.2.2.1 *Το πρότυπο publish–subscribe*

Το συγκεκριμένο πρότυπο υποστηρίζει την αποστολή μηνυμάτων από έναν αποστολέα, χρησιμοποιώντας το socket `PUB`, σε πολλούς παραλήπτες, οι οποίοι όμως έχουν τη δυνατότητα να επιλέγουν ποια μηνύματα θα λαμβάνουν στα SUB socket τους, εγγραφόμενοι σε αυτά χρησιμοποιώντας ένα topic. Το topic στην παρούσα υλοποίηση είναι μία κανονική έκφραση (regular expression), η οποία, εφ' όσον ταιριάζει με την αρχή ενός μηνύματος, τότε το μήνυμα αυτό προωθείται στο `SUB` socket που εγγράφηκε χρησιμοποιώντας το topic αυτό. Σε περίπτωση που κάποια ουρά ενός `SUB` socket γεμίσει από μηνύματα και δεν μπορεί να δεχθεί άλλα, τότε τα επιπλέον μηνύματα που στέλνονται σε αυτό το `SUB` socket απορρίπτονται (drop). Τα `PUB` socket μπορούν μόνο να στέλνουν μηνύματα και τα `SUB` socket μπορούν μόνο να λαμβάνουν μηνύματα.

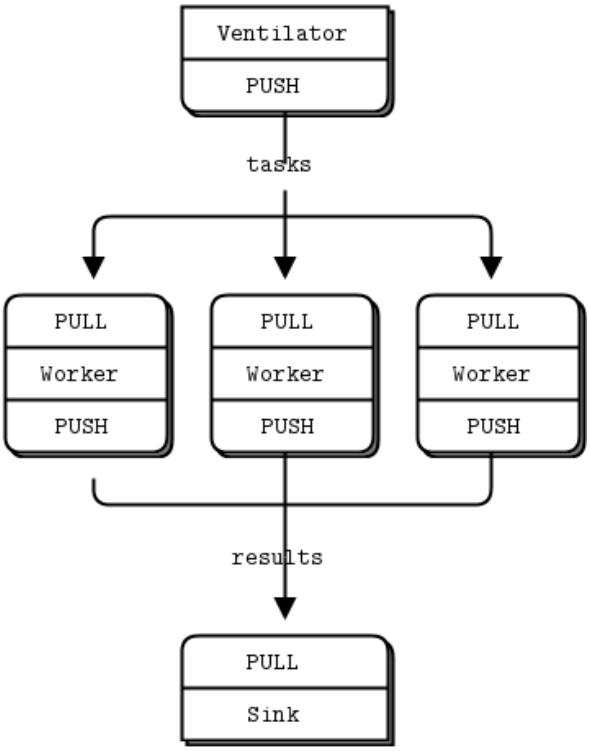


Σχήμα 4: Το πρότυπο publish–subscribe

4.2.2.2 Το πρότυπο push–pull

Στο πρότυπο αυτό χρησιμοποιούνται `PUSH` και `PULL` socket. Τα `PUSH` socket προωθούν τα μηνύματα προς όλα τα συνδεδεμένα με αυτά `PULL` socket με λογική round-robin. Τα `PUSH` socket προωθούν στην εφαρμογή τα ειλημμένα μηνύματα από τα `PULL` socket με τη σειρά που τα έλαβαν. Σε περίπτωση που η ουρά ενός `PUSH` socket γεμίσει (αυτό συμβαίνει όταν γεμίσουν οι ουρές όλων των `PULL` socket που είναι συνδεδεμένα σε αυτό), τότε η αποστολή μηνυμάτων μπλοκάρει μέχρι να δημιουργηθεί κενή θέση στην ουρά. Τα `PUSH` socket μπορούν μόνο να στείλουν μηνύματα, ενώ τα `PULL` socket μόνο να λάβουν

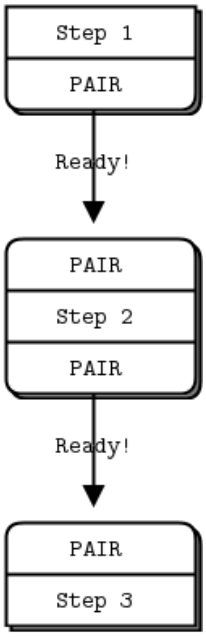
Το πρότυπο αυτό είναι χρήσιμο για την παραλληλοποίηση διεργασιών, αλλά και για τη δημιουργία pipeline.



Σχήμα 5: Το πρότυπο push–pull

4.2.2.3 Το πρότυπο pair

Το πρότυπο αυτό χρησιμοποιεί `PAIR` socket, τα οποία υποστηρίζουν αμφίδρομη ασύγχρονη επικοινωνία. Έχουν σχεδιαστεί αποκλειστικά και την επικοινωνία μεταξύ νημάτων (thread) ή διεργασιών (process) και είναι επίσης κατάλληλα για δημιουργία pipeline.



Σχήμα 6: Το πρότυπο pair

5 Marketplace για υπηρεσίες cloud

Τα τελευταία χρόνια οι υπηρεσίες cloud έχουν γίνει εξαιρετικά δημοφιλείς. Αυτό έχει ως αποτέλεσμα τη συνεχή ανάπτυξη και παροχή όλο και περισσότερων υπηρεσιών, που διαφέρουν μεταξύ τους σε πολλά επίπεδα, όπως το αν πρόκειται για IaaS, PaaS, SaaS, ή τη λειτουργικότητα που προσφέρουν. Το ίδιο ισχύει και για τις εταιρείες παροχής υπηρεσιών cloud, οι οποίες αυξάνονται ραγδαία, και, ακόμα και στην περίπτωση που παρέχουν παραπλήσιες υπηρεσίες με τις ανταγωνιστικές τους, το πλήθος και η πολυπλοκότητα των τιμολογιακών πολιτικών που εφαρμόζουν καθώς και το επίπεδο ποιότητας υπηρεσιών που προσφέρουν καθιστούν την επιλογή του καταλληλότερου πακέτου υπηρεσιών cloud ένα πολύ απαιτητικό πρόβλημα. Για την επίλυση αυτού του προβλήματος, αρχίζει συχνά να γίνεται όλο και πιο απαραίτητη η δημιουργία marketplace (ηλεκτρονικών καταστημάτων) για εύρεση και αγορά υπηρεσιών cloud. Τα marketplace αυτά θα αναζητούν μεταξύ των χιλιάδων διαθέσιμων υπηρεσιών και θα πρέπει να είναι σε θέση να διακρίνουν ποιες από αυτές είναι συμβατές τόσο με τις απαιτήσεις του χρήστη όσο και μεταξύ τους. Στη συνέχεια, είναι απαραίτητο να έχουν έναν «έξυπνο» τρόπο κατάταξης των υπηρεσιών ή των πακέτων υπηρεσιών βάσει του τι είναι πιο συμφέρον (τόσο από οικονομικής όσο και από τεχνικής άποψης) για τον χρήστη.

5.1 Τα στάδια ανάλυσης των απαιτήσεων του χρήστη και αξιολόγησης των αποτελεσμάτων

Για να μπορέσει το ηλεκτρονικό κατάστημα να αναζητήσει κατάλληλες προσφορές για τον χρήστη, ο τελευταίος πρέπει να δώσει μια σειρά από απαιτήσεις που να περιγράφουν τα χαρακτηριστικά του επιθυμητού συστήματος τόσο σε τεχνικό, επιχειρηματικό και τιμολογιακό επίπεδο. Το marketplace αναζητά υπηρεσίες και πακέτα υπηρεσιών με βάση τις τεχνικές προδιαγραφές, στη συνέχεια βρίσκει τις επιχειρηματικές προδιαγραφές των επιλεγμένων υπηρεσιών από όλους τους διαθέσιμους παρόχους, τις τιμολογεί και εν τέλει τις κατατάσσει σύμφωνα με το πόσο ταιριάζουν στις ανάγκες του χρήστη, πριν του τις παρουσιάσει.

5.1.1 Ανάλυση τεχνικών χαρακτηριστικών

Κατ' αρχάς, το κατάστημα θα πρέπει να μελετήσει τις απαιτήσεις που εισήγαγε ο χρήστης και να ανατρέξει στη βάση δεδομένων, προκειμένου να βρει υπηρεσίες που ικανοποιούν τις απαιτήσεις αυτές. Εν συνεχεία, θα πρέπει να εξετάσει όλους τους δυνατούς (και συμβατούς) συνδυασμούς, ούτως ώστε να κατασκευάσει όλα τα τεχνικώς εφικτά πακέτα προσφορών που συμμορφώνονται με τις απαιτήσεις του χρήστη.

Αναλόγως του τύπου της παρεχόμενης υπηρεσίας (IaaS, PaaS, SaaS) αλλά και του τι ακριβώς παρέχεται, υπάρχει πλήθος παραμέτρων που πρέπει να ληφθούν υπ' όψιν. Παραδείγματος χάριν, ένας χρήστης που αναζητά υπηρεσία για αποθήκευση και διαχείριση δεδομένων σε επίπεδο PaaS, έχει να διαλέξει μεταξύ λειτουργικού συστήματος (τύπος και έκδοση), αρχιτεκτονικής συστήματος (x84 ή x64), τύπου βάσεων δεδομένων (σχεσιακή, NoSQL κλπ.), ταχύτητας δικτύου και πολλά άλλα. Αν η παρεχόμενη υπηρεσία είναι σε επίπεδο SaaS, τότε θα πρέπει να ελεγχθούν και συμβατότητες μεταξύ των υπηρεσιών που θα επιλεγούν τόσο με τις

προδιαγραφές που έθεσε ο χρήστης, όσο και μεταξύ τους. Σε αυτό το στάδιο λοιπόν συλλέγονται όλες οι υπηρεσίες που πληρούν τις απαιτήσεις του χρήστη και –αναλόγως της μεταξύ τους συμβατότητας– συνδυάζονται προς σχηματισμό ολοκληρωμένων και λειτουργικών πακέτων τεχνικών προδιαγραφών.

5.1.2 Ανάλυση των επιχειρηματικών απαιτήσεων

Σε αυτό το βήμα το κατάστημα θα αναζητήσει κάθε υπηρεσία που περιέχεται στα πακέτα προσφοράς του προηγούμενου βήματος στη βάση δεδομένων, προκειμένου να βρει παρόχους για τις υπηρεσίες αυτές. Οι πάροχοι και οι υπηρεσίες τους οφείλουν να πληρούν τις επιχειρηματικές απαιτήσεις του χρήστη.

Για παράδειγμα, αν ο χρήστης ψάχνει για έναν εξυπηρετητή φιλοξενίας δυναμικών ιστοσελίδων, μπορεί να έχει αποκλείσει μοιραζόμενους εξυπηρετητές (shared server), να απαιτεί συγκεκριμένη αποδεδειγμένη διαθεσιμότητα πάνω από ορισμένα επίπεδα, ή να αποκλείσει ορισμένες γεωγραφικές περιοχές λόγω κρατικής νομοθεσίας περί προστασίας του απορρήτου των δεδομένων. Για κάθε πακέτο τεχνικών προδιαγραφών από το προηγούμενο στάδιο θα δημιουργηθούν συνδυασμοί παρόχων που διαθέτουν τις αιτούμενες υπηρεσίες.

5.1.3 Τιμολόγηση των προτάσεων

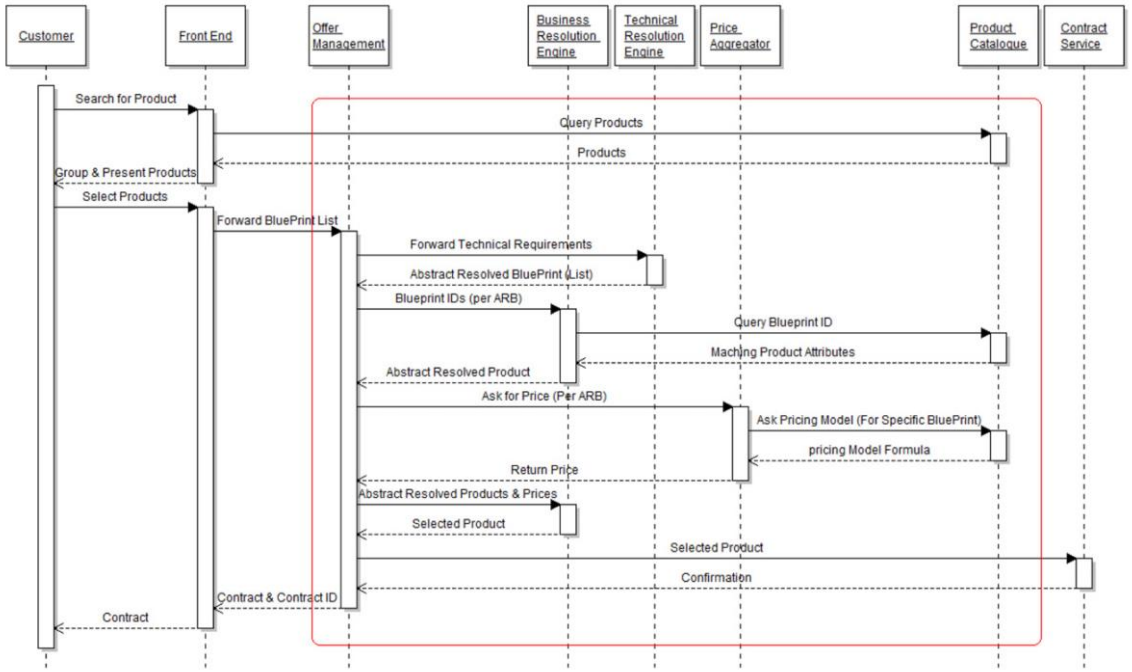
Για κάθε συνδυασμό που δημιουργείται από το προηγούμενο βήμα θα πρέπει τώρα να αναζητηθούν οι διαθέσιμες τιμολογιακές πολιτικές. Κάθε πάροχος μπορεί να χρεώνει τις υπηρεσίες του με περισσότερους από έναν τρόπους, όπως με προοδευτική χρέωση αναλόγως της χρήσης, διαφορετικές τιμές αναλόγως την ώρα (ημέρα, νύχτα), ή –σπανιότερα– με προκαθορισμένη χρέωση ανεξαρτήτως χρήσης. Σε αυτό το στάδιο επίσης θα απορριφθούν τυχόν τιμολογιακές πολιτικές που δεν συμμορφώνονται με τις απαιτήσεις του χρήστη ή που υπερβαίνουν το μέγιστο κόστος που έχει θέσει.

5.1.4 Κατάταξη των προτάσεων και επιλογή της καλύτερης

Οι τιμολογημένες πλέον προτάσεις θα πρέπει να αξιολογηθούν με βάση το ποια ταιριάζει περισσότερο στις απαιτήσεις του χρήστη, ποια έχει την καλύτερη εσωτερική συμβατότητα μεταξύ των υπηρεσιών της και, βεβαίως, της τιμής της. Οι καλύτερες εξ αυτών θα σταλούν πίσω στο χρήστη προκειμένου να επιλέξει.

5.2 Σχετική εργασία

Ένα εξελιγμένο marketplace που έχει προταθεί για να καλύψει τις ανάγκες που αναλύσαμε παραπάνω είναι το 4CaaSt. Είναι σχεδιασμένο ώστε να αναζητά υπηρεσίες cloud και των τριών τύπων (IaaS, PaaS και SaaS) και να επεξεργάζεται σενάρια για διαφόρων ειδών πολιτικές τιμολόγησης σύμφωνα με τις προδιαγραφές που δίνει ο χρήστης, προκειμένου να επιλέξει τους βέλτιστους συνδυασμούς.



Σχήμα 7: Το marketplace για συστήματα cloud 4CaaSt

Όπως φαίνεται στο σχήμα, υλοποιούνται στο 4CaaSt marketplace όλα τα προαναφερθέντα στάδια για την αναζήτηση, αξιολόγηση και κατάταξη των υπηρεσιών. Αναλυτικότερα, ο χρήστης αρχικά καλείται από το `Front End`, το οποίο επικοινωνεί με τον `Product Catalogue`, το component δηλαδή που είναι υπεύθυνο για την αναζήτηση των πληροφοριών στη βάση δεδομένων, να επιλέξει τις κατηγορίες προϊόντων που τον ενδιαφέρουν. Στη συνέχεια, με βάση την απάντηση του `Front End`, ο χρήστης συντάσσει τις τεχνικές, επιχειρηματικές και τιμολογιακές προδιαγραφές του απαιτούμενου συστήματος, οι οποίες προωθούνται στο component `Offer Management`, το οποίο και είναι επιφορτισμένο με τον έλεγχο και την επικοινωνία μεταξύ όλων των component που συνεργάζονται για την επίλυση του προβλήματος.

Το `Offer Management` με τη σειρά του στέλνει τις τεχνικές προδιαγραφές στο Technical Resolution Engine, το οποίο θα τις αναλύσει, όπως περιεγράφηκε στο πρώτο στάδιο του προηγούμενου υποκεφαλαίου. Στη συνέχεια θα στείλει τα αποτελέσματα πίσω στο `Offer Management`, το οποίο τα προωθεί μαζί με τις επιχειρηματικές προδιαγραφές στο `Business Resolution Engine`. Το τελευταίο θα αναζητήσει τις υπηρεσίες που πληρούν τα επιθυμητά κριτήρια στο `Product Catalogue` (δεύτερο στάδιο) και θα επιστρέψει τα αποτελέσματα. Σειρά έχει η αναζήτηση και ο συνδυασμός των τιμολογιακών πολιτικών για κάθε υπηρεσία του πακέτου (τρίτο στάδιο), την οποία εκτελεί το `Price Aggregator Engine`, υποβάλλοντας ερωτήματα στο `Product Catalogue`. Στη συνέχεια, το `Business Resolution Engine` θα κατατάξει τις λύσεις που συντάχθηκαν κατά τα προηγούμενα στάδια και θα επιλέξει αυτή που είναι η πιο συμφέρουσα για το χρήστη, τιμολογιακά, τεχνικά και επιχειρηματικά (τέταρτο στάδιο).

Τέλος, το `Offer Management` αναθέτει στο `Contract Service` να «κλείσει» τη συμφωνία για την επικρατέστερη υπηρεσία ή πακέτο υπηρεσιών στον πιο συμφέροντα πάροχο.

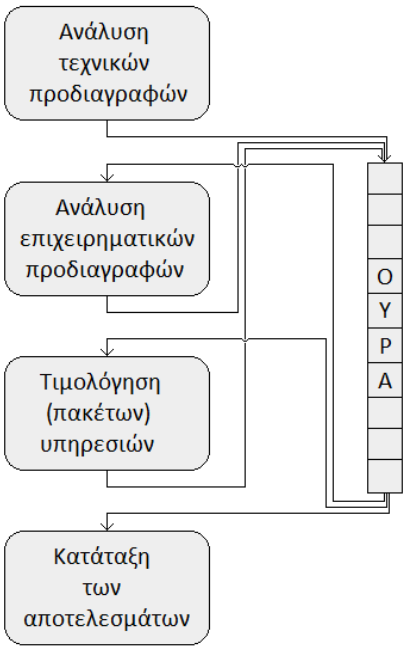
5.3 Η επικοινωνία μεταξύ των σταδίων

Λόγω του μεγάλου όγκου των δεδομένων, αλλά και της υψηλής πολυπλοκότητας της διαδικασίας ανάλυσης, που μεταφράζεται σε αυξημένο χρόνο, ιδιαίτερο ενδιαφέρον παρουσιάζει ο τρόπος με τον οποίο προωθούνται τα αποτελέσματα από κάθε στάδιο στο επόμενο. Σε αυτό πρέπει να ληφθεί υπόψιν ότι ένα τέτοιας έκτασης marketplace είναι πολύ πιθανόν να έχει εγκατασταθεί σε περιβάλλον cloud, το οποίο μάλιστα μπορεί να είναι και κατανεμημένο. Σε αυτή την περίπτωση τα δεδομένα θα πρέπει να μεταδίδονται μέσω διαδικτύου, το οποίο γεννά ανάγκη για υψηλή ταχύτητα σύνδεσης και αυξάνει ακόμα περισσότερο τη διάρκεια επίλυσης του προβλήματος.

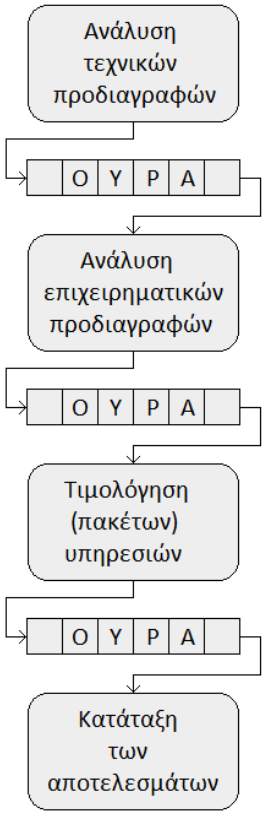
Παρατηρούμε ότι στο 4CaaSt κάθε component στέλνει τα αποτελέσματά του όλα μαζί στο τέλος, αφού ολοκληρώσει τόσο την αναζήτησή τους, όσο και την επεξεργασία τους. Αυτό δημιουργεί επιπλέον καθυστέρηση για δύο λόγους. Πρώτον επειδή το επόμενο component θα μπορούσε να έχει αρχίσει να επεξεργάζεται τα αποτελέσματα του προηγούμενου που είναι ήδη έτοιμα, χωρίς να χρειάζεται να το περιμένει να ολοκληρώσει την επεξεργασία. Δεύτερον, στην περίπτωση που το σύστημα είναι κατανεμημένο, η μετάδοση των δεδομένων απαιτεί αρκετό χρόνο, μέρος του οποίου θα μπορούσε να είχε εξοικονομηθεί εφ’ όσον η αποστολή κάθε αποτελέσματος άρχιζε αμέσως.

Προκειμένου να αντιμετωπισθεί αυτό το θέμα και να επιτευχθεί παραλληλισμός στην επίλυση του προβλήματος, θα χρησιμοποιήσουμε ουρές μηνυμάτων (message queue) για την αποστολή των αποτελεσμάτων από το ένα στάδιο στο άλλο, μπορούν να χρησιμοποιηθούν ουρές μηνυμάτων (message queue). Οι ουρές θα αναλαμβάνουν το έργο της προσωρινής αποθήκευσης των δεδομένων και την αποστολή τους από το ένα στάδιο στο άλλο, εξασφαλίζοντας έτσι την απρόσκοπτη λειτουργία των υπόλοιπων σταδίων όταν κάποιο δεν μπορεί να δεχθεί μηνύματα λόγω φόρτου.

Υπάρχουν δύο τρόποι με τους οποίους μπορεί να υλοποιηθεί το σύστημα επικοινωνίας μεταξύ των σταδίων με χρήση ουρών μηνυμάτων: με μία, κοινή ουρά, η οποία θα δέχεται τα μηνύματα από τα διάφορα στάδια και θα τα προωθεί αναλόγως, ή με ξεχωριστές ουρές, μέσω τον οποίων κάθε στάδιο θα προωθεί στο επόμενο τα μηνύματά του απευθείας. Στην παρούσα διπλωματική εργασία θα μελετηθούν συγκριτικά οι δύο αυτές τεχνικές και θα συγκριθούν επίσης με τον αρχικό τρόπο υλοποίησης, που προέβλεπε μαζική αποστολή των αποτελεσμάτων κατά την περάτωση κάθε σταδίου. Κατ’ αυτό τον τρόπο θα αποδείξουμε ότι η χρήση ουρών μηνυμάτων βελτιώνει την απόδοση του marketplace, αλλά και θα διαπιστώσουμε ποια από τις δύο τεχνικές (της κοινής ουράς μηνυμάτων ή των ξεχωριστών) υπερτερεί και σε ποιες περιπτώσεις.



Σχήμα 8: Επικοινωνία μεταξύ των σταδίων με κοινή ουρά μηνυμάτων



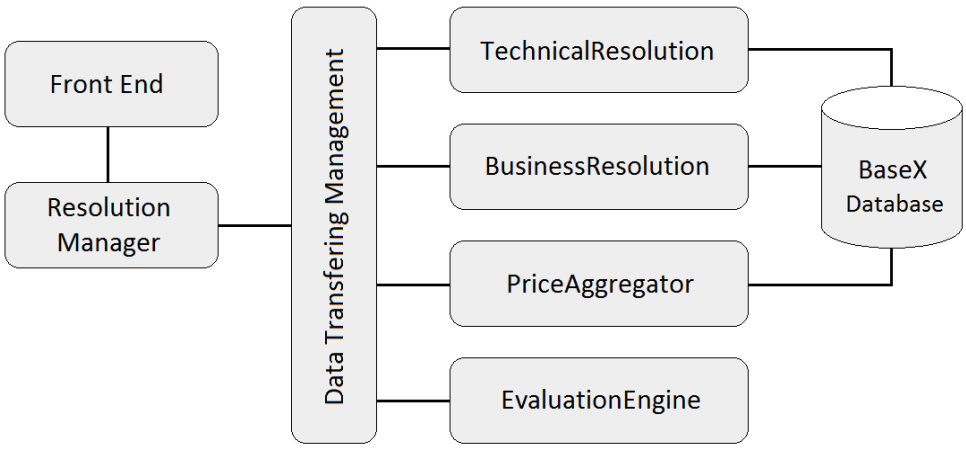
Σχήμα 9: Επικοινωνία μεταξύ των σταδίων με ξεχωριστές ουρές μηνυμάτων

6 Σχεδιασμός ενός marketplace για υπηρεσίες cloud

Στα πλαίσια της παρούσας διπλωματικής και προκειμένου να συγκρίνουμε την αποδοτικότητα του αρχικού τρόπου συνολικής μεταφοράς των δεδομένων μεταξύ των σταδίων στο τέλος κάθε φάσης επεξεργασίας και των υλοποιήσεων με ουρές μηνυμάτων, υλοποιήσαμε ένα απλοποιημένο μοντέλο του 4CaaSt, δίνοντας έμφαση στα τέσσερα στάδια επεξεργασίας των μηνυμάτων. Η ανάπτυξη έγινε στη γλώσσα προγραμματισμού C#, οι προδιαγραφές και τα αποτελέσματα αποθηκεύονται και μεταδίδονται ως αρχεία XML και ως βάση δεδομένων χρησιμοποιήθηκε η μη σχεσιακή βάση δεδομένων BaseX.

6.1 Η αρχιτεκτονική του συστήματος

Το σύστημα αποτελείται από ξεχωριστά components, ούτως ώστε να προσομοιάζει στις κατανεμημένες αρχιτεκτονικές των συστημάτων cloud. Ο `ResolutionManager` είναι η κλάση που φέρει την ευθύνη του συντονισμού της όλης διαδικασίας. Ο χρήστης μπορεί να εισάγει τις επιθυμητές προδιαγραφές σε αρχεία XML σύμφωνα με το σχήμα που θα αναλυθεί παρακάτω. Η κλάση `DataTransferingManagement` είναι υπεύθυνη για τη μεταφορά των μηνυμάτων μεταξύ των τεσσάρων κλάσεων που υλοποιούν τα στάδια που έχουμε προαναφέρει. Επίσης αναλαμβάνει την προώθηση των απαιτήσεων του χρήστη από τον `ResolutionManager` και την επιστροφή σε αυτόν των αποτελεσμάτων από την κλάση `EvaluationEngine`. Καθεμία από τις κλάσεις `TechnicalResolution`, `BusinessResolution`, `PriceAggregator` και `EvaluationEngine` υλοποιεί τα αντίστοιχα στάδια της επίλυσης του προβλήματος. Οι τρεις πρώτες από αυτές επικοινωνούν απευθείας με τη βάση δεδομένων BaseX προκειμένου να λαμβάνουν τις απαραίτητες πληροφορίες που χρειάζονται κάθε φορά για την επεξεργασία των δεδομένων τους.



Σχήμα 10: Η αρχιτεκτονική του marketplace

6.2 Οι παρεχόμενες υπηρεσίες

Για τις ανάγκες τα παροχής δεδομένων προς επεξεργασία στα component του marketplace, δημιουργήθηκαν στοιχειώδη μοντέλα υπηρεσιών cloud, τα οποία καταχωρούνται σε μορφή XML. Οι υπηρεσίες που μοντελοποιήθηκαν και παρέχονται είναι: λειτουργικό σύστημα, βάση δεδομένων, σύστημα διαχείρισης πελατειακών σχέσεων (CRM – Customer Relations Management) και δίκτυο. Κάθε μοντέλο έχει δύο ομάδες χαρακτηριστικών, μία που αφορά στις τεχνικές του προδιαγραφές και μία στις επιχειρηματικές και την τιμολόγηση μαζί.

Παρακάτω παρουσιάζονται τα τεχνικά χαρακτηριστικά που αφορούν κάθε παρεχόμενη υπηρεσία:

- Λειτουργικό σύστημα: Το λειτουργικό σύστημα περιλαμβάνει στα τεχνικά χαρακτηριστικά του τον *τύπο* του (Windows, Linux, Mac OS), την *αρχιτεκτονική* του (x86 ή x64) και τη *διανομή* του (για παράδειγμα Windows Server 2008 r2).
- Βάση δεδομένων: Οι τεχνικές προδιαγραφές των παρεχόμενων βάσεων δεδομένων αφορούν στο *όνομα* της βάσης (για παράδειγμα MySQL), την *αρχιτεκτονική* της (x86 ή x64) καθώς και τον *τύπο* της (σχεσιακή, χωρική κλπ.).
- Σύστημα διαχείρισης πελατειακών σχέσεων (CRM): Εδώ το μοντέλο μας για τα τεχνικά χαρακτηριστικά περιλαμβάνει το *όνομα* και την *αρχιτεκτονική* (x86 ή x64).
- Δίκτυο: Ομοίως και στην περίπτωση του δικτύου, στα τεχνικά χαρακτηριστικά εντάσσεται το *όνομα* της παρεχόμενης υπηρεσίας (VDSL Connection, T3 Connection) και ο τύπος του *φυσικού μέσου* (καλώδιο, οπτική ίνα κ.λπ.).

Τα επιχειρηματικά και τιμολογιακά χαρακτηριστικά είναι πολλαπλά για κάθε παρεχόμενη υπηρεσία και διαφέρουν για κάθε πάροχο. Για λόγους απλοποίησης του προβλήματος είναι κοινά και για τις τέσσερεις παρεχόμενες υπηρεσίες είναι τα εξής:

- το όνομα του παρόχου·
- η διαθεσιμότητα της υπηρεσίας, εκφραζόμενη ως ποσοστό επί τοις εκατό του χρόνου που η υπηρεσία είναι εγγυημένα διαθέσιμη·
- η ασφάλεια, βαθμολογούμενη από 1 έως 5·
- η φήμη, βαθμολογούμενη από 1 έως 5·
- η αξιοπιστία, βαθμολογούμενη από 1 έως 5·
- η εμπιστευτικότητα, η οποία είτε παρέχεται είτε όχι, και
- η τιμή, το μοναδικό χαρακτηριστικό που αφορά την τιμολογιακή ανάλυση του προβλήματος.

6.3 Η βάση δεδομένων

Για της ανάγκες της αποθήκευσης των χαρακτηριστικών που αφορούν στις παρεχόμενες υπηρεσίες χρησιμοποιήθηκε η μη σχεσιακή βάση δεδομένων BaseX. Η BaseX είναι μία βάση δεδομένων XML, η οποία υποστηρίζει αναζήτηση με χρήση XPath και XQuery (εκφράσεις FLOWR).

Κάθε καταχώρηση τεχνικών χαρακτηριστικών αποθηκεύεται στη βάση δεδομένων ως XML element. Το ίδιο ισχύει και για κάθε καταχώρηση επιχειρηματικών-τιμολογιακών

χαρακτηριστικών. Δεδομένου ότι για κάθε υπηρεσία υπάρχει μόνο μία καταχώρηση τεχνικών χαρακτηριστικών, αλλά τουλάχιστον μία επιχειρηματικών-τιμολογιακών (επειδή κάθε υπηρεσία μπορεί να παρέχεται από πολλούς διαφορετικούς παρόχους), χρειάζεται και ένα αναγνωριστικό (ID), με βάση το οποίο θα γνωρίζει το marketplace τις αντιστοιχίες μεταξύ τους. Το σύνολο των τεχνικών προδιαγραφών καταχωρείται στη βάση σε ένα αρχείο XML και το σύνολο των επιχειρηματικών-τιμολογιακών σε ένα δεύτερο. Παρακάτω αντιστοιχούν παραδείγματα τεχνικών προδιαγραφών για κάθε υπηρεσία και ένα παράδειγμα επιχειρηματικών προδιαγραφών και κόστους, που είναι ίδιες για όλες τις υπηρεσίες:

- Λειτουργικό σύστημα:

```
<OperatingSystem>
  <ID>2002003201</ID>
  <Type>Linux</Type>
  <Distribution>Ubuntu 12.04.5 LTS</Distribution>
  <OP>x86</OP>
</OperatingSystem>
```

- Βάση δεδομένων:

```
<Database>
  <ID>1002003230</ID>
  <Name>Neo4J</Name>
  <OP>x86</OP>
  <Type>NoSQL</Type>
</Database>
```

- Σύστημα διαχείρισης πελατειακών σχέσεων (CRM):

```
<CRM>
  <ID>3002003201</ID>
  <Name>Drupal 7</Name>
  <OP>x86</OP>
</CRM>
```

- Δίκτυο:

```
<Network>
  <ID>4002003203</ID>
  <Name>VDSL Connection</Name>
  <Type>fiber</Type>
</Network>
```

- Επιχειρηματικά-τιμολογιακά χαρακτηριστικά:

```
<POperatingSystem>
  <ID>2002003209</ID>
  <ServiceProvider>VMWare</ServiceProvider>
  <Availability>78</Availability>
  <Security>2</Security>
  <Reputation>4</Reputation>
  <Reliability>2</Reliability>
  <Confidentiality>YES</Confidentiality>
```

```
        <Price>3300</Price>
      </POperatingSystem>
```

6.4 Η μορφή των αρχείων XML των προδιαγραφών του χρήστη

Ο χρήστης πρέπει να εισάγει στο σύστημα ένα αρχείο XML, στο οποίο θα περιγράφει τις τεχνικές, επιχειρηματικές και τιμολογιακές του απαιτήσεις. Για τις ανάγκες της παρούσας διπλωματικής, όπως είδαμε, ο χρήστης μπορεί να επιλέξει μεταξύ των παρεχόμενων υπηρεσιών, που είναι: λειτουργικό σύστημα, βάση δεδομένων, σύστημα διαχείρισης πελατειακών σχέσεων και δίκτυο. Σε αυτά μπορεί να δώσει συγκεκριμένες επιθυμητές προδιαγραφές για τα τεχνικά τους χαρακτηριστικά, εφ' όσον επιθυμεί. Ως προς τις επιχειρηματικές προδιαγραφές, μπορεί να ορίσει την ελάχιστη διαθεσιμότητα που απαιτείται για το σύστημά του (`MinAvailability`), καθώς και το ελάχιστο επίπεδο ασφάλειας (`MinSecurity`), αξιοπιστίας (`MinReliability`) και φήμης (`MinReputation`) που απαιτεί για τις υπηρεσίες που θα του παρασχεθούν. Μπορεί ακόμα να επιλέξει εάν θέλει οι υπηρεσίες να παρέχονται εμπιστευτικά (`Confidentiality`). Τέλος, μπορεί να θέσει το μέγιστο χρηματικό ποσόν που είναι διατεθειμένος να πληρώσει για το πακέτο των υπηρεσιών (`MaxPrice`). Ακολουθεί ένα παράδειγμα τέτοιου αρχείου:

```
<?xml version="1.0" encoding="UTF-8"?>
<XML>
  <UserRequirements>
    <Optimization>
      <MinAvailability>95</MinAvailability>
      <MinSecurity>2</MinSecurity>
      <MinReputation>4</MinReputation>
      <MinReliability>3</MinReliability>
      <Confidentiality>YES</Confidentiality>
      <MaxPrice>30000</MaxPrice>
    </Optimization>
    <Service>
      <Database>
        <OP>x86</OP>
        <Type>SQL</Type>
      </Database>
      <OperatingSystem>
        <Type>Linux</Type>
        <OP>x86</OP>
        <Distribution>Ubuntu 12.04.5 LTS</Distribution>
      </OperatingSystem>
      <CRM>
        <OP>x86</OP>
      </CRM>
      <Network>
        <Type>copper</Type>
      </Network>
    </Service>
  </UserRequirements>
  <Technical>
  </Technical>
  <Business>
  </Business>
```

```
  <TotalPrice>
  </TotalPrice>
  <Evaluation>
  </Evaluation>
</XML>
```

Τα element `Technical`, `Business`, `TotalPrice` και `Evaluation` του αρχείου αυτού θα συμπληρωθούν από τις αντίστοιχες κλάσεις, όπως θα δούμε παρακάτω, στην ανάλυση των σταδίων επίλυσης.

6.5 Τα στάδια επίλυσης του προβλήματος

Όπως και στο marketplace 4CaaSt, έτσι και εδώ η επίλυση του προβλήματος γίνεται σε τέσσερα στάδια. Ο `ResolutionManager` λαμβάνει από το `FrontEnd` το αρχείο XML με τις προδιαγραφές των απαιτήσεων του χρήστη και το προωθεί στο πρώτο στάδιο, δηλαδή στον `TechnicalResolution`. Όλες οι λύσεις του `TechnicalResolution` προωθούνται στον `BusinessResolution` κ.ο.κ., έως ότου η `EvaluationEngine` να επιστρέψει το τελικό αποτέλεσμα στον `ResolutionManager`. Ο τελευταίος θα προωθήσει την απάντηση στον χρήστη.

6.5.1 Ο καθορισμός των τεχνικών παραμέτρων

Ο `TechnicalResolution` θα λάβει το αρχείο XML της εισόδου του χρήστη και θα αναζητήσει στις καταχωρήσεις τεχνικών προδιαγραφών της βάσης δεδομένων όλες τις συμβατές με τα τεχνικά χαρακτηριστικά που δήλωσε ο χρήστης υπηρεσίες. Στη συνέχεια, για κάθε δυνατό συνδυασμό μεταξύ αυτόν τον υπηρεσιών θα δημιουργήσει από ένα νέο XML αρχείο, αντίγραφο της εισόδου του χρήστη, και θα προσθέσει στο element `Technical` το συνδυασμό αυτόν. Στη συνέχεια θα προωθήσει όλα αυτά τα μηνύματα προς το επόμενο στάδιο.

6.5.2 Ο καθορισμός των επιχειρηματικών παραμέτρων

Ο `BusinessResolution` θα λάβει όλες τις λύσεις που συνέταξε ο `TechnicalResolution`. Για κάθε μία από αυτές θα αναζητήσει στη βάση δεδομένων όλες τις υπάρχουσες καταχωρήσεις που αφορούν τις υπηρεσίες που περιλαμβάνει και, όπως και στο προηγούμενο βήμα, θα φτιάξει ένα νέο αρχείο XML για κάθε δυνατό συνδυασμό, τον οποίο θα τοποθετήσει εντός του element `Business`. Τέλος, θα προωθήσει όλα αυτά τα αρχεία στο επόμενο στάδιο.

6.5.3 Η τιμολόγηση των πακέτων

Για κάθε αρχείο XML που λαμβάνει ο `PriceAggregator`, αθροίζει τις επί μέρους τιμές της κάθε υπηρεσίας. Βρίσκει επίσης και γράφει το άθροισμα εντός του element `TotalPrice`. Στη συνέχεια προωθεί τα αρχεία αυτά στο επόμενο στάδιο.

6.5.4 Η αξιολόγηση και κατάταξη των αποτελεσμάτων και η επιλογή της βέλτιστης λύσης

Τέλος, η `EvaluationEngine` λαμβάνει όλα τα τιμολογημένα πακέτα υπηρεσιών και αναλαμβάνει το έργο της κατάταξής τους. Για το σκοπό αυτό εξάγει τον μέσο όρο για κάθε ένα από τα χαρακτηριστικά της διαθεσιμότητας, ασφάλειας, αξιοπιστίας και φήμης. Επίσης, ως μέσος όρος του χαρακτηριστικού της αξιοπιστίας λογίζεται το άθροισμα όσων υπηρεσιών πληρούν αυτό το κριτήριο. Αφού υπολογιστούν οι προαναφερθέντες μέσοι όροι, τα πακέτα υπηρεσιών κατατάσσονται με βάση τον ακόλουθο τύπο:

$$200 \cdot avg(Availability) + 2.000 \cdot avg(Security) + 2.000 \cdot avg(Reputation) +$$
$$1.000 \cdot avg(Reliability) + 1.000 \cdot avg(Confidentiality) - TotalPrice$$

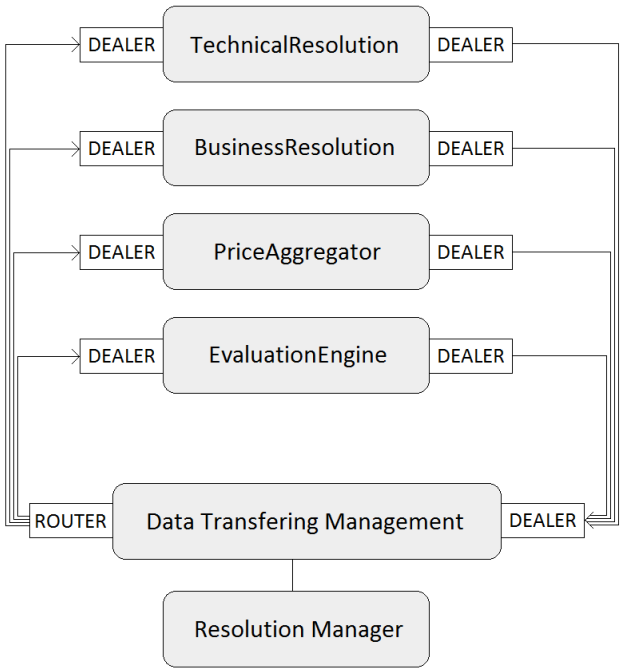
Όσο υψηλότερο είναι το σταθμισμένο αυτό άθροισμα, τόσο καλύτερο θεωρείται και το πακέτο υπηρεσιών. Όταν η `EvaluationEngine` λάβει και το τελευταίο τιμολογημένο πακέτο υπηρεσιών από τον `PriceAggregator`, τότε θα επιστρέψει το καλύτερο πακέτο στον `ResolutionManager`, προκειμένου να προωθηθεί στο `FrontEnd` και να το λάβει ο χρήστης.

6.6 Υλοποίηση της επικοινωνίας με κοινή ουρά μηνυμάτων

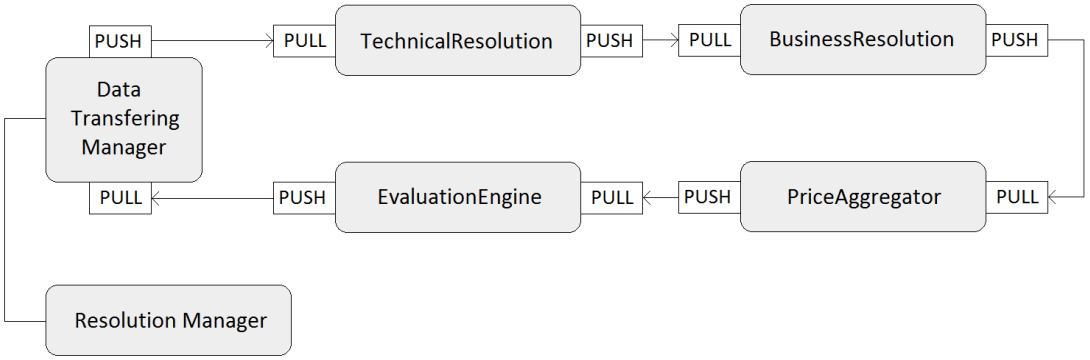
Για την υλοποίηση των ουρών μηνυμάτων χρησιμοποιήθηκε η ουρά ZeroMQ, που αναλύθηκε σε προηγούμενο κεφάλαιο. Καθεμία από τις κλάσεις που υλοποιούν τα τέσσερα στάδια της επεξεργασίας χρησιμοποιεί από ένα `DEALER` socket για την προώθηση των μηνυμάτων στο επόμενο στάδιο και από ένα `DEALER` socket για τη λήψη των μηνυμάτων από το προηγούμενο στάδιο. Η κλάση `DataTransferingManagement` χρησιμοποιεί ένα `DEALER` socket για τη λήψη των μηνυμάτων από όλες τις άλλες κλάσεις και ένα `ROUTER` socket για τη διανομή των πακέτων στη σωστή κλάση.

6.7 Υλοποίηση της επικοινωνίας με ξεχωριστές ουρές μηνυμάτων

Και εδώ χρησιμοποιείται η ZeroMQ. Κάθε κλάση διαθέτει ένα PULL και ένα PUSH socket, τα οποία όμως στην προκειμένη περίπτωση προωθούν τα πακέτα απευθείας στο επόμενο στάδιο μέσω ξεχωριστών ουρών. Δημιουργείται κατ' αυτόν τον τρόπο ένα pipeline κατά την επίλυση του προβλήματος.



Σχήμα 11: Η υλοποίηση με κοινή ουρά μηνυμάτων



Σχήμα 12: Η υλοποίηση με ξεχωριστές ουρές μηνυμάτων

6.8 Η παραμετροποίηση των πειραμάτων

Οι διαφορές μεταξύ ενός πραγματικού, λειτουργικού marketplace και της απλοποιημένης εκδοχής που υλοποιήθηκε δημιουργούν την ανάγκη ορισμένων μεταβλητών παραμέτρων, οι οποίες θα τροποποιούν τη συμπεριφορά των σταδίων επίλυσης του προβλήματος, ώστε να προσομοιάζει κατά το δυνατόν περισσότερο στις πραγματικές συνθήκες λειτουργίας.

Σε πραγματικές συνθήκες λειτουργίας το αντίστοιχο marketplace θα παρήγαγε πολύ μεγαλύτερο αριθμό μηνυμάτων που θα έπρεπε να σταλούν από το κάθε στάδιο στο επόμενο. Επίσης, η πολυπλοκότητα επεξεργασίας των προδιαγραφών σε κάθε στάδιο και ειδικά στο στάδιο της ανάλυσης των τιμολογιακών πολιτικών είναι στην πραγματικότητα πολύ μεγαλύτερη σε σχέση την απλουστευμένη υλοποίηση που έγινε στα πλαίσια της παρούσας διπλωματικής. Πέραν

τούτων, μας ενδιαφέρει να αξιολογήσουμε την αποδοτικότητα του αρχικού (χωρίς ουρές) σχεδίου και των δικών μας προτάσεων (με κοινή και ξεχωριστές ουρές) για διαφορετικούς φόρτους εργασίας σε κάθε στάδιο επεξεργασίας. Προκειμένου να μπορέσουμε να τροποποιήσουμε τα σενάρια που θα μελετήσουμε, αποφασίστηκε η εισαγωγή δύο επιπλέον παραμέτρων, που δεν σχετίζονται με την επίλυση του προβλήματος: του όγκου των αποστελλόμενων μηνυμάτων και το χρόνο επεξεργασίας για τη σύνταξη κάθε πακέτου προδιαγραφών.

6.8.1 Όγκος μηνυμάτων

Είναι απαραίτητο να μελετηθεί η συμπεριφορά του marketplace σε όλα τα σενάρια (χωρίς ουρά μηνυμάτων, με κοινή ουρά και με ξεχωριστές ουρές) σε διαφορετικό όγκο μηνυμάτων, ώστε να δούμε πώς αντιδρά σε κάθε περίπτωση. Για το σκοπό αυτό ο `TechnicalResolution` μπορεί να αυξομειώνει τον αριθμό των αποστελλόμενων μηνυμάτων, στέλνοντας πολλαπλά αντίγραφα κάθε μηνύματος εξόδου, με βάση τον συντελεστή που προσδιορίζει ο χρήστης.

6.8.2 Εισαγωγή τεχνητού φόρτου εργασίας

Καθοριστικό ρόλο στη μελέτη της συμπεριφοράς του marketplace και ειδικά στη συγκριτική μελέτη κοινής και ξεχωριστών ουρών μηνυμάτων διαδραματίζει ο χρόνος που απαιτείται από κάθε στάδιο για την ολοκλήρωση της επεξεργασίας των δεδομένων του. Προκειμένου λοιπόν να μεταβάλλουμε το χρόνο αυτό, πριν τη δημιουργία κάθε αρχείου εξόδου σε κάθε στάδιο, εκτελείται η δημιουργία, πλήρωση με τυχαίους ακεραίους και ο πολλαπλασιασμός δύο τετραγωνικών πινάκων. Τροποποιώντας το μέγεθος των πινάκων αυτών είμαστε σε θέση να αυξομειώνουμε τον χρόνο λειτουργίας του κάθε σταδίου.

7 Η υλοποίηση του marketplace

Για τη δημιουργία του απλοποιημένου marketplace για υπηρεσίες cloud εργαστήκαμε στην πλατφόρμα ανάπτυξης Visual Studio Ultimate 2013 της Microsoft. Χρησιμοποιήσαμε επίσης τη μη σχεσιακή XML βάση δεδομένων BaseX 8.3. Για της ουρές μηνυμάτων χρησιμοποιήθηκε η ουρά μηνυμάτων ZeroMQ 4.1.1 (ØMQ). Το λειτουργικό σύστημα του υπολογιστή που χρησιμοποιήθηκε για την ανάπτυξη και τις προσομοιώσεις είναι Windows 10 Pro N.

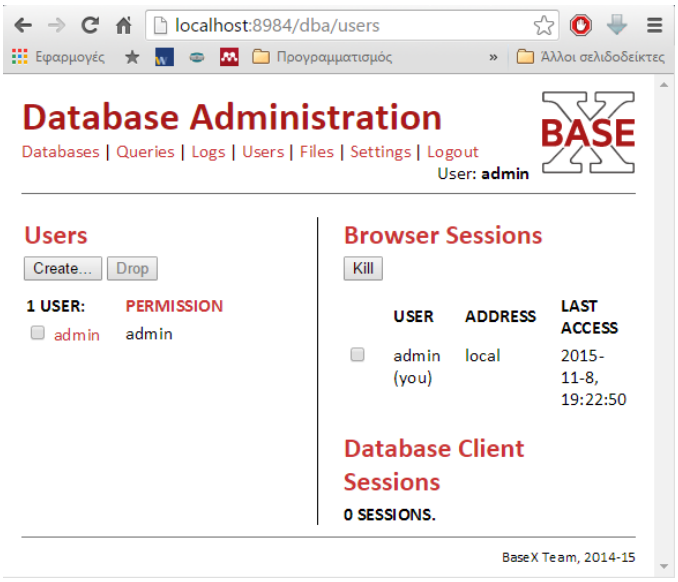
7.1 Η εγκατάσταση και παραμετροποίηση της βάσης BaseX

Ο εξυπηρετητής (server) και η γραφική διεπιφάνεια χρήστη (GUI – Graphical User Interface) της βάσης δεδομένων BaseX μπορούν να μεταφορτωθούν και να εγκατασταθούν χρησιμοποιώντας το αντίστοιχο εκτελέσιμο αρχείο (.exe) από την ιστοσελίδα http://basex.org/products/download/all-downloads/. Κατεβάσαμε και εκτελέσαμε τον installer για Windows, ο οποίος εγκατέστησε τη BaseX στον υπολογιστή.

Για την επικοινωνία των κλάσεων του project με τη βάση δεδομένων χρησιμοποιείται η υλοποίηση του πελάτη (client) σε C#· πρόκειται για δύο κλάσεις (τη `Session` και την `Query`), που βρίσκονται εντός του namespace `BaseXClient`. Ο κώδικας του client μπορεί να βρεθεί στο GitHub, στην ακόλουθη ιστοσελίδα: https://github.com/BaseXdb/basex/blob/master/basex-api/src/main/c%23/BaseXClient.cs.

Αξίζει να σημειωθεί εδώ ότι για να μπορεί να επικοινωνήσει ο κώδικας του πελάτη με τη βάση δεδομένων, πρέπει να εκκινήσουμε τον εξυπηρετητή της τελευταίας. Αυτό μπορεί να γίνει είτε μέσω της γραμμής εντολών, πληκτρολογώντας «`basexserver -S`»· έτσι ο εξυπηρετητής εκκινείται και «ακούει» στη θύρα 1984. Αν θέλουμε να εκκινήσουμε τον εξυπηρετητή σε συγκεκριμένη θύρα, π.χ. την 1234, μπορούμε να πληκτρολογήσουμε την εντολή «`basexserver -p 1234`». Για να διακόψουμε τη λειτουργία του server πληκτρολογούμε «`basexserver stop`». Εναλλακτικά, μπορούν να χρησιμοποιηθούν τα αρχεία δέσμης Windows (.bat) basexserver.bat και basexserverstop.bat, που βρίσκονται στο φάκελο bin, εντός του φακέλου εγκατάστασης της BaseX.

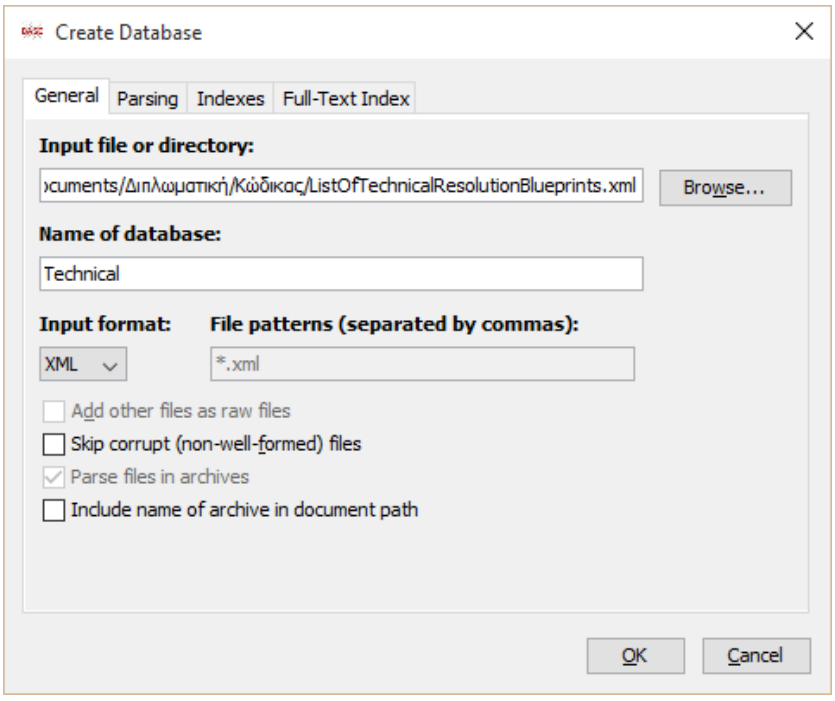
Για την επικοινωνία με τον εξυπηρετητή στα πλαίσια της παρούσας διπλωματικής χρησιμοποιείται ο λογαριασμός διαχειριστή `admin` (με κωδικό πρόσβασης «`admin`»). Σε περιβάλλον πραγματικής λειτουργίας, πρέπει οπωσδήποτε να δημιουργηθεί ένας νέος λογαριασμός με περιορισμένα δικαιώματα. Για να γίνει αυτό ο χρήστης πρέπει να χρησιμοποιήσει τη διεπιφάνεια web του εξυπηρετητή, πηγαίνοντας στη διεύθυνση http://localhost:8984/dba/. Στη συνέχεια επιλέγει Users από το μενού και δημιουργεί νέο λογαριασμό.



Σχήμα 13: Δημιουργία νέου λογαριαμού

7.2 Η δημιουργία των βάσεων δεδομένων

Για την εισαγωγή των χαρακτηριστικών που περιεγράφηκαν στο προηγούμενο κεφάλαιο, χρησιμοποιούμε το GUI της BaseX. Συγκεκριμένα, κάνοντας κλικ στο μενού Database→New, εισάγουμε το αρχείο XML που περιέχει τα τεχνικά χαρακτηριστικά, ονομάζουμε την νέα βάση δεδομένων και πατάμε OK. Επαναλαμβάνουμε το ίδιο βήμα και για την εισαγωγή του αρχείου XML με τα επιχειρηματικά και τιμολογιακά χαρακτηριστικά.

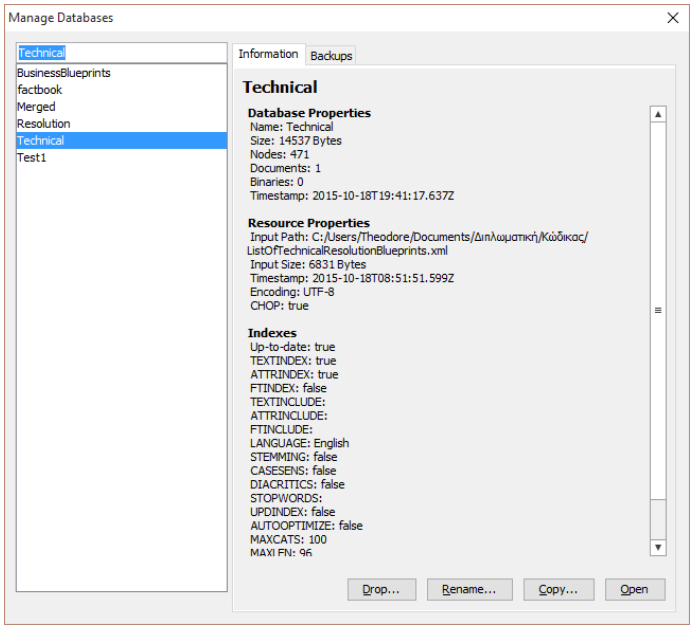


Σχήμα 14:Δημιουργία βάσης δεδομένων

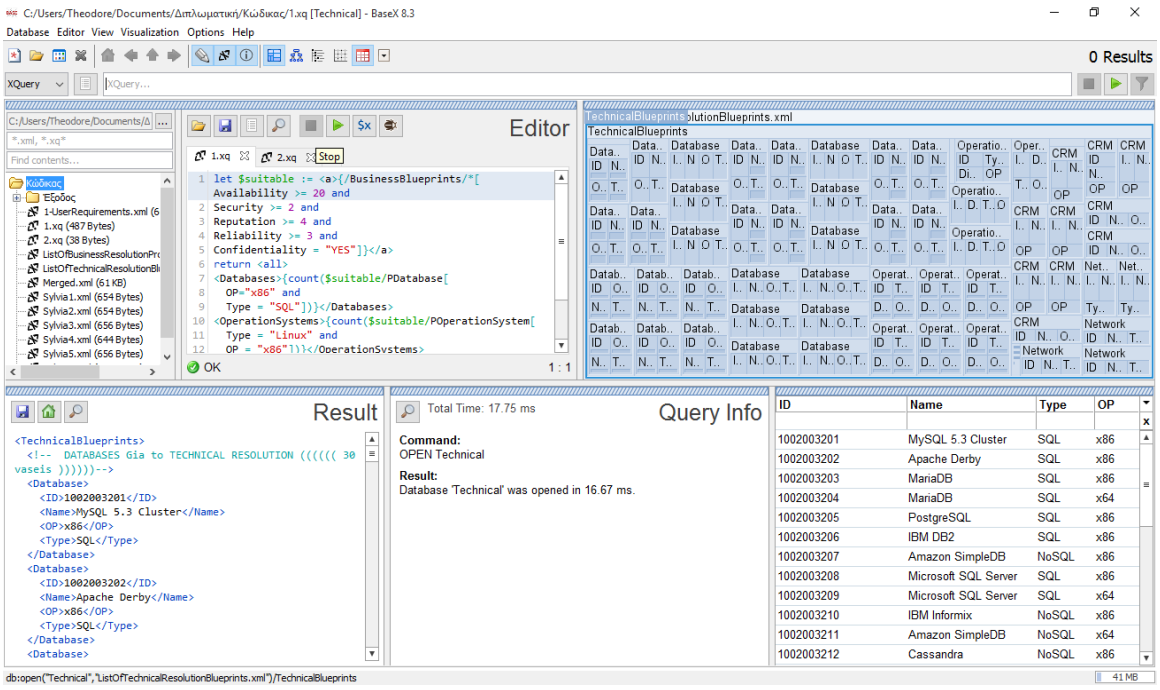
7.3 Η εκτέλεση ερωτημάτων (query) προς τη βάση δεδομένων BaseX

7.3.1 Μέσω της γραφικής διεπιφάνειας χρήστη (GUI)

Προκειμένου να πραγματοποιήσουμε query προς μία βάση δεδομένων, πρέπει κατ' αρχάς να την ανοίξουμε. Για να το κάνουμε αυτό κάνουμε κλικ στο μενού Database→Open & Manage, και από τη λίστα στα αριστερά επιλέγουμε την επιθυμητή βάση δεδομένων· στη συνέχεια κάνουμε κλικ στο OK. Στη συνέχεια, μπορούμε να πληκτρολογούμε και να εκτελούμε τα query μας από τον Editor. Η BaseX υποστηρίζει query με τη χρήση εκφράσεων XPath και FLOWR. Οι απαντήσεις στα εκτελούμενα query εμφανίζονται κάτω αριστερά, στο παράθυρο Results.



Σχήμα 15: Άνοιγμα βάσης δεδομένων



Σχήμα 16: Η γραφική διεπιφάνεια χρήστη της BaseX

7.3.2 Χρησιμοποιώντας τον BaseXClient

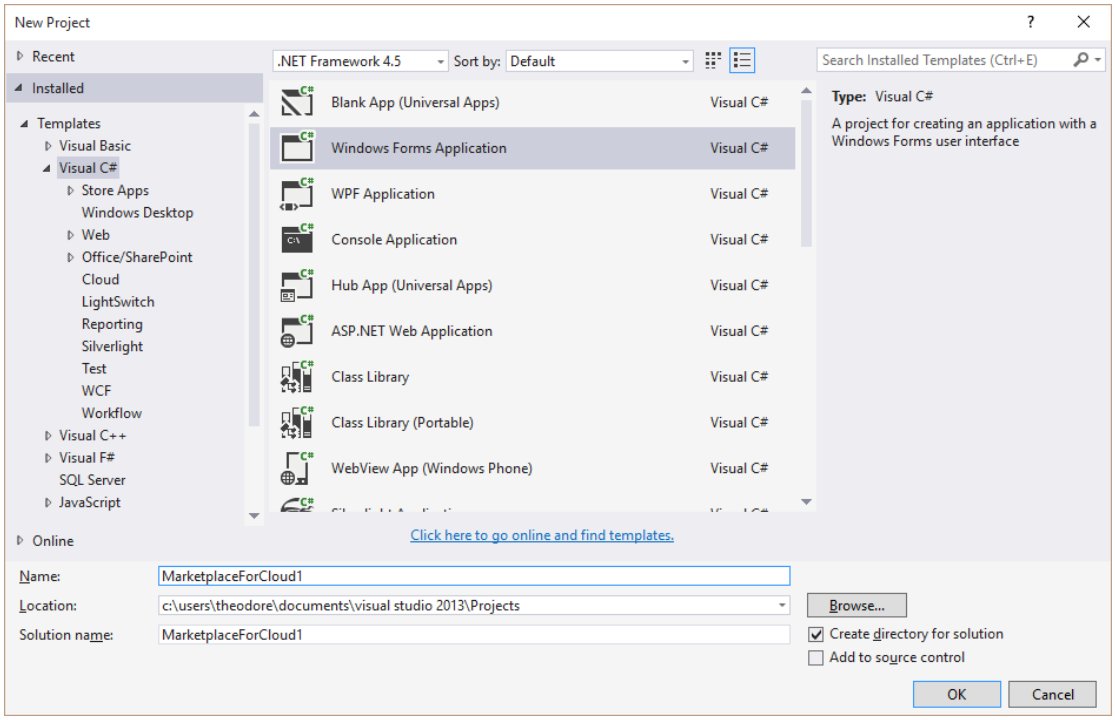
Όλα τα ερωτήματα που πραγματοποιούνται προς τον εξυπηρετητή της BaseX πρέπει να γίνονται στα πλαίσια ενός session. Στη συνέχεια ακολουθεί ένα παράδειγμα session, το οποίο συνδέεται στον εξυπηρετητή, ανοίγει τη βάση δεδομένων TechnicalBlueprints και επιστρέφει όλες τις καταχωρήσεις προϊόντων:

```
Session session = new Session("localhost", 1984, "admin", "admin");
session.Execute("open TechnicalBlueprints");
String result = session.Query("/TechnicalBlueprints/*").Execute();
session.Close();
```

Στο τέλος κάθε session είναι σημαντικό να εκτελείται η μέθοδος `Close()`, προκειμένου να απελευθερώνονται οι πόροι του συστήματος που έχουν δεσμευθεί.

7.4 Δημιουργία του project και προσθήκη των απαραίτητων βιβλιοθηκών και κλάσεων

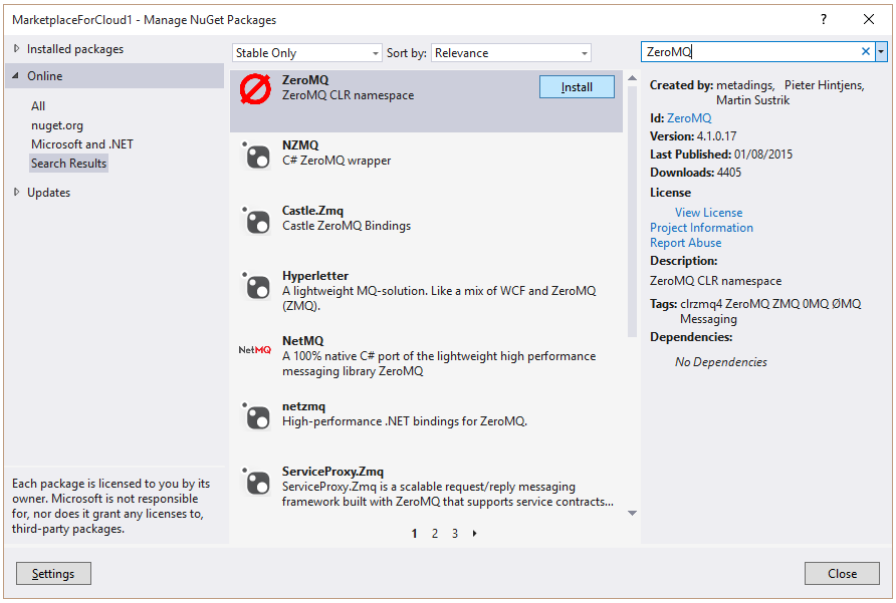
Για την υλοποίηση του marketplace δημιουργήσαμε ένα νέο project στο Visual Studio, τύπου Windows Forms Application.



Σχήμα 17: Δημιουργία νέου project

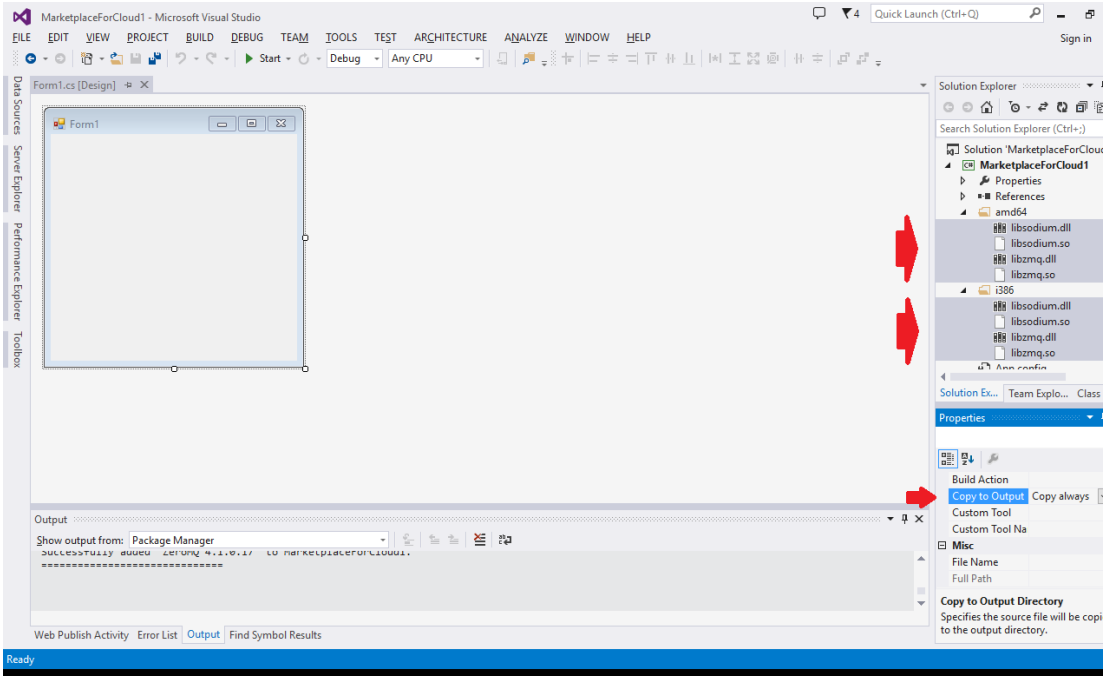
Προκειμένου να μπορούμε να χρησιμοποιήσουμε την ZeroMQ στο project μας, πρέπει να προσθέσουμε τις αντίστοιχες βιβλιοθήκες (`libsodium.dll` και `libzmq.dll`). Οι βιβλιοθήκες αυτές είναι διαθέσιμες στα πακέτα λογισμικού του NuGet. Για να τις προσθέσουμε πρέπει στο Solution Explorer να κάνουμε δεξί κλικ πάνω στο project μας και να επιλέξουμε

Manage NuGet Packages. Στη συνέχεια αναζητούμε το πακέτο ZeroMQ και το εγκαθιστούμε πατώντας Install.



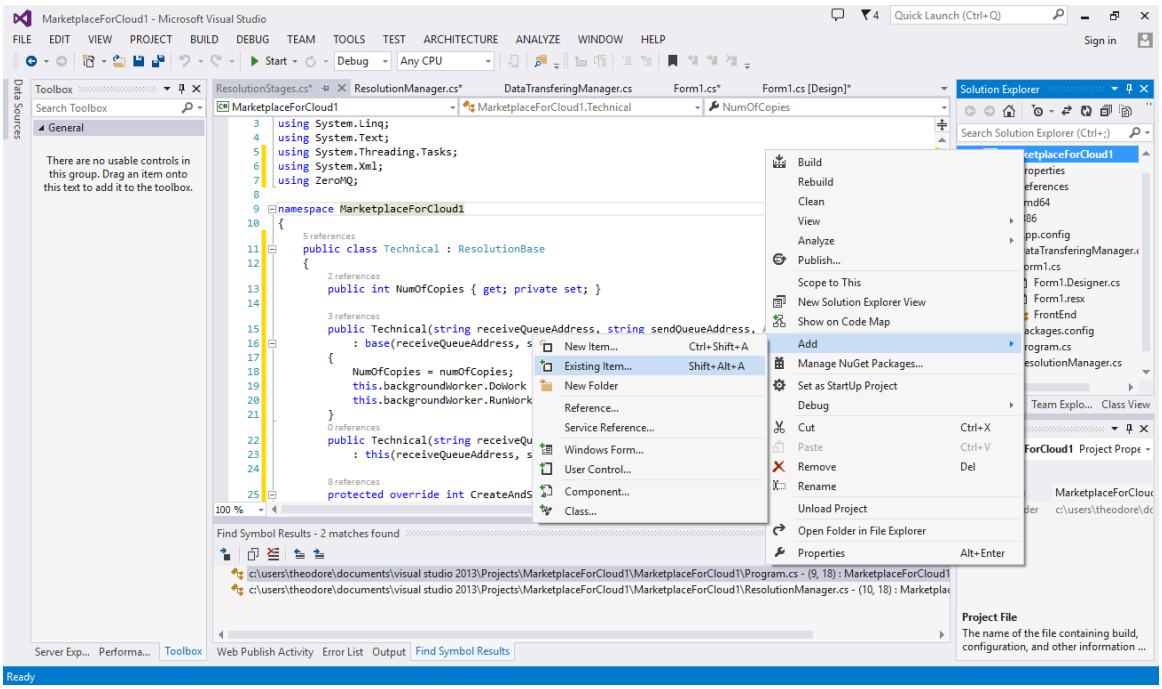
Σχήμα 18: Διαχείριση πακέτων NuGet

Ιδιαίτερη προσοχή απαιτείται στις ιδιότητες των βιβλιοθηκών που προσθέσαμε, καθώς αυτές πρέπει να αντιγράφονται στον φάκελο με το εκτελέσιμο αρχείο (.exe), όταν «χτίζουμε» το project. Η προεπιλεγμένη συμπεριφορά κατά το building της εφαρμογής είναι να μην αντιγράφονται, οπότε πρέπει να την τροποποιήσουμε, επιλέγοντας στο παράθυρο Properties και στο κελί «Copy to Output» την επιλογή «Copy always».



Σχήμα 19: Αντιγραφή των βιβλιοθηκών της ZeroMQ στο φάκελο με το εκτελέσιμο.

Για να μπορέσουμε να συνδεθούμε στο server της βάσης δεδομένων BaseX, όπως προαναφέρθηκε, πρέπει να χρησιμοποιήσουμε των κώδικα πελάτη (client), που βρίσκεται στο αρχείο BaseXClient.cs που αναφέρθηκε προηγουμένως. Αυτό γίνεται εντοπίζοντας το project μας στον Solution Explorer, κάνοντας δεξί κλικ επάνω του → Add → Existing item.

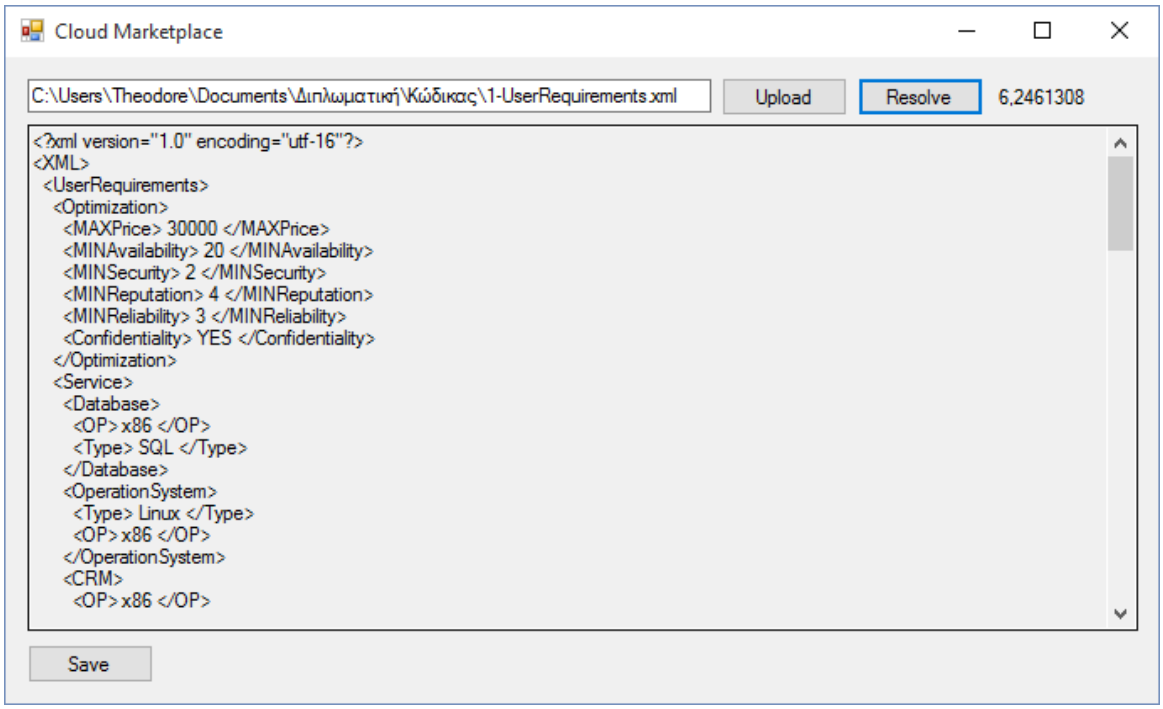


Σχήμα 20: Προσθήκη του BaseXClient.cs

Ολόκληρος ο κώδικας του project παρατίθεται στο Παράρτημα ΙΙ.

7.5 Η γραφική διεπιφάνεια χρήστη

Ο χρήστης μπορεί να επιλέξει και να ανεβάσει στο πρόγραμμα το αρχείο XML με τις επιθυμητές προδιαγραφές μέσω της φόρμας αυτής. Όταν πατήσει Resolve, ξεκινούν τα τέσσερα στάδια της επίλυσης του προβλήματος. Όταν ολοκληρωθεί η επίλυση του προβλήματος, η βέλτιστη λύση (ή το μήνυμα ότι δεν βρέθηκε λύση για τις δοθείσες απαιτήσεις), εμφανίζεται στο μεγάλο TextBox του παραθύρου. Επίσης, πάνω δεξιά φαίνεται ο χρόνος σε δευτερόλεπτα που μεσολάβησε από την αποστολή των απαιτήσεων του χρήστη στον `TechnicalResolution`, μέχρι τη στιγμή που η `EvaluationEngine` επέστρεψε στον `ResolutionManager` τη βέλτιστη λύση. Ο χρήστης έχει τη δυνατότητα να αποθηκεύει το XML της βέλτιστης λύσης στο φάκελο της επιλογής του, κάνοντας κλικ στο Save.



Σχήμα 21: Το γραφικό περιβάλλον του marketplace

7.6 Η παραμετροποίηση των ουρών μηνυμάτων

Στην περίπτωση της κοινής ουράς μηνυμάτων, η κοινή ουρά χρησιμοποιεί για τη λήψη των μηνυμάτων τη θύρα TCP 49021 και για την αποστολή των μηνυμάτων τη θύρα TCP 49022. Ως εκ τούτου, τα τέσσερα στάδια επίλυσης του προβλήματος και ο `ResolutionManager` στέλνουν όλοι τα μηνύματά τους προς τη θύρα TCP 49021 (όπου λαμβάνει η κοινή ουρά) και λαμβάνουν όλοι τα μηνύματά τους από τη θύρα 49022 (απ' όπου στέλνει η κοινή ουρά).

Στην περίπτωση των ξεχωριστών ουρών μηνυμάτων, ο `ResolutionManager` και τα τέσσερα στάδια επίλυσης, χρησιμοποιούν για την αποστολή και λήψη μηνυμάτων τις εξής θύρες TCP:

	Θύρα TCP λήψης	**Θύρα TCP αποστολής**
`ResolutionManager`	49035	49031
`TechnicalResolution`	49031	49032
`BusinessResolution`	49032	49033
`PriceAggregator`	49033	49034
`EvaluationEngine`	49034	49035

Πίνακας 1: Οι χρησιμοποιούμενες TCP θύρες για την επικοινωνία στην περίπτωση των ξεχωριστών ουρών

7.7 Η διαδικασία παραμετροποίησης των μετρήσεων

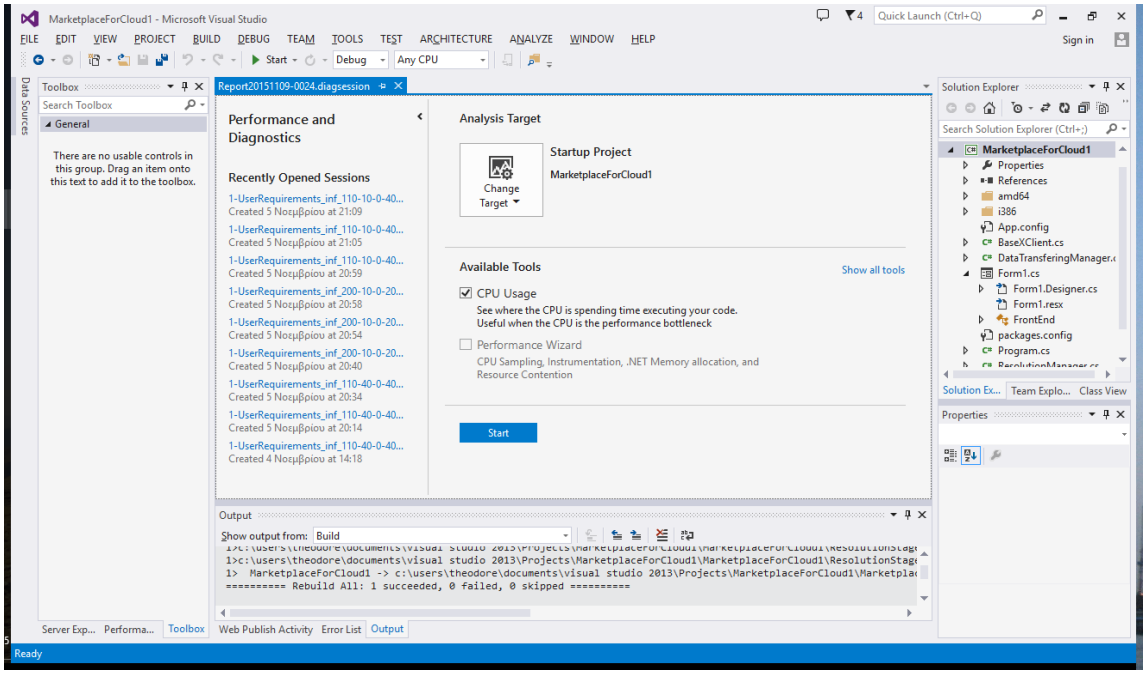
Προκειμένου να παραμετροποιήσουμε τη λειτουργία του marketplace σύμφωνα με όσα αναφέρθηκαν στο προηγούμενο κεφάλαιο, πρέπει να αλλάξουμε τις τιμές των εξής μεταβλητών της κλάσης `MarketplaceResolutionManager`:

- `int technicalDelay`: Ο ακέραιος αυτός μας δείχνει τη διάσταση των τετραγωνικών πινάκων που χρησιμοποιούνται για την εισαγωγή τεχνητής καθυστέρησες στον `TechnicalResolution`.
- `int businessDelay`: Ο ακέραιος αυτός μας δείχνει τη διάσταση των τετραγωνικών πινάκων που χρησιμοποιούνται για την εισαγωγή τεχνητής καθυστέρησες στον `BusingessResolution`.
- `int priceAggregatorDelay`: Ο ακέραιος αυτός μας δείχνει τη διάσταση των τετραγωνικών πινάκων που χρησιμοποιούνται για την εισαγωγή τεχνητής καθυστέρησες στον `PriceAggregator`.
- `int evaluationDelay`: Ο ακέραιος αυτός μας δείχνει τη διάσταση των τετραγωνικών πινάκων που χρησιμοποιούνται για την εισαγωγή τεχνητής καθυστέρησες στην `EvaluationEngine`.
- `int numOfCopies`: Ο ακέραιος αυτός μας δείχνει πόσα αντίγραφα κάθε αρχείου εξόδου θα στείλει ο `TechnicalResolution` στον `BusinessResolution`, προκειμένου να αυξηθεί η κίνηση της ουράς.

7.8 Η λήψη των μετρήσεων

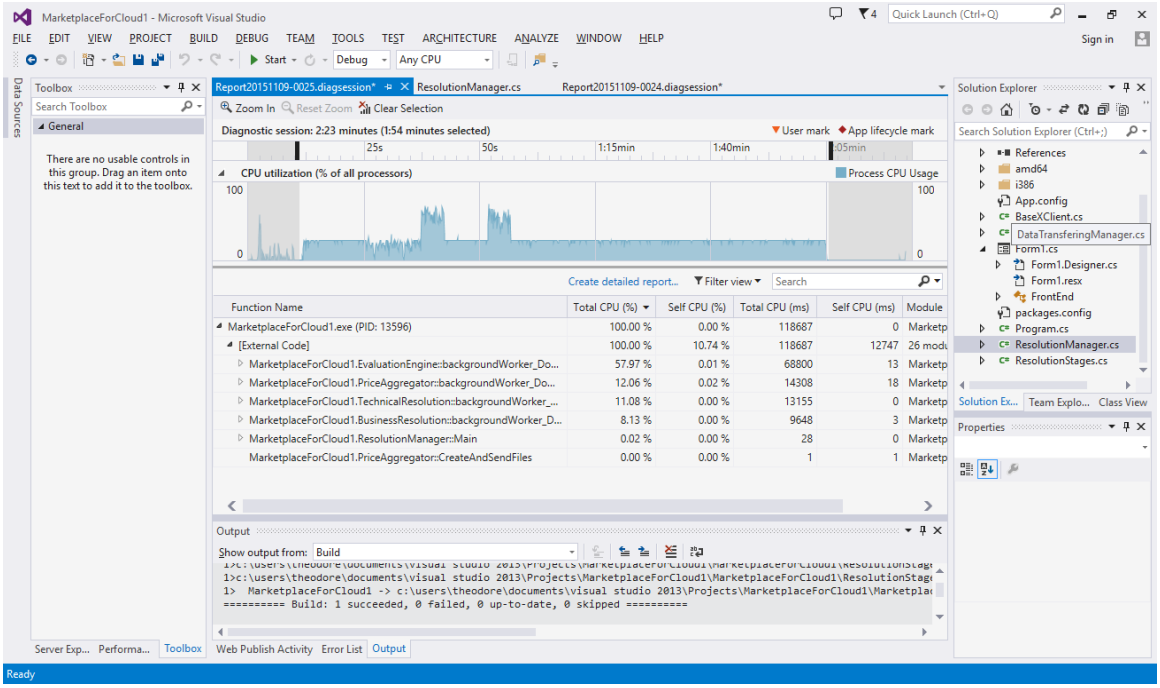
Η μέτρηση του συνολικού πραγματικού χρόνου εκτέλεσης των τεσσάρων σταδίων της επίλυσης γίνεται από το ίδιο το πρόγραμμα και εμφανίζεται στην διεπιφάνεια χρήστη όταν ολοκληρωθεί η διαδικασία.

Για τη μέτρηση των χρόνων CPU κάθε σταδίου χρησιμοποιήθηκε το εργαλείο Performance and Diagnostics του Visual Studio (Μενού Analyze→Performance and Diagnostics). Στο παράθυρο εκκίνησής του επιλέγουμε ως στόχο ανάλυσης (Analysis Target) το project εκκίνησης (Startup Project) και από τα διαθέσιμα εργαλεία (Available Tools) επιλέγουμε το CPU Usage. Σε αυτό το σημείο μπορούμε να εκκινήσουμε τη διαδικασία.



Σχήμα 22: Η παραμετροποίηση του εργαλείου Performance and Diagnostics

Αφού ο χρήστης εισάγει τις επιθυμητές προδιαγραφές, επιλυθεί το πρόβλημα και επιστραφεί η βέλτιστη λύση στον χρήστη, τότε κλείνουμε τη διεπιφάνεια χρήστη· η καταγραφή ολοκληρώνεται και το Visual Studio αναλύει τα συλλεγμένα δεδομένα. Στο παράθυρο της αναφοράς που θα ανοίξει, επιλέγουμε στο διάγραμμα χρήσης της CPU το χρονικό διάστημα που αντιστοιχεί στην εκτέλεση των τεσσάρων σταδίων της επίλυσης. Καταγράφουμε τα δεδομένα από τις στήλες Total CPU (%) και Total CPU (ms) για καθεμιά εκ των κλάσεων TechnicalResolution, BusinessResolution, PriceAggregator, EvaluationEngine και CommonQueue (εφ' όσον υπάρχει). Σε περίπτωση που κάποια κλάση εμφανιστεί δύο ή περισσότερες φορές (με διαφορετική καταχώρηση για κάθε εκτελούμενη συνάρτηση) τότε καταγράφουμε ως αποτέλεσμα τα αθροίσματα των τιμών κάθε καταχώρησης.



Σχήμα 23: Η μέτρηση ποσοστού χρήσης και χρόνου CPU

8 Μετρήσεις – συμπεράσματα

Για την εξαγωγή των συμπερασμάτων πραγματοποιήθηκαν μετρήσεις τόσο με διαφορετικό φόρτο εργασίας σε κάθε στάδιο (ώστε να αλλάζουν τα σημεία συμφόρησης της ουράς), όσο και με διαφορετικό πλήθος αποστελλόμενων πακέτων. Το αρχείο που χρησιμοποιήθηκε ως είσοδος υπάρχει στο Παράρτημα Ι. Η επεξεργασία του αρχείου αυτού από τον `TechnicalResolution` δημιουργεί 120 αρχεία εξόδου, τα οποία επεξεργάζεται ο `BusinessResolution`, ο οποίος δημιουργεί ως έξοδο 10.080 αρχεία. Από αυτά ο `PriceAggregator` θα προωθήσει στην `EvaluationEngine` τα 8.760.

Για τις ανάγκες της πειραματικής διαδικασίας ο `TechnicalResolution` παραμετροποιήθηκε όπως προαναφέρθηκε ώστε να εξάγει πολλαπλάσιο αριθμό αντιγράφων από τον κανονικό· συγκεκριμένα για τις μετρήσεις που έγιναν στα πλαίσια της διπλωματικής εξήγαγε τα πενταπλάσια ή τα δεκαπλάσια αρχεία. Έτσι οι μετρήσεις πραγματοποιήθηκαν για τρεις διαφορετικούς αριθμούς αρχείων, ήτοι:

- 1 → 120 → 10.080 →8.760 → 1,
- 1 → 600 → 50.400 → 43.800 → 1 και
- 1 → 1.200 → 100.800 →87.600 → 1.

Επίσης, μελετήσαμε τέσσερεις διαφορετικές περιπτώσεις για τα σημεία συμφόρησης της ουράς. Συγκεκριμένα:

- ομοιόμορφη συμφόρηση σε όλα τα στάδια (περίπτωση α΄),
- υψηλή συμφόρηση στο πρώτο στάδιο, δηλαδή στον `TechnicalResolution` (περίπτωση β΄),
- υψηλή συμφόρηση στο τελευταίο στάδιο, δηλαδή στη `ResolutionEngine` (περίπτωση γ΄) και
- υψηλή συμφόρηση στο δεύτερο και στο τέταρτο στάδιο ταυτοχρόνως, δηλαδή στα `BusinessResolution` και `ResolutionEngine` (περίπτωση δ΄).

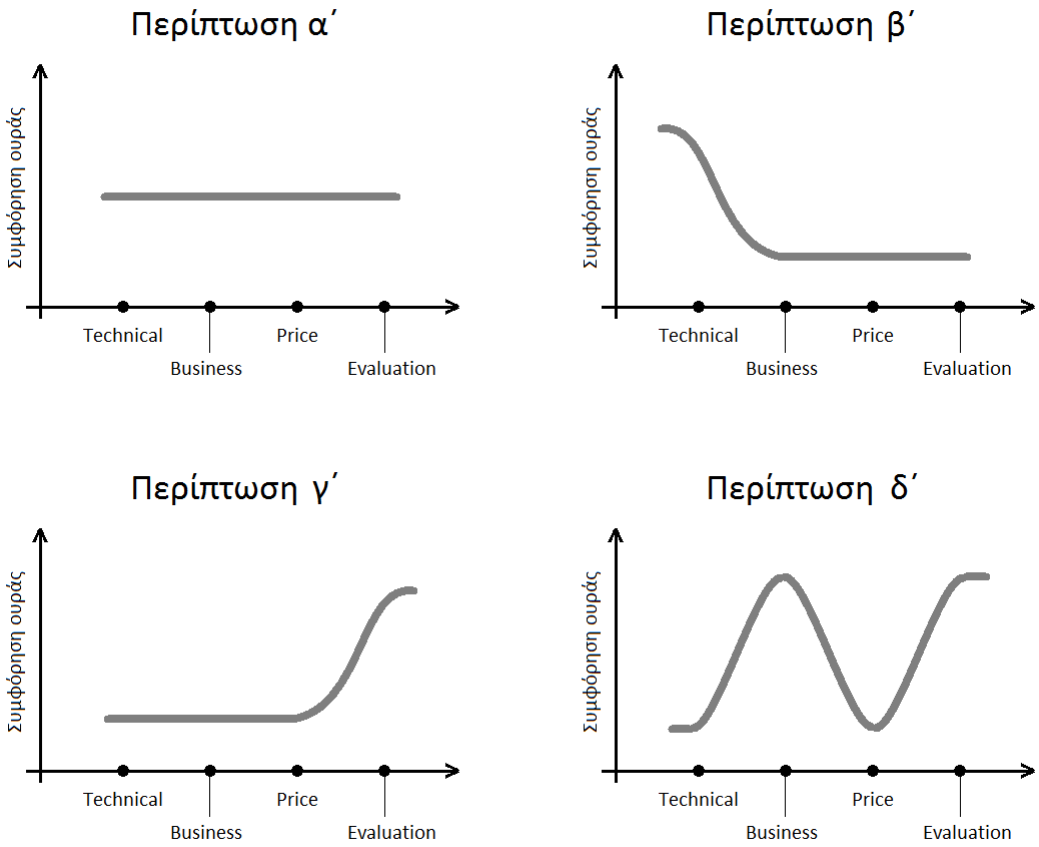
Για να επιτευχθούν οι παραμετροποιήσεις αυτές, κατόπιν δοκιμών, καταλήξαμε στα ακόλουθα μεγέθη των τετραγωνικών πινάκων καθυστέρησης:

	Technical Resolution	Business Resolution	Price Aggregator	Resolution Engine
Περ. α΄	110	10	0	20
Περ. β΄	200	10	0	20
Περ. γ΄	110	10	0	40
Περ. δ΄	110	40	0	20

Πίνακας 2: Το μέγεθος των τετραγωνικών πινάκων καθυστέρησης για κάθε περίπτωση

Για κάθε περίπτωση συμφόρησης της ουράς (4 συνολικά), μετρήσαμε τις επιδόσεις του marketplace και για τους 3 αριθμούς αρχείων που προαναφέραμε, οπότε πραγματοποιήθηκαν συνολικά 12 μετρήσεις. Οι παράμετροι που μετρήθηκαν σε κάθε μέτρηση είναι:

- ο χρόνος που έτρεξε κάθε στάδιο στη CPU (στην περίπτωση της κοινής ουράς μηνυμάτων μετρήσαμε και τον χρόνο CPU του `DataTransferingManager`),
- το ποσοστό του χρόνου που κατανάλωσε κάθε στάδιο σε σχέση με το σύνολο του χρόνου CPU του marketplace και
- τον πραγματικό χρόνο που χρειάσθηκε για να επιλυθεί το πρόβλημα.



Σχήμα 24: Οι τέσσερεις περιπτώσεις συμφόρησης ουράς

8.1 Ομοιόμορφη συμφόρηση ουράς (περίπτωση α΄)

Στην πρώτη περίπτωση, δηλαδή της ουράς με ομοιόμορφη συμφόρηση σε όλα τα στάδια, ελήφθησαν οι ακόλουθες μετρήσεις:

		Χωρίς χρήση ουράς		Κοινή ουρά		Ξεχωριστές ουρές	
		% CPU	Χρόνος CPU (s)	% CPU	Χρόνος CPU (s)	% CPU	Χρόνος CPU (s)
x1	Technical	23,99	2,766	15,89	3,354	18,59	3,213
	Business	16,14	1,861	15,5	3,272	18,23	3,15
	Price	18,91	2,181	17,56	3,707	20,66	3,572
	Evaluation	22,11	2,549	17,65	3,727	21,18	3,659
	DataTrans.	–	–	2,4	0,506	–	–
	Σύνολο CPU	**81,15**	**9,357**	**69**	**14,566**	**78,66**	**13,594**
	Πραγματικός χρόνος		**10,59**		**8,68**		**7,64**
x5	Technical	23,55	13,364	16,38	16,76	18,96	16,429
	Business	16,43	9,322	15,33	15,687	18,43	15,977
	Price	18,89	10,718	17,9	18,32	20,71	17,946
	Evaluation	22,27	12,637	17,83	18,244	21,05	18,244
	DataTrans			2,43	2,486		
	Σύνολο CPU	**81,14**	**46,041**	**69,87**	**71,497**	**79,15**	**68,596**
	Πραγματικός χρόνος		**48,62**		**43,14**		**36,62**
x10	Technical	23,02	25,546	16,54	33,03	18,98	33,064
	Business	16,28	18,059	15,36	30,674	18,25	31,79
	Price	19,16	21,263	17,69	35,328	20,64	35,954
	Evaluation	22,62	25,097	17,78	35,506	21,09	36,734
	DataTrans.			2,25	4,484		
	Σύνολο CPU	**81,08**	**89,965**	**69,62**	**139,022**	**78,96**	**137,542**
	Πραγματικός χρόνος		**94,42**		**85,76**		**72,31**

Πίνακας 3: Αποτελέσματα μετρήσεων για ομοιόμορφη συμφόρηση ουράς

8.2 Υψηλή συμφόρηση στο πρώτο στάδιο (περίπτωση β΄)

Στη δεύτερη περίπτωση, δηλαδή της ουράς με υψηλότερη συμφόρηση στο πρώτο στάδιο, ελήφθησαν οι ακόλουθες μετρήσεις:

		Χωρίς χρήση ουράς		Κοινή ουρά		Ξεχωριστές ουρές	
		% CPU	Χρόνος CPU (s)	% CPU	Χρόνος CPU (s)	% CPU	Χρόνος CPU (s)
x1	Technical	64,03	15,008	49,47	17,805	54,25	17,543
	Business	7,42	1,74	9,17	3,3	10,12	3,274
	Price	8,92	2,09	10,31	3,712	11,36	3,673
	Evaluation	10,41	2,44	10,56	3,8	11,88	3,839
	DataTrans.	–	–	2,4	0,506	–	–
	Σύνολο CPU	**90,78**	**21,278**	**81,1**	**29,191**	**87,61**	**28,329**
	Πραγματικός χρόνος		**21,95**		**20,71**		**19,58**
x5	Technical	63,54	75,907	48,93	88,189	54,84	88,175
	Business	7,52	8,986	9,16	16,511	10,14	16,303
	Price	8,83	105,49	10,42	18,788	11,27	18,118
	Evaluation	10,74	12,827	10,33	18,615	11,59	18,641
	DataTrans	–	–	1,84	3,312	–	–
	Σύνολο CPU	**90,63**	**203,21**	**80,68**	**145,415**	**87,84**	**141,237**
	Πραγματικός χρόνος		**113,28**		**104,25**		**99,62**
x10	Technical	62,83	163,386	51,03	194,141	54,5	174,909
	Business	7,79	20,27	8,77	33,351	10,22	32,793
	Price	9,26	24,09	9,86	37,513	11,39	36,562
	Evaluation	10,66	27,72	9,86	37,528	11,64	37,352
	DataTrans.	–	–	1,63	6,207	–	–
	Σύνολο CPU	**90,54**	**235,466**	**81,15**	**308,74**	**87,75**	**281,616**
	Πραγματικός χρόνος		**244,49**		**234,04**		**199,49**

Πίνακας 4: Αποτελέσματα μετρήσεων για συμφόρηση ουράς στο πρώτο στάδιο

8.3 Υψηλή συμφόρηση στο τελευταίο στάδιο (περίπτωση γ΄)

Στην τρίτη περίπτωση, δηλαδή της ουράς με υψηλότερη συμφόρηση στο τελευταίο στάδιο, ελήφθησαν οι ακόλουθες μετρήσεις:

		Χωρίς χρήση ουράς		Κοινή ουρά		Ξεχωριστές ουρές	
		% CPU	Χρόνος CPU (s)	% CPU	Χρόνος CPU (s)	% CPU	Χρόνος CPU (s)
x1	Technical	13,32	2,583	11,71	3,436	12,87	3,4
	Business	9,32	1,806	10,72	3,147	12,12	3,201
	Price	10,44	2,024	11,52	3,379	13,71	3,622
	Evaluation	56,02	10,859	43,64	12,806	46,74	12,348
	DataTrans.	–	–	1,4	0,411	–	–
	Σύνολο CPU	89,1	17,272	78,99	23,179	85,44	22,571
	Πραγματικός χρόνος		**17,88**		**16,11**		**15,65**
x5	Technical	12,73	11,99	11,93	17,903	13	17,226
	Business	8,99	8,47	10,54	15,815	12,1	16,031
	Price	10,22	9,625	11,35	17,047	13,58	18,001
	Evaluation	57,05	53,744	44,01	66,055	47,26	62,614
	DataTrans	–	–	1,36	2,043	–	–
	Σύνολο CPU	**88,99**	**83,829**	**79,19**	**118,863**	**85,94**	**113,872**
	Πραγματικός χρόνος		**87,87**		**81,07**		**76,29**
x10	Technical	12,92	14,335	11,95	35,503	13,22	35,036
	Business	9,04	17,021	10,5	31,194	12,11	32,086
	Price	10,6	19,969	11,47	34,081	13,66	36,211
	Evaluation	56,25	105,943	43,53	129,286	46,82	124,063
	DataTrans.	–	–	1,38	4,104	–	–
	Σύνολο CPU	**88,81**	**157,268**	**78,83**	**234,168**	**85,81**	**227,396**
	Πραγματικός χρόνος		**174,07**		**168,72**		**154,57**

Πίνακας 5: Αποτελέσματα μετρήσεων για συμφόρηση ουράς στο τελευταίο στάδιο

8.4 Υψηλή συμφόρηση στο δεύτερο κι στο τέταρτο στάδιο (περίπτωση δ΄)

Στην τέταρτη περίπτωση, δηλαδή της ουράς με υψηλότερη συμφόρηση στο δεύτερο και στο τέταρτο στάδιο, ελήφθησαν οι ακόλουθες μετρήσεις:

		Χωρίς χρήση ουράς		Κοινή ουρά		Ξεχωριστές ουρές	
		% CPU	Χρόνος CPU (s)	% CPU	Χρόνος CPU (s)	% CPU	Χρόνος CPU (s)
x1	Technical	8,5	2,591	6,59	3,677	7,52	3,519
	Business	42,18	12,853	33,87	18,893	39,79	18,624
	Price	6,71	2,044	6,62	3,692	7,45	3,489
	Evaluation	35,39	10,784	28,19	15,724	32,95	15,422
	DataTrans.	–	–	2,12	1,184	–	–
	Σύνολο CPU	**92,78**	**28,272**	**77,39**	**43,17**	**87,71**	**41,054**
	Πραγματικός χρόνος		**29,09**		**22,34**		**21,3**
x5	Technical	8,22	12,807	7,54	17,339	7,75	18,253
	Business	41,51	64,687	36,92	85,888	39,76	93,649
	Price	6,83	10,649	7,76	17,852	7,49	17,634
	Evaluation	36,26	56,505	31,07	71,432	32,78	77,201
	DataTrans	–	–	1,05	2,405	–	–
	Σύνολο CPU	**92,82**	**144,648**	**84,34**	**194,916**	**87,78**	**206,737**
	Πραγματικός χρόνος		**148,2**		**114,06**		**105,92**
x10	Technical	8,33	26,21	7,39	34,496	7,65	36,66
	Business	41,65	130,991	38,05	177,489	39,85	190,883
	Price	6,88	21,634	7,5	34,969	7,52	36,041
	Evaluation	35,55	111,792	30,34	141,538	32,74	156,848
	DataTrans.	–	–	0,97	4,522	–	–
	Σύνολο CPU	**92,41**	**290,627**	**84,25**	**393,014**	**87,76**	**420,432**
	Πραγματικός χρόνος		**302,42**		**285,5**		**216,21**

Πίνακας 6: Αποτελέσματα μετρήσεων για συμφόρηση ουράς στο δεύτερο και στο τέταρτο στάδιο

8.5 Συμπεράσματα

Κατ' αρχάς διαπιστώνουμε ότι και στις τέσσερις περιπτώσεις ο πραγματικός χρόνος επίλυσης του προβλήματος είναι αισθητά μικρότερος όταν χρησιμοποιούνται ουρές μηνυμάτων. Αυτό συμβαίνει εξαιτίας του επιτυγχανόμενου παραλληλισμού των σταδίων. Συγκεκριμένα, η κοινή ουρά αποδεικνύεται κατά μέσον όρο 9,2% γρηγορότερη στην περίπτωση α΄, 3,1% γρηγορότερη στην περίπτωση β΄, 5,6% γρηγορότερη στην περίπτωση γ΄ και 4,2% γρηγορότερη στην περίπτωση δ΄. Η μεγάλη διαφορά όμως φαίνεται στη χρήση ξεχωριστών ουρών, όπου κατά μέσον όρο η βελτίωση της ταχύτητας είναι 23,4% στην περίπτωση α΄, 11,2% στην περίπτωση β΄, 28,5% στην περίπτωση γ΄ και 18,4% στην περίπτωση δ΄. Είναι προφανές λοιπόν ότι η χρήση ουρών μηνυμάτων βελτιώνει σημαντικά το χρόνο εκτέλεσης και ότι η υλοποίηση με ξεχωριστές ουρές είναι σημαντικά αποδοτικότερη από αυτή με κοινή ουρά.

Παρατηρούμε επίσης ότι η βελτίωση που προσφέρουν οι ουρές είναι σχετικά μικρότερη στις περιπτώσεις β΄ και γ΄, όπου ο υψηλός φόρτος εργασίας παρατηρείται στην αρχή και στο τέλος αντίστοιχα της επίλυσης, πράγμα που μειώνει σημαντικά τη δυνατότητα παραλληλισμού.

Αναλύοντας τους χρόνους CPU κάθε σταδίου αλλά συνολικά, παρατηρούμε ότι στις περιπτώσεις όπου χρησιμοποιείται ουρά οι χρόνοι αυτοί είναι υψηλότεροι από τους αντίστοιχους της συμβατικής –χωρίς ουρές– υλοποίησης. Αυτό παρατηρείται πιο έντονα στις περιπτώσεις α΄ και δ΄. Αυτό οφείλεται στο ότι και τα τέσσερα στάδια τρέχουν στο ίδιο μηχάνημα, οπότε ο παραλληλισμός έχει ως αποτέλεσμα να μειώνει τις διαθέσιμες χρονοθυρίδες στους πυρήνες του επεξεργαστή. Αυτός είναι και ο λόγος που το φαινόμενο αυτό παρατηρείται πιο έντονα στις περιπτώσεις α΄ και δ΄, που επιτρέπουν περισσότερο παραλληλισμό. Επίσης, από αυτήν την παρατήρηση συνάγεται ότι, εάν τα τέσσερα στάδια εκτελούνταν σε διαφορετικά μηχανήματα, ο πραγματικός χρόνος επίλυσης θα διαρκούσε ακόμα λιγότερο.

Τελικά, συμπεραίνουμε ότι η χρήση ουρών μηνυμάτων, και ειδικά ξεχωριστής ουράς μεταξύ διαδοχικών σταδίων, μειώνει σημαντικά το χρόνο επίλυσης του προβλήματος. Η διαφορά αυτή θα είναι ακόμα πιο αισθητή όταν τα στάδια τρέχουν σε διαφορετικά μηχανήματα, καθώς η εκτέλεση κάθε σταδίου δεν θα επηρεάζει τη διαθεσιμότητα του επεξεργαστή στον οποίον τρέχουν τα άλλα στάδια.

9 Βιβλιογραφία

[1] M. Chen, S. Mao and Y. Liu, "Big data: A survey," *Mobile Networks and Applications,* vol. 19, no. 2, pp. 171-209, 2014.

[2] B. Gantz and D. Reinsel, "Extracting Value from Chaos State of the Universe : An Executive Summary," *IDC iView,* no. June, pp. 1-12, 2011.

[3] "Informatique | CERN," CERN, [Online]. Available: http://home.web.cern.ch/about/computing. [Accessed October 2015].

[4] X.Wu, X. Zhu, G. Wu and X. Ding, "Data Mining with Big Data," *IEEE Transactions on Knowledge and Data Engineering,* vol. 26, pp. 97-107, 2014.

[5] A. B. M. Moniruzzaman and S. A. Hossain, "NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison," *International Journal of Database Theory and Application,* vol. 6, no. 4, pp. 1-13, 2013.

[6] B. Mukherjee, Optical WDM Neworks, Springer, 2006.

[7] J. Vollbrecht, "Dynamic Circuit Network: An Introduction," 26 May 2008. [Online]. Available: http://indico.cern.ch/event/27666/session/7/contribution/8/attachments/503116/694720/DCN-Intro-Atlas-05.2008.LHC.pdf. [Accessed October 2015].

[8] C. P. Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data," *Information Sciences,* vol. 275, pp. 314-347, 2014.

[9] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Proceedings of 6th Symposium on Operating Systems Design and Implementation,* pp. 137-149, 2004.

[10] Canonical, "FrontPage - Storm," Canonical, 15 March 2011. [Online]. Available: https://storm.canonical.com/. [Accessed 2015].

[11] Infso D.4 Networked Enterprise & RFID; Infso G.2 Micro & Nanosystems; Working group RFID of the ETP EPoSS, "Internet of Things in 2020, Roadmap for the future," no. 1.1, 2008.

[12] L. M. Vaquero, L. Rodero-Merino, J. Caceres and M. Lindner, "A Break in the Clouds: Towards a Cloud Definition," *ACM SIGCOMM Computer Communication Review,* vol. 39, no. 1, pp. 50-55, 2009.

[13] P. Mell and T. Grance, "The NIST Definition of Cloud Computing - Recommendations of the National Institute of Standards and Technology," National Instityte of Standards and Technology, 2011.

[14] «Business as a Service (BaaS) - The New Cloud Management Concept,» AtomRain, 21 6 2012. [Ηλεκτρονικό]. [Πρόσβαση 10 2015].

[15] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger and D. Leaf, "NIST Cloud Computing Reference Architecture - Recommendations of the National Institute Standards and Technology," National Institute of Standards and Technology, 2011.

[16] G. Lu and W. H. Zeng, "Cloud Computing Survey," *Applied Mechanics and Materials,* Vols. 530-531, pp. 650-661, 2014.

[17] iMatix Corporation, "ØMQ - The Guide - ØMQ - The Guide," iMatix Corporation, 2014. [Online]. Available: http://zguide.zeromq.org/page:all. [Accessed 10 2015].

[18] iMatix Corporation, "zmq_socket(3) - 0MQ Api," iMatix Corporation, 2012. [Online]. Available: http://api.zeromq.org/3-2:zmq-socket. [Accessed 10 2015].

[19] A. Menychtas, J. Vogel, A. Giessmann, A. Gatzioura, S. Garcia Gomez, V. Moulos, F. Junker, M. Müller, D. Kyriazis, K. Stanoevska-Slabeva και T. Vargarigou, «4CaaSt marketplace: An advanced business environment for trading cloud services,» *Future Generation Computer Systems,* τόμ. 41, pp. 104-120, 2014.

Παράρτημα Ι

Το αρχείο εισόδου XML των απαιτήσεων του χρήστη που χρησιμοποιήθηκε στις μετρήσεις. Οι απαιτήσεις είναι αρκετά χαμηλές, ούτως ώστε να εξασφαλιστεί η δημιουργία πολλών πακέτων προσφορών με υπηρεσίες που διαφορετικά θα απορρίπτονταν.

```
<?xml version="1.0" encoding="UTF-8"?>
<XML>
  <UserRequirements>
    <Optimization>
      <MINAvailability>20</MINAvailability>
      <MINSecurity>2</MINSecurity>
      <MINReputation>4</MINReputation>
      <MINReliability>3</MINReliability>
      <Confidentiality>YES</Confidentiality>
      <MAXPrice>30000</MAXPrice>
    </Optimization>
    <Service>
      <Database>
        <OP>x86</OP>
        <Type>SQL</Type>
      </Database>
      <OperationSystem>
        <Type>Linux</Type>
        <OP>x86</OP>
      </OperationSystem>
      <CRM>
        <OP>x86</OP>
      </CRM>
      <Network>
        <Type>copper</Type>
      </Network>
    </Service>
  </UserRequirements>
  <Technical>
  </Technical>
  <Business>
  </Business>
</XML>
```

Παράρτημα ΙΙ

Ο κώδικας του Front End μοιράζεται σε δύο αρχεία, το Form1.cs και το Form1.Designer.cs.

Form1.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml;

namespace MarketplaceForCloud1
{
    public partial class FrontEnd : Form
    {
        public double ElapsedSeconds
        {
            set
            {
                this.label1.Text = value.ToString();
            }
        }
        public FrontEnd()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            if ((openFileDialog1.ShowDialog()== DialogResult.OK)
                && (!String.IsNullOrWhiteSpace(openFileDialog1.FileName)))
                textBox1.Text = openFileDialog1.FileName;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            if (File.Exists(textBox1.Text))
            {
                MarketplaceResolutionManager resolutionManager = new
MarketplaceResolutionManager();
                string userRequirements = File.ReadAllText(textBox1.Text);
                string resolution;
                this.label1.Text = resolutionManager.Resolve(userRequirements, out
resolution).ToString();

                // Format XML file
                try
                {
                    StringBuilder sb = new StringBuilder();
                    XmlWriterSettings settings = new XmlWriterSettings
                    {
                        Indent = true,
                        IndentChars = "  ",
                        NewLineChars = "\r\n",
                        NewLineHandling = NewLineHandling.Replace
```

```
                    };
                    using (XmlWriter writer = XmlWriter.Create(sb, settings))
                    {
                        XmlDocument doc = new XmlDocument();
                        doc.LoadXml(resolution);
                        doc.Save(writer);
                    }
                    resolution = sb.ToString();
                }
                catch { }
                this.textBox2.Text = resolution;
            }
        }

        private void button3_Click(object sender, EventArgs e)
        {
            if (this.saveFileDialog1.ShowDialog() == DialogResult.OK)
                File.WriteAllText(saveFileDialog1.FileName, this.textBox2.Text);
        }
    }
}
```

Form1.Designer.cs:

```
namespace MarketplaceForCloud1
{
    partial class FrontEnd
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed;
otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.button1 = new System.Windows.Forms.Button();
            this.button2 = new System.Windows.Forms.Button();
            this.label1 = new System.Windows.Forms.Label();
            this.textBox2 = new System.Windows.Forms.TextBox();
            this.openFileDialog1 = new System.Windows.Forms.OpenFileDialog();
            this.saveFileDialog1 = new System.Windows.Forms.SaveFileDialog();
            this.button3 = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // textBox1
            //
            this.textBox1.Location = new System.Drawing.Point(13, 13);
            this.textBox1.Name = "textBox1";
            this.textBox1.Size = new System.Drawing.Size(407, 20);
            this.textBox1.TabIndex = 0;
            //
            // button1
            //
            this.button1.Location = new System.Drawing.Point(426, 12);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(75, 23);
            this.button1.TabIndex = 1;
            this.button1.Text = "Upload";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            // button2
            //
```

```
        this.button2.Location = new System.Drawing.Point(507, 12);
        this.button2.Name = "button2";
        this.button2.Size = new System.Drawing.Size(75, 23);
        this.button2.TabIndex = 2;
        this.button2.Text = "Resolve";
        this.button2.UseVisualStyleBackColor = true;
        this.button2.Click += new System.EventHandler(this.button2_Click);
        //
        // label1
        //
        this.label1.AutoSize = true;
        this.label1.Location = new System.Drawing.Point(588, 17);
        this.label1.Name = "label1";
        this.label1.Size = new System.Drawing.Size(0, 13);
        this.label1.TabIndex = 3;
        //
        // textBox2
        //
        this.textBox2.Location = new System.Drawing.Point(13, 40);
        this.textBox2.Multiline = true;
        this.textBox2.Name = "textBox2";
        this.textBox2.ReadOnly = true;
        this.textBox2.ScrollBars = System.Windows.Forms.ScrollBars.Both;
        this.textBox2.Size = new System.Drawing.Size(661, 301);
        this.textBox2.TabIndex = 4;
        //
        // openFileDialog1
        //
        this.openFileDialog1.FileName = "openFileDialog1";
        //
        // button3
        //
        this.button3.Location = new System.Drawing.Point(13, 349);
        this.button3.Name = "button3";
        this.button3.Size = new System.Drawing.Size(75, 23);
        this.button3.TabIndex = 5;
        this.button3.Text = "Save";
        this.button3.UseVisualStyleBackColor = true;
        this.button3.Click += new System.EventHandler(this.button3_Click);
        //
        // FrontEnd
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(686, 384);
        this.Controls.Add(this.button3);
        this.Controls.Add(this.textBox2);
        this.Controls.Add(this.label1);
        this.Controls.Add(this.button2);
        this.Controls.Add(this.button1);
        this.Controls.Add(this.textBox1);
        this.Name = "FrontEnd";
        this.Text = "Cloud Marketplace";
        this.ResumeLayout(false);
        this.PerformLayout();

    }

    #endregion

    private System.Windows.Forms.TextBox textBox1;
    private System.Windows.Forms.Button button1;
```

```
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox textBox2;
        private System.Windows.Forms.OpenFileDialog openFileDialog1;
        private System.Windows.Forms.SaveFileDialog saveFileDialog1;
        private System.Windows.Forms.Button button3;
    }
}
```

Ο κώδικας του ResolutionManager βρίσκεται στο αρχείο **ResolutionManager.cs**:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ZeroMQ;

namespace MarketplaceForCloud1
{
    public class MarketplaceResolutionManager
    {
        // Measurement parameters
        Architectures architecture = Architectures.SeparateQueues;
        int technicalDelay = 110;
        int businessDelay = 10;
        int priceAggregatorDelay = 0;
        int evaluationDelay = 40;
        int numOfCopies = 5;

        public double Resolve(string userRequirements, out string resolution)
        {
            int queueSize = 0; // Infinite queue size
            DateTime? startTime = null;

            ZMessage resolutionMessage = null;
            switch (architecture)
            {
                case Architectures.SeparateQueues:
                case Architectures.NoQueue:
                    new TechnicalResolution("tcp://localhost:49031",
"tcp://*:49032", architecture, queueSize / 4, technicalDelay, numOfCopies);
                    new BusinessResolution("tcp://localhost:49032",
"tcp://*:49033", architecture, queueSize / 4, businessDelay);
                    new PriceAggregator("tcp://localhost:49033", "tcp://*:49034",
architecture, queueSize / 4, priceAggregatorDelay);
                    new EvaluationEngine("tcp://localhost:49034", "tcp://*:49035",
architecture, queueSize / 4, evaluationDelay);

                    using (ZContext context = new ZContext())
                    using (ZSocket sendQueue = new ZSocket(context,
ZSocketType.DEALER),
                                   receiveQueue = new ZSocket(context,
ZSocketType.DEALER))
                    using (ZMessage messageUserRequirements = new ZMessage(),
                                    messageTotalFiles = new ZMessage())
                    {
                        sendQueue.Bind("tcp://*:49031");
                        receiveQueue.Connect("tcp://localhost:49035");
                        messageUserRequirements.Add(new ZFrame(userRequirements));
                        System.Threading.Thread.Sleep(2000);
                        startTime = DateTime.Now;
                        Console.WriteLine("Sending user requirements.");
                        sendQueue.Send(messageUserRequirements);
                        messageTotalFiles.Add(new ZFrame("TotalFiles1"));
                        sendQueue.Send(messageTotalFiles);
                        resolutionMessage = receiveQueue.ReceiveMessage();
                    }
                    break;
                case Architectures.CommonQueue:
```

```
                    new CommonQueue("tcp://*:49021", "tcp://*:49022", queueSize);
                    new TechnicalResolution("tcp://localhost:49022",
"tcp://localhost:49021", architecture, queueSize, technicalDelay, numOfCopies);
                    new BusinessResolution("tcp://localhost:49022",
"tcp://localhost:49021", architecture, queueSize, businessDelay);
                    new PriceAggregator("tcp://localhost:49022",
"tcp://localhost:49021", architecture, queueSize, priceAggregatorDelay);
                    new EvaluationEngine("tcp://localhost:49022",
"tcp://localhost:49021", architecture, queueSize, evaluationDelay);

                    using (ZContext context = new ZContext())
                    using (ZSocket sendQueue = new ZSocket(context,
ZSocketType.DEALER),
                                    receiveQueue = new ZSocket(context,
ZSocketType.DEALER))
                    using (ZMessage messageUserRequirements = new ZMessage(),
                                    messageTotalFiles = new ZMessage())
                    {
                        sendQueue.Connect("tcp://localhost:49021");
                        receiveQueue.IdentityString =
"MarketplaceResolutionManager";
                        receiveQueue.Connect("tcp://localhost:49022");
                        // Single queue routing initialization
                        messageUserRequirements.Add(new
ZFrame("TechnicalResolution"));
                        messageUserRequirements.Add(new ZFrame(userRequirements));
                        System.Threading.Thread.Sleep(2000);
                        startTime = DateTime.Now;
                        Console.WriteLine("Sending user requirements.");
                        sendQueue.Send(messageUserRequirements);
                        messageTotalFiles.Add(new ZFrame("TechnicalResolution"));
                        messageTotalFiles.Add(new ZFrame("TotalFiles1"));
                        sendQueue.Send(messageTotalFiles);
                        resolutionMessage = receiveQueue.ReceiveMessage();
                    }
                    break;
            }
            if ((resolutionMessage != null)
                && (resolutionMessage[0] != null))
                resolution = resolutionMessage[0].ReadString();
            else
                resolution = "Error";
            return (DateTime.Now - (DateTime)startTime).TotalSeconds;
        }
    }


}
```

Ο κώδικας του DataTransferingManager βρίσκεται στο αρχείο **DataTransferingManager.cs**:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ZeroMQ;
```

```
namespace MarketplaceForCloud1
{
    public enum Architectures { CommonQueue, SeparateQueues, NoQueue }

    public class CommonQueue
    {
        public string ReceiveQueueAddress { get; private set; }
        public string SendQueueAddress { get; private set; }
        public int RcvHwm { get; private set; }
        private BackgroundWorker backgroundWorker;
        public CommonQueue(string receiveQueueAddress, string sendQueueAddress,
int rcvHwm)
        {
            ReceiveQueueAddress = receiveQueueAddress;
            SendQueueAddress = sendQueueAddress;
            RcvHwm = rcvHwm;
            this.backgroundWorker = new BackgroundWorker();
            this.backgroundWorker.DoWork += backgroundWorker_DoWork;
            this.backgroundWorker.RunWorkerAsync();
        }
        private void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
        {
            using (ZContext context = new ZContext())
            using (ZSocket receiveQueue = new ZSocket(context,
ZSocketType.DEALER),
                           sendQueue = new ZSocket(context, ZSocketType.ROUTER))
            {
                receiveQueue.ReceiveHighWatermark = RcvHwm;
                receiveQueue.Bind(ReceiveQueueAddress);
                sendQueue.SendHighWatermark = 2;
                sendQueue.RouterMandatory = RouterMandatory.Report;
                sendQueue.Bind(SendQueueAddress);
                //ZContext.Proxy(sendQueue, receiveQueue);
                while (true)
                    sendQueue.SendMessage(receiveQueue.ReceiveMessage());
            }
        }

    }
}
```

Ο κώδικας των τεσσάρων σταδίων επίλυσης βρίσκεται στο αρχείο **ResolutionStages.cs**:

```
using BaseXClient;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml;
using ZeroMQ;

namespace MarketplaceForCloud1
{
    public class TechnicalResolution : ResolutionBase
    {
        public int NumOfCopies { get; private set; }

        public TechnicalResolution(string receiveQueueAddress, string
sendQueueAddress, Architectures architecture, int rcvHwm, int arraySize, int
numOfCopies)
            : base(receiveQueueAddress, sendQueueAddress, architecture, rcvHwm,
arraySize)
        {
            NumOfCopies = numOfCopies;
            this.backgroundWorker.DoWork += backgroundWorker_DoWork;
            this.backgroundWorker.RunWorkerAsync();
        }
        public TechnicalResolution(string receiveQueueAddress, string
sendQueueAddress, Architectures architecture, int rcvHwm, int arraySize)
            : this(receiveQueueAddress, sendQueueAddress, architecture, rcvHwm,
arraySize, 1) { }

        protected override int CreateAndSendFiles(XmlDocument xmlIn, ZSocket
sendQueue)
        {
            if (xmlIn == null)
                return 0;
            // Parse user requirements.
            string databaseType =
xmlIn.SelectSingleNode("/XML/UserRequirements/Service/Database/Type").InnerText;
            string databaseOp =
xmlIn.SelectSingleNode("/XML/UserRequirements/Service/Database/OP").InnerText;
            string operationSystemType =
xmlIn.SelectSingleNode("/XML/UserRequirements/Service/OperationSystem/Type").Inner
Text;
            string operationSystemOp =
xmlIn.SelectSingleNode("/XML/UserRequirements/Service/OperationSystem/OP").InnerTe
xt;
            string crmOp =
xmlIn.SelectSingleNode("/XML/UserRequirements/Service/CRM/OP").InnerText;
            string networkType =
xmlIn.SelectSingleNode("/XML/UserRequirements/Service/Network/Type").InnerText;

            // Query technical blueprints database.
            Session session = new Session("localhost", 1984, "admin", "admin");
            session.Execute("open TechnicalBlueprints");
            string queryString;
            XmlDocument databases = new XmlDocument();
            XmlDocument operationSystems = new XmlDocument();
            XmlDocument crms = new XmlDocument();
            XmlDocument networks = new XmlDocument();
```

```
            // Databases
            queryString = "<Databases>{/TechnicalBlueprints/Database";
            if (!String.IsNullOrWhiteSpace(databaseType)
                || !String.IsNullOrWhiteSpace(databaseOp))
            {
                queryString += "[";
                if (!String.IsNullOrWhiteSpace(databaseType))
                    queryString += "Type = \"" + databaseType.Trim() + "\" and ";
                if (!String.IsNullOrWhiteSpace(databaseOp))
                    queryString += "OP = \"" + databaseOp.Trim() + "\" and ";
                queryString = queryString.Substring(0, queryString.Length - 5) +
"]";
            }
            queryString += "}</Databases>";
            databases.LoadXml(session.Query(queryString).Execute());
            // Operation Systems
            queryString =
"<OperationSystems>{/TechnicalBlueprints/OperationSystem";
            if (!String.IsNullOrWhiteSpace(operationSystemType)
                || !String.IsNullOrWhiteSpace(operationSystemOp))
            {
                queryString += "[";
                if (!String.IsNullOrWhiteSpace(operationSystemType))
                    queryString += "Type = \"" + operationSystemType.Trim() + "\"
and ";
                if (!String.IsNullOrWhiteSpace(operationSystemOp))
                    queryString += "OP = \"" + operationSystemOp.Trim() + "\" and
";
                queryString = queryString.Substring(0, queryString.Length - 5) +
"]";
            }
            queryString += "}</OperationSystems>";
            operationSystems.LoadXml(session.Query(queryString).Execute());
            // CRMs
            queryString = "<CRMs>{/TechnicalBlueprints/CRM";
            if (!String.IsNullOrWhiteSpace(crmOp))
                queryString += "[OP = \"" + crmOp.Trim() + "\"]";
            queryString += "}</CRMs>";
            crms.LoadXml(session.Query(queryString).Execute());
            // Networks
            queryString = "<Networks>{/TechnicalBlueprints/Network";
            if (!String.IsNullOrWhiteSpace(networkType))
                queryString += "[Type = \"" + networkType.Trim() + "\"]";
            queryString += "}</Networks>";
            networks.LoadXml(session.Query(queryString).Execute());
            session.Close();

            // Create and send technical resolutions.
            int filesSent = 0;
            for (int i = 0; i < NumOfCopies; i++)
                foreach (XmlNode database in
databases.SelectNodes("/Databases/Database"))
                    foreach (XmlNode operationSystem in
operationSystems.SelectNodes("/OperationSystems/OperationSystem"))
                        foreach (XmlNode crm in crms.SelectNodes("/CRMs/CRM"))
                            foreach (XmlNode network in
networks.SelectNodes("/Networks/Network"))
                            {
                                // Create one XML.
                                XmlDocument xmlOut = new XmlDocument();
                                xmlOut.LoadXml(xmlIn.OuterXml);
```

```
                                XmlNode node =
xmlOut.SelectSingleNode("/XML/Technical");
                                node.AppendChild(xmlOut.ImportNode(database,
true));

node.AppendChild(xmlOut.ImportNode(operationSystem, true));
                                node.AppendChild(xmlOut.ImportNode(crm, true));
                                node.AppendChild(xmlOut.ImportNode(network,
true));
                                // Delay
                                Delay();
                                // Send one XML.
                                if (Architecture != Architectures.NoQueue)
                                    using (ZMessage message = new ZMessage())
                                    {
                                        switch (Architecture)
                                        {
                                            case Architectures.SeparateQueues:
                                                message.Add(new
ZFrame(xmlOut.OuterXml));
                                                break;
                                            case Architectures.CommonQueue:
                                                message.Add(new
ZFrame("BusinessResolution"));
                                                message.Add(new
ZFrame(xmlOut.OuterXml));
                                                break;
                                        }
                                        sendQueue.Send(message);
                                    }
                                else
                                    this.filesToBeSentList.Add(xmlOut.OuterXml);
                                filesSent++;
                            }
            return filesSent;
        }
        protected void Delay()
        {
            if (ArraySize <= 0)
                return;
            int d1 = ArraySize, d2 = ArraySize, d3 = ArraySize;
            int[,] factor1 = new int[d1, d2];
            int[,] factor2 = new int[d2, d3];
            int[,] product = new int[d1, d3];
            Random random = new Random();
            for (int i = 0; i < factor1.GetLength(0); i++)
                for (int j = 0; j < factor1.GetLength(1); j++)
                    factor1[i, j] = random.Next();
            for (int i = 0; i < factor2.GetLength(0); i++)
                for (int j = 0; j < factor2.GetLength(1); j++)
                    factor2[i, j] = random.Next();
            for (int i = 0; i < factor1.GetLength(0); i++)
                for (int j = 0; j < factor2.GetLength(1); j++)
                {
                    product[i, j] = 0;
                    for (int k = 0; k < d2; k++)
                        product[i, j] += factor1[i, k] * factor2[k, j];
                }
            return;
        }
        protected void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
        {
```

```
            using (ZContext context = new ZContext())
            using (ZSocket receiveQueue = new ZSocket(context,
ZSocketType.DEALER),
                          sendQueue = new ZSocket(context, ZSocketType.DEALER))
            {
                int filesToBeReceived = 0;
                // Initialize queues
                sendQueue.SendHighWatermark = 2;
                if (!String.IsNullOrWhiteSpace(SendQueueAddress))
                    switch (Architecture)
                    {
                        case Architectures.SeparateQueues:
                        case Architectures.NoQueue:
                            receiveQueue.ReceiveHighWatermark = RcvHwm;
                            sendQueue.Bind(SendQueueAddress);
                            break;
                        case Architectures.CommonQueue:
                            sendQueue.Connect(SendQueueAddress);
                            receiveQueue.ReceiveHighWatermark = 2;
                            receiveQueue.IdentityString = this.GetType().Name;
                            break;
                    }
                receiveQueue.Connect(ReceiveQueueAddress);
                Console.WriteLine(this.GetType().Name + " ready.");
                while (true)
                {
                    // Receive message.
                    string messageString;
                    using (ZFrame frame = receiveQueue.ReceiveMessage()[0])
                    {
                        messageString = frame.ReadString();
                    }
                    if (messageString.StartsWith("TotalFiles"))
                        filesToBeReceived =
Int32.Parse(messageString.Substring("TotalFiles".Length));
                    else
                    {
                        XmlDocument xmlIn = new XmlDocument();
                        xmlIn.LoadXml(messageString);
                        FilesReceived++;
                        FilesSent += CreateAndSendFiles(xmlIn, sendQueue);
                    }
                    if (FilesReceived == filesToBeReceived)
                    {
                        if (Architecture == Architectures.NoQueue)
                            foreach (string xmlOut in this.filesToBeSentList)
                                using (ZFrame frame = new ZFrame(xmlOut))
                                using (ZMessage message = new ZMessage(new
ZFrame[] { frame }))
                                {
                                    sendQueue.SendMessage(message);
                                }
                        ZMessage messageTotalFiles = new ZMessage();
                        if (Architecture == Architectures.CommonQueue)
                            switch (this.GetType().Name)
                            {
                                case "TechnicalResolution":
                                    messageTotalFiles.Add(new
ZFrame("BusinessResolution"));
                                    break;
                                case "BusinessResolurion":
```

```
                                    messageTotalFiles.Add(new
ZFrame("PriceAggregator"));
                                    break;
                                case "PriceAggregator":
                                    messageTotalFiles.Add(new
ZFrame("EvaluationEngine"));
                                    break;
                                case "EvaluationEngine":
                                    messageTotalFiles.Add(new
ZFrame("MarketplaceResolutionManager"));
                                    break;
                            }
                        messageTotalFiles.Add(new ZFrame("TotalFiles" +
FilesSent));
                        sendQueue.SendMessage(messageTotalFiles);
                        break;
                    }
                }
                Console.WriteLine(this.GetType().Name + ": " + filesToBeReceived +
" -> " + FilesSent);
            }

        }
    }

    public class BusinessResolution : ResolutionBase
    {
        public BusinessResolution(string receiveQueueAddress, string
sendQueueAddress, Architectures architecture, int rcvHwm, int arraySize)
            : base(receiveQueueAddress, sendQueueAddress, architecture, rcvHwm,
arraySize)
        {
            this.backgroundWorker.DoWork += backgroundWorker_DoWork;
            this.backgroundWorker.RunWorkerAsync();
        }


        protected override int CreateAndSendFiles(XmlDocument xmlIn, ZSocket
sendQueue)
        {
            if (xmlIn == null)
                return 0;
            // Parse technical resolution.
            string databaseId =
xmlIn.SelectSingleNode("/XML/Technical/Database/ID").InnerText;
            string operationSystemId =
xmlIn.SelectSingleNode("/XML/Technical/OperationSystem/ID").InnerText;
            string crmId =
xmlIn.SelectSingleNode("/XML/Technical/CRM/ID").InnerText;
            string networkId =
xmlIn.SelectSingleNode("/XML/Technical/Network/ID").InnerText;

            // Query business bleuprints database.
            Session session = new Session("localhost", 1984, "admin", "admin");
            session.Execute("open BusinessBlueprints");
            string queryString;
            XmlDocument pDatabases = new XmlDocument();
            XmlDocument pOperationSystems = new XmlDocument();
            XmlDocument pCrms = new XmlDocument();
            XmlDocument pNetworks = new XmlDocument();
            // Databases
            queryString = "<PDatabases>{BusinessBlueprints/PDatabase";
```

```
            if (!String.IsNullOrWhiteSpace(databaseId))
                queryString += "[ID = " + databaseId.Trim() + "]";
            queryString += "}</PDatabases>";
            pDatabases.LoadXml(session.Query(queryString).Execute());
            // OperationSystems
            queryString =
"<POperationSystems>{BusinessBlueprints/POperationSystem";
            if (!String.IsNullOrWhiteSpace(operationSystemId))
                queryString += "[ID = " + operationSystemId.Trim() + "]";
            queryString += "}</POperationSystems>";
            pOperationSystems.LoadXml(session.Query(queryString).Execute());
            // CRMs
            queryString = "<PCrms>{BusinessBlueprints/PCRM";
            if (!String.IsNullOrWhiteSpace(crmId))
                queryString += "[ID = " + crmId.Trim() + "]";
            queryString += "}</PCrms>";
            pCrms.LoadXml(session.Query(queryString).Execute());
            // Networks
            queryString = "<PNetworks>{BusinessBlueprints/PNetwork";
            if (!String.IsNullOrWhiteSpace(networkId))
                queryString += "[ID = " + networkId.Trim() + "]";
            queryString += "}</PNetworks>";
            pNetworks.LoadXml(session.Query(queryString).Execute());

            // Create and send business resolutions.
            int filesSent = 0;
            foreach (XmlNode pDatabase in
pDatabases.SelectNodes("/PDatabases/PDatabase"))
                foreach (XmlNode pOperationSystem in
pOperationSystems.SelectNodes("/POperationSystems/POperationSystem"))
                    foreach (XmlNode pCrm in pCrms.SelectNodes("/PCrms/PCRM"))
                        foreach (XmlNode pNetwork in
pNetworks.SelectNodes("/PNetworks/PNetwork"))
                        {
                            // Create one XML.
                            XmlDocument xmlOut = new XmlDocument();
                            xmlOut.LoadXml(xmlIn.OuterXml);
                            XmlNode node =
xmlOut.SelectSingleNode("/XML/Business");
                            node.AppendChild(xmlOut.ImportNode(pDatabase, true));
                            node.AppendChild(xmlOut.ImportNode(pOperationSystem,
true));
                            node.AppendChild(xmlOut.ImportNode(pCrm, true));
                            node.AppendChild(xmlOut.ImportNode(pNetwork, true));
                            //Delay
                            Delay();
                            // Send one XML
                            if (Architecture != Architectures.NoQueue)
                                using (ZMessage message = new ZMessage())
                                {
                                    switch (Architecture)
                                    {
                                        case Architectures.SeparateQueues:
                                            message.Add(new
ZFrame(xmlOut.OuterXml));
                                            break;
                                        case Architectures.CommonQueue:
                                            message.Add(new
ZFrame("PriceAggregator"));
                                            message.Add(new
ZFrame(xmlOut.OuterXml));
                                            break;
```

```
                                }
                                sendQueue.Send(message);
                            }
                        else
                            this.filesToBeSentList.Add(xmlOut.OuterXml);
                        filesSent++;
                    }
            return filesSent;
        }
        protected void Delay()
        {
            if (ArraySize <= 0)
                return;
            int d1 = ArraySize, d2 = ArraySize, d3 = ArraySize;
            int[,] factor1 = new int[d1, d2];
            int[,] factor2 = new int[d2, d3];
            int[,] product = new int[d1, d3];
            Random random = new Random();
            for (int i = 0; i < factor1.GetLength(0); i++)
                for (int j = 0; j < factor1.GetLength(1); j++)
                    factor1[i, j] = random.Next();
            for (int i = 0; i < factor2.GetLength(0); i++)
                for (int j = 0; j < factor2.GetLength(1); j++)
                    factor2[i, j] = random.Next();
            for (int i = 0; i < factor1.GetLength(0); i++)
                for (int j = 0; j < factor2.GetLength(1); j++)
                {
                    product[i, j] = 0;
                    for (int k = 0; k < d2; k++)
                        product[i, j] += factor1[i, k] * factor2[k, j];
                }
            return;
        }
        protected void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
        {
            using (ZContext context = new ZContext())
            using (ZSocket receiveQueue = new ZSocket(context,
ZSocketType.DEALER),
                           sendQueue = new ZSocket(context, ZSocketType.DEALER))
            {
                int filesToBeReceived = 0;
                // Initialize queues
                sendQueue.SendHighWatermark = 2;
                if (!String.IsNullOrWhiteSpace(SendQueueAddress))
                    switch (Architecture)
                    {
                        case Architectures.SeparateQueues:
                        case Architectures.NoQueue:
                            receiveQueue.ReceiveHighWatermark = RcvHwm;
                            sendQueue.Bind(SendQueueAddress);
                            break;
                        case Architectures.CommonQueue:
                            sendQueue.Connect(SendQueueAddress);
                            receiveQueue.ReceiveHighWatermark = 2;
                            receiveQueue.IdentityString = this.GetType().Name;
                            break;
                    }
                receiveQueue.Connect(ReceiveQueueAddress);
                Console.WriteLine(this.GetType().Name + " ready.");
                while (true)
                {
                    // Receive message.
```

```
                    string messageString;
                    using (ZFrame frame = receiveQueue.ReceiveMessage()[0])
                    {
                        messageString = frame.ReadString();
                    }
                    if (messageString.StartsWith("TotalFiles"))
                        filesToBeReceived =
Int32.Parse(messageString.Substring("TotalFiles".Length));
                    else
                    {
                        XmlDocument xmlIn = new XmlDocument();
                        xmlIn.LoadXml(messageString);
                        FilesReceived++;
                        FilesSent += CreateAndSendFiles(xmlIn, sendQueue);
                    }
                    if (FilesReceived == filesToBeReceived)
                    {
                        if (Architecture == Architectures.NoQueue)
                            foreach (string xmlOut in this.filesToBeSentList)
                                using (ZFrame frame = new ZFrame(xmlOut))
                                using (ZMessage message = new ZMessage(new
ZFrame[] { frame }))
                                {
                                    sendQueue.SendMessage(message);
                                }
                        ZMessage messageTotalFiles = new ZMessage();
                        if (Architecture == Architectures.CommonQueue)
                            switch (this.GetType().Name)
                            {
                                case "TechnicalResolution":
                                    messageTotalFiles.Add(new
ZFrame("BusinessResolution"));
                                    break;
                                case "BusinessResolution":
                                    messageTotalFiles.Add(new
ZFrame("PriceAggregator"));
                                    break;
                                case "PriceAggregator":
                                    messageTotalFiles.Add(new
ZFrame("EvaluationEngine"));
                                    break;
                                case "EvaluationEngine":
                                    messageTotalFiles.Add(new
ZFrame("MarketplaceResolutionManager"));
                                    break;
                            }
                        messageTotalFiles.Add(new ZFrame("TotalFiles" +
FilesSent));
                        sendQueue.SendMessage(messageTotalFiles);
                        break;
                    }
                }
                Console.WriteLine(this.GetType().Name + ": " + filesToBeReceived +
" -> " + FilesSent);
            }

        }
    }

    public class PriceAggregator : ResolutionBase
    {
```

```
        public PriceAggregator(string receiveQueueAddress, string
sendQueueAddress, Architectures architecture, int rcvHwm, int arraySize)
            : base(receiveQueueAddress, sendQueueAddress, architecture, rcvHwm,
arraySize)
        {
            this.backgroundWorker.DoWork += backgroundWorker_DoWork;
            this.backgroundWorker.RunWorkerAsync();
        }
        protected override int CreateAndSendFiles(XmlDocument xmlIn, ZSocket
sendQueue)
        {
            if (xmlIn == null)
                return 0;
            // Delay
            Delay();
            int databasePrice, operationSystemPrice, crmPrice, networkPrice,
maxPrice = 0;
            if
(!Int32.TryParse(xmlIn.SelectSingleNode("/XML/Business/PDatabase/Price").InnerText
, out databasePrice)
                ||
!Int32.TryParse(xmlIn.SelectSingleNode("/XML/Business/POperationSystem/Price").Inn
erText, out operationSystemPrice)
                ||
!Int32.TryParse(xmlIn.SelectSingleNode("/XML/Business/PCRM/Price").InnerText, out
crmPrice)
                ||
!Int32.TryParse(xmlIn.SelectSingleNode("/XML/Business/PNetwork/Price").InnerText,
out networkPrice))
                return 0; // If a price is missing or cannot be parsed, do not
proceed. REVIEW
            int totalPrice = databasePrice + operationSystemPrice + crmPrice +
networkPrice;
            if
((xmlIn.SelectSingleNode("/XML/UserRequirements/Optimization/MAXPrice") != null)
                &&
(!Int32.TryParse(xmlIn.SelectSingleNode("/XML/UserRequirements/Optimization/MAXPri
ce").InnerText, out maxPrice)
                    || (totalPrice > maxPrice)))
                return 0; // If the total price of the resolution is higher than
the user specified price, reject it.

            // Create and send price resolution.
            XmlDocument xmlOut = new XmlDocument();
            xmlOut.LoadXml(xmlIn.OuterXml);
            XmlElement totalPriceNode = xmlOut.CreateElement("TotalPrice");
            totalPriceNode.InnerText = totalPrice.ToString();
            xmlOut.SelectSingleNode("/XML").AppendChild(totalPriceNode);
            if (Architecture != Architectures.NoQueue)
                using (ZMessage message = new ZMessage())
                {
                    switch (Architecture)
                    {
                        case Architectures.SeparateQueues:
                            message.Add(new ZFrame(xmlOut.OuterXml));
                            break;
                        case Architectures.CommonQueue:
                            message.Add(new ZFrame("EvaluationEngine"));
                            message.Add(new ZFrame(xmlOut.OuterXml));
                            break;
                    }
                    sendQueue.Send(message);
```

```
                }
            else
                this.filesToBeSentList.Add(xmlOut.OuterXml);
            return 1;
        }
        protected void Delay()
        {
            if (ArraySize <= 0)
                return;
            int d1 = ArraySize, d2 = ArraySize, d3 = ArraySize;
            int[,] factor1 = new int[d1, d2];
            int[,] factor2 = new int[d2, d3];
            int[,] product = new int[d1, d3];
            Random random = new Random();
            for (int i = 0; i < factor1.GetLength(0); i++)
                for (int j = 0; j < factor1.GetLength(1); j++)
                    factor1[i, j] = random.Next();
            for (int i = 0; i < factor2.GetLength(0); i++)
                for (int j = 0; j < factor2.GetLength(1); j++)
                    factor2[i, j] = random.Next();
            for (int i = 0; i < factor1.GetLength(0); i++)
                for (int j = 0; j < factor2.GetLength(1); j++)
                {
                    product[i, j] = 0;
                    for (int k = 0; k < d2; k++)
                        product[i, j] += factor1[i, k] * factor2[k, j];
                }
            return;
        }
        protected void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
        {
            using (ZContext context = new ZContext())
            using (ZSocket receiveQueue = new ZSocket(context,
ZSocketType.DEALER),
                           sendQueue = new ZSocket(context, ZSocketType.DEALER))
            {
                int filesToBeReceived = 0;
                // Initialize queues
                sendQueue.SendHighWatermark = 2;
                if (!String.IsNullOrWhiteSpace(SendQueueAddress))
                    switch (Architecture)
                    {
                        case Architectures.SeparateQueues:
                        case Architectures.NoQueue:
                            receiveQueue.ReceiveHighWatermark = RcvHwm;
                            sendQueue.Bind(SendQueueAddress);
                            break;
                        case Architectures.CommonQueue:
                            sendQueue.Connect(SendQueueAddress);
                            receiveQueue.ReceiveHighWatermark = 2;
                            receiveQueue.IdentityString = this.GetType().Name;
                            break;
                    }
                receiveQueue.Connect(ReceiveQueueAddress);
                Console.WriteLine(this.GetType().Name + " ready.");
                while (true)
                {
                    // Receive message.
                    string messageString;
                    using (ZFrame frame = receiveQueue.ReceiveMessage()[0])
                    {
                        messageString = frame.ReadString();
```

```
                    }
                    if (messageString.StartsWith("TotalFiles"))
                        filesToBeReceived =
Int32.Parse(messageString.Substring("TotalFiles".Length));
                    else
                    {
                        XmlDocument xmlIn = new XmlDocument();
                        xmlIn.LoadXml(messageString);
                        FilesReceived++;
                        FilesSent += CreateAndSendFiles(xmlIn, sendQueue);
                    }
                    if (FilesReceived == filesToBeReceived)
                    {
                        if (Architecture == Architectures.NoQueue)
                            foreach (string xmlOut in this.filesToBeSentList)
                                using (ZFrame frame = new ZFrame(xmlOut))
                                using (ZMessage message = new ZMessage(new
ZFrame[] { frame }))
                                {
                                    sendQueue.SendMessage(message);
                                }
                        ZMessage messageTotalFiles = new ZMessage();
                        if (Architecture == Architectures.CommonQueue)
                            switch (this.GetType().Name)
                            {
                                case "TechnicalResolution":
                                    messageTotalFiles.Add(new
ZFrame("BusinessResolution"));
                                    break;
                                case "BusinessResolution":
                                    messageTotalFiles.Add(new
ZFrame("PriceAggregator"));
                                    break;
                                case "PriceAggregator":
                                    messageTotalFiles.Add(new
ZFrame("EvaluationEngine"));
                                    break;
                                case "EvaluationEngine":
                                    messageTotalFiles.Add(new
ZFrame("MarketplaceResolutionManager"));
                                    break;
                            }
                        messageTotalFiles.Add(new ZFrame("TotalFiles" +
FilesSent));
                        sendQueue.SendMessage(messageTotalFiles);
                        break;
                    }
                }
                Console.WriteLine(this.GetType().Name + ": " + filesToBeReceived +
" -> " + FilesSent);
            }
        }
    }

    public class EvaluationEngine : ResolutionBase
    {
        private XmlDocument bestResolution;
        private int bestTotal;
        public EvaluationEngine(string receiveQueueAddress, string
sendQueueAddress, Architectures architecture, int rcvHwm, int arraySize)
            : base(receiveQueueAddress, sendQueueAddress, architecture, rcvHwm,
arraySize)
```

```
        {
            this.backgroundWorker.DoWork += backgroundWorker_DoWork;
            this.backgroundWorker.RunWorkerCompleted +=
backgroundWorker_RunWorkerCompleted;
            this.bestTotal = Int32.MinValue;
            this.backgroundWorker.RunWorkerAsync();
        }

        protected override int CreateAndSendFiles(XmlDocument xmlIn, ZSocket
sendQueue)
        {
            // Delay
            Delay();
            // Check optimization requirements.
            int minAvailability = 0, minSecurity = 0, minReputation = 0,
minReliability = 0;
            int avgAvailability = 0, avgSecurity = 0, avgReputation = 0,
avgReliability = 0, avgConfidentiality = 0;
            bool requestedConfidentiality = false;

Int32.TryParse(xmlIn.SelectSingleNode("/XML/UserRequirements/Optimization/MINAvail
ability").InnerText, out minAvailability);

Int32.TryParse(xmlIn.SelectSingleNode("/XML/UserRequirements/Optimization/MINSecur
ity").InnerText, out minSecurity);

Int32.TryParse(xmlIn.SelectSingleNode("/XML/UserRequirements/Optimization/MINReput
ation").InnerText, out minReputation);

Int32.TryParse(xmlIn.SelectSingleNode("/XML/UserRequirements/Optimization/MINRelia
bility").InnerText, out minReliability);
            if
((xmlIn.SelectSingleNode("/XML/UserRequirements/Optimization/Confidentiality") !=
null)
                &&
(xmlIn.SelectSingleNode("/XML/UserRequirements/Optimization/Confidentiality").Inne
rText.Trim().ToUpper() == "YES"))
                requestedConfidentiality = true;
            bool constraintsMet = true;
            XmlNodeList nodeList;
            // Check availability requirements.
            nodeList = xmlIn.SelectNodes("/XML/Business//Availability");
            foreach (XmlNode node in nodeList)
            {
                int availability;
                if (!Int32.TryParse(node.InnerText, out availability)
                    || (availability < minAvailability))
                {
                    constraintsMet = false;
                    break;
                }
                avgAvailability += availability / nodeList.Count;
            }
            if (!constraintsMet)
                return 0; // If any constraint is not met, ignore the resolution.
            // Check security requirements.
            nodeList = xmlIn.SelectNodes("/XML/Business//Security");
            foreach (XmlNode node in nodeList)
            {
                int security;
                if (!Int32.TryParse(node.InnerText, out security)
                    || (security < minSecurity))
```

```
                {
                    constraintsMet = false;
                    break;
                }
                avgSecurity += security / nodeList.Count;
            }
            if (!constraintsMet)
                return 0; // If any constraint is not met, ignore the resolution.
            // Check reputation requirements.
            nodeList = xmlIn.SelectNodes("/XML/Business//Reputation");
            foreach (XmlNode node in nodeList)
            {
                int reputation;
                if (!Int32.TryParse(node.InnerText, out reputation)
                    || (reputation < minReputation))
                {
                    constraintsMet = false;
                    break;
                }
                avgReputation += reputation / nodeList.Count;
            }
            if (!constraintsMet)
                return 0; // If any constraint is not met, ignore the resolution.
            // Check reliability requirements.
            nodeList = xmlIn.SelectNodes("/XML/Business//Reliability");
            foreach (XmlNode node in nodeList)
            {
                int reliability;
                if (!Int32.TryParse(node.InnerText, out reliability)
                    || (reliability < minReliability))
                {
                    constraintsMet = false;
                    break;
                }
                avgReliability += reliability / nodeList.Count;
            }
            if (!constraintsMet)
                return 0; // If any constraint is not met, ignore the resolution.
            // Check confidentiality requirements
            nodeList = xmlIn.SelectNodes("/XML/Business//Confidentiality");
            foreach (XmlNode node in nodeList)
            {
                if (requestedConfidentiality
                    && (node.InnerText.Trim().ToUpper() != "YES"))
                {
                    constraintsMet = false;
                    break;
                }
                if (node.InnerText.Trim().ToUpper() == "YES")
                    avgConfidentiality++;
            }
            if (!constraintsMet)
                return 0; // If any constraint is not met, ignore the resolution.

            // Overall evaluation
            int totalPrice;
            if
(!Int32.TryParse(xmlIn.SelectSingleNode("/XML/TotalPrice").InnerText, out
totalPrice))
                return 0;
            int total = 2000 * (avgAvailability / 10 + avgSecurity + avgReputation
+ avgReliability / 2 + avgConfidentiality / 2) - totalPrice;
```

```
            if (total <= this.bestTotal)
                return 0; // If the current resolution isn't any better, discard
it

            XmlDocument resolution = new XmlDocument();
            resolution.LoadXml(xmlIn.OuterXml);
            // Add info to the resolution
            XmlElement element;
            // Availability
            element = resolution.CreateElement("TotalAvailability");
            element.InnerText = avgAvailability.ToString();
            resolution.SelectSingleNode("/XML").AppendChild(element);
            // Security
            element = resolution.CreateElement("TotalSecurity");
            element.InnerText = avgSecurity.ToString();
            resolution.SelectSingleNode("/XML").AppendChild(element);
            // Reputation
            element = resolution.CreateElement("TotalReputation");
            element.InnerText = avgReputation.ToString();
            resolution.SelectSingleNode("/XML").AppendChild(element);
            // Reliability
            element = resolution.CreateElement("TotalReliability");
            element.InnerText = avgReliability.ToString();
            resolution.SelectSingleNode("/XML").AppendChild(element);
            // Confidentiality
            element = resolution.CreateElement("TotalConfidentiality");
            element.InnerText = avgConfidentiality.ToString();
            resolution.SelectSingleNode("/XML").AppendChild(element);
            // Reposition TotalPrice

resolution.SelectSingleNode("/XML").AppendChild(resolution.SelectSingleNode("/XML/
TotalPrice"));
            // TotalTotal
            element = resolution.CreateElement("TotalTotal");
            element.InnerText = total.ToString();
            resolution.SelectSingleNode("/XML").AppendChild(element);
            int returnValue = (this.bestResolution == null) ? 1 : 0;
            this.bestTotal = total;
            this.bestResolution = resolution;
            return returnValue;
        }
        private void backgroundWorker_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
        {
            if (this.bestResolution == null)
                return;

this.bestResolution.Save(@"C:\Users\Theodore\Documents\Διπλωματική\Κώδικας\Έξοδος\
Test8\" + this.bestTotal + ".xml");
        }
        protected void Delay()
        {
            if (ArraySize <= 0)
                return;
            int d1 = ArraySize, d2 = ArraySize, d3 = ArraySize;
            int[,] factor1 = new int[d1, d2];
            int[,] factor2 = new int[d2, d3];
            int[,] product = new int[d1, d3];
            Random random = new Random();
            for (int i = 0; i < factor1.GetLength(0); i++)
                for (int j = 0; j < factor1.GetLength(1); j++)
                    factor1[i, j] = random.Next();
```

```
            for (int i = 0; i < factor2.GetLength(0); i++)
                for (int j = 0; j < factor2.GetLength(1); j++)
                    factor2[i, j] = random.Next();
            for (int i = 0; i < factor1.GetLength(0); i++)
                for (int j = 0; j < factor2.GetLength(1); j++)
                {
                    product[i, j] = 0;
                    for (int k = 0; k < d2; k++)
                        product[i, j] += factor1[i, k] * factor2[k, j];
                }
            return;
        }
        protected void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
        {
            using (ZContext context = new ZContext())
            using (ZSocket receiveQueue = new ZSocket(context,
ZSocketType.DEALER),
                           sendQueue = new ZSocket(context, ZSocketType.DEALER))
            {
                int filesToBeReceived = 0;
                // Initialize queues
                sendQueue.SendHighWatermark = 2;
                if (!String.IsNullOrWhiteSpace(SendQueueAddress))
                    switch (Architecture)
                    {
                        case Architectures.SeparateQueues:
                        case Architectures.NoQueue:
                            receiveQueue.ReceiveHighWatermark = RcvHwm;
                            sendQueue.Bind(SendQueueAddress);
                            break;
                        case Architectures.CommonQueue:
                            sendQueue.Connect(SendQueueAddress);
                            receiveQueue.ReceiveHighWatermark = 2;
                            receiveQueue.IdentityString = this.GetType().Name;
                            break;
                    }
                receiveQueue.Connect(ReceiveQueueAddress);
                Console.WriteLine(this.GetType().Name + " ready.");
                while (true)
                {
                    // Receive message.
                    string messageString;
                    using (ZFrame frame = receiveQueue.ReceiveMessage()[0])
                    {
                        messageString = frame.ReadString();
                    }
                    if (messageString.StartsWith("TotalFiles"))
                        filesToBeReceived =
Int32.Parse(messageString.Substring("TotalFiles".Length));
                    else
                    {
                        XmlDocument xmlIn = new XmlDocument();
                        xmlIn.LoadXml(messageString);
                        FilesReceived++;
                        FilesSent += CreateAndSendFiles(xmlIn, sendQueue);
                    }
                    if (FilesReceived == filesToBeReceived)
                    {
                        if (Architecture == Architectures.NoQueue)
                            foreach (string xmlOut in this.filesToBeSentList)
                                using (ZFrame frame = new ZFrame(xmlOut))
```

```
                                using (ZMessage message = new ZMessage(new
ZFrame[] { frame }))
                                {
                                    sendQueue.SendMessage(message);
                                }
                            ZMessage messageResolution = new ZMessage();
                            if (Architecture == Architectures.CommonQueue)
                                switch (this.GetType().Name)
                                {
                                    case "TechnicalResolution":
                                        messageResolution.Add(new
ZFrame("BusinessResolution"));
                                        break;
                                    case "BusinessResolution":
                                        messageResolution.Add(new
ZFrame("PriceAggregator"));
                                        break;
                                    case "PriceAggregator":
                                        messageResolution.Add(new
ZFrame("EvaluationEngine"));
                                        break;
                                    case "EvaluationEngine":
                                        messageResolution.Add(new
ZFrame("MarketplaceResolutionManager"));
                                        break;
                                }
                            //messageTotalFiles.Add(new ZFrame("TotalFiles" +
FilesSent));
                            if (this.bestResolution == null)
                                messageResolution.Add(new ZFrame("No services are
available for the specified user requirements."));
                            else
                                messageResolution.Add(new
ZFrame(this.bestResolution.OuterXml));
                            sendQueue.SendMessage(messageResolution);
                            break;
                        }
                    }
                    Console.WriteLine(this.GetType().Name + ": " + filesToBeReceived +
" -> " + FilesSent);
                }
            }
        }
        public abstract class ResolutionBase
        {
            public string ReceiveQueueAddress { get; private set; }
            public string SendQueueAddress { get; private set; }
            public Architectures Architecture { get; private set; }
            public int RcvHwm { get; private set; }
            public int ArraySize { get; set; }
            protected List<string> filesToBeSentList;
            protected int FilesReceived;
            protected int FilesSent;
            private int filesReceived;
            private int filesSent;
            protected BackgroundWorker backgroundWorker;

            protected ResolutionBase(string receiveQueueAddress, string
sendQueueAddress, Architectures architecture, int rcvHwm, int arraySize)
            {
                ReceiveQueueAddress = receiveQueueAddress;
                SendQueueAddress = sendQueueAddress;
```

```
            Architecture = architecture;
            RcvHwm = rcvHwm;
            ArraySize = arraySize;
            FilesReceived = 0;
            FilesSent = 0;
            this.filesToBeSentList = new List<string>();
            this.backgroundWorker = new BackgroundWorker();
        }

        protected abstract int CreateAndSendFiles(XmlDocument xmlIn, ZSocket
sendQueue);
    }
}
```

Τέλος, η `Main` βρίσκεται στο αρχείο **Program.cs**:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MarketplaceForCloud1
{
    static class ResolutionManager
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FrontEnd());
        }
    }
}
```