Τ.Ε.Ι ΚΑΛΑΜΑΤΑΣ ΠΑΡΑΡΤΗΜΑ ΣΠΑΡΤΗΣ

# Υλοποίηση αναδρομικών αλγορίθμων για παράλληλη εκτέλεσή τους σε πλέγμα (grid) ή cloud υπολογιστών χρησιμοποιώντας τεχνολογίες Java.

---

**Βασίλης Ντοχάς, Ιωάννης Ροζής**

# Περίληψη

Αντικείμενο αυτής της πτυχιακής εργασίας είναι η μελέτη και η εφαρμογή των τεχνολογιών cloud, τεχνολογιών δηλαδή, που επιτρέπουν τον ελεγχόμενο διαμοιρασμό πόρων, σε απομακρυσμένα ή και τοπικά υπολογιστικά συστήματα με σκοπό τη δημιουργία ενός εικονικού, πολύ ισχυρού υπολογιστή.

Αρχικά γίνεται μια εισαγωγή στο cloud computing όπου αναφέρονται και αναλύονται οι κατηγορίες που το αποτελούν, η αρχιτεκτονική του, οι δυνατότητές του, τα αίτια εξέλιξης του, οι κύριες χρήσης του καθώς και σύγκριση μεταξύ των κατηγοριών. Στη συνέχεια αναλύονται δυο κύριες μέθοδοι ανάπτυξης σε πλέγμα. Στο κεφάλαιο 3 γίνεται μια σύγκριση ανάμεσα στα δυο πιο γνωστά cloud middleware το Hadoop και το GridGain και στη συνεχεία αναπτύσσουμε πως δομείται και πια χαρακτηριστικά έχει το middleware που επιλέξαμε βάση της σύγκρισης που κάναμε. Ακόμη αναλύουμε την υλοποίηση του αλγορίθμου γρήγορης ταξινόμησης(quicksort) και στο τέλος παραθέτουμε την υλοποίηση της γρήγορης ταξινόμησης στο πλέγμα καθώς και σύγκριση της με την υλοποίηση του αλγορίθμου για πολλούς επεξεργαστές.

# Κεφάλαιο 1

# Cloud Computing

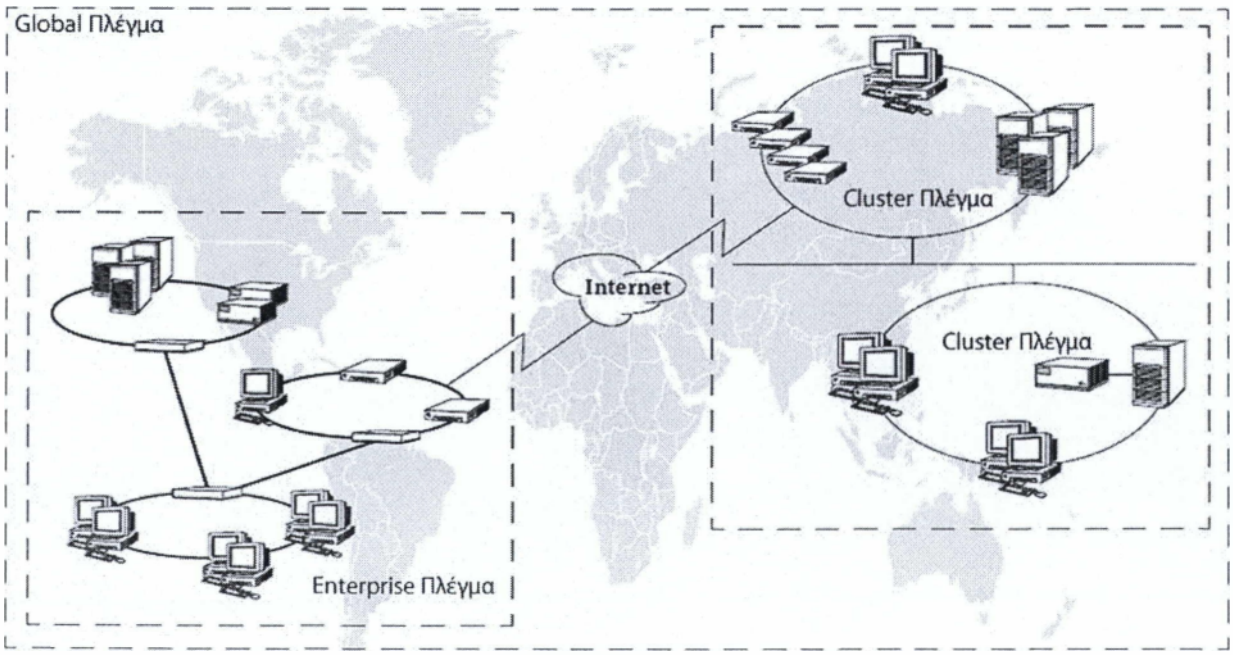
# 1.1 Cloud Computing

Το cloud computing είναι ένα νέο συμπληρωματικό μοντέλο υπηρεσιών πληροφορικής που βασίζονται στο διαδίκτυο, και κατά κανόνα προσφέρει μέσω του διαδικτύου δυναμικά επεκτάσιμους εικονικούς πόρους. Είναι ένα προϊόν που παρέχει εύκολη πρόσβαση σε απομακρυσμένες τοποθεσίες υπολογιστών που παρέχονται από το διαδίκτυο. Αυτό γίνεται με τη μορφή web-based εργαλείων ή εφαρμογών που οι χρήστες μπορούν να έχουν πρόσβαση μέσω ενός web browser σαν ήταν ένα πρόγραμμα που εγκαθίσταται τοπικά στον υπολογιστή τους. Το cloud computing μπορεί να χαρακτηριστεί απόγονος του πλέγματος.

## 1.2.1 Ιδιωτικό Cloud(Πλέγμα)

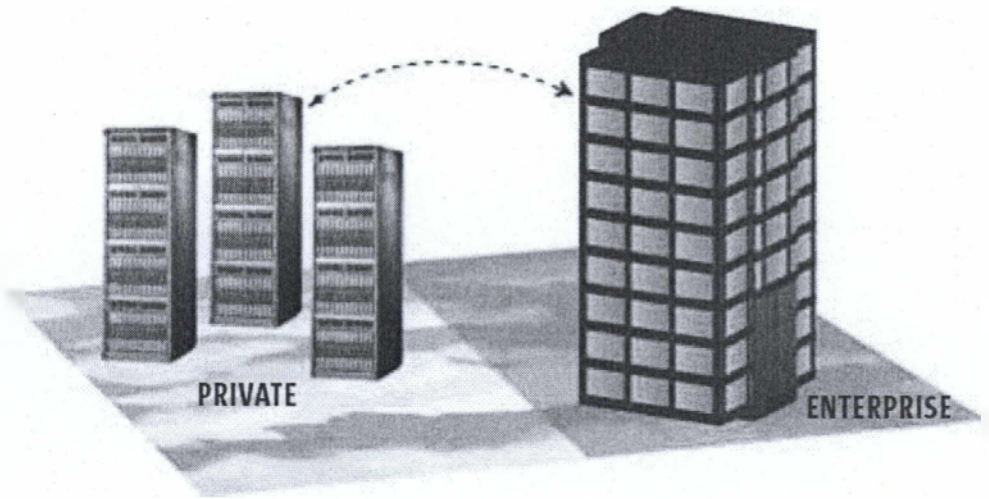
Το πλέγμα είναι μια συλλογή από υπολογιστικούς πόρους. Στην απλή του μορφή το πλέγμα φαίνεται στους χρηστές σαν ένα μεγάλο σύστημα που παρέχει ένα σημείο πρόσβασης σε μεγάλους κατανεμημένους πόρους. Υπάρχουν τρεις τύποι πλεγμάτων:

- Τα Cluster πλέγματα είναι η πιο απλή κατηγορία. Τα Cluster πλέγματα είναι φτιαγμένα από ένα σύνολο υπολογιστών που φιλοξενούν τις εργασίες σαν ένα σύνολο. Παρέχουν ένα σημείο πρόσβασης στους χρηστές με ένα project.
- Τα Enterprise πλέγματα έχουν τη δυνατότητα να φιλοξενήσουν πολλά projects μέσα σ' ένα οργανισμό μοιράζοντας υπολογιστικούς πόρους. Οι οργανισμοί μπορούν να χρησιμοποιήσουν τα Enterprise πλέγματα για διάφορες εργασίες συμπεριλαμβανομένου του data mining και την επεξεργασία frames.
- Τα Global πλέγματα είναι μια συλλογή από Enterprise πλέγματα που επεκτείνονται έξω από τα όρια της επιχείρησης δημιουργώντας μεγάλα εικονικά συστήματα. Οι χρηστές έχουν πρόσβαση όταν οι υπολογιστικοί πόροι της επιχείρησης εξαντληθούν. [1]




Εικόνα 1 [2]

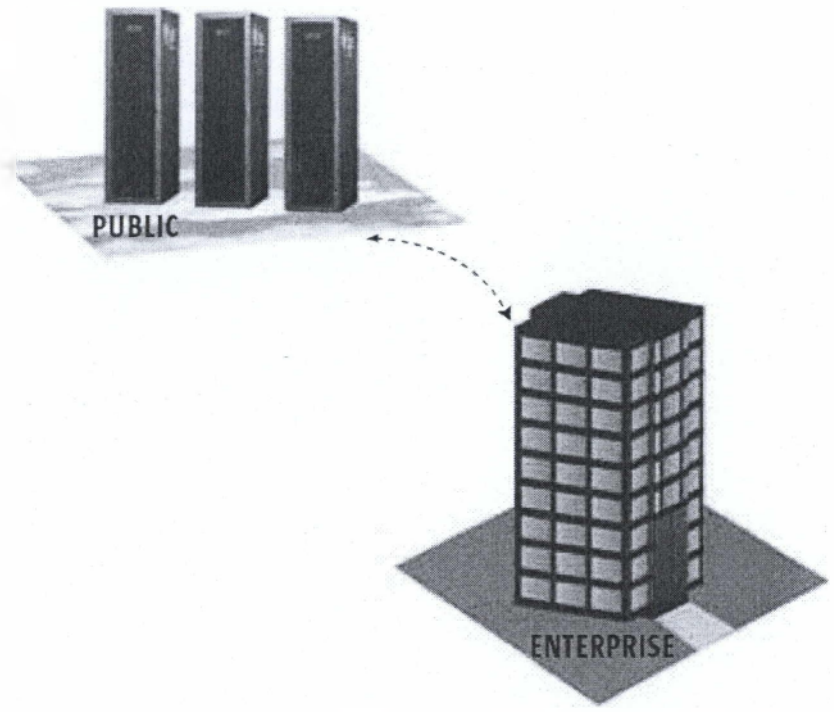
Οι περισσότερες ομάδες ανάπτυξης εφαρμογών δεν αναφέρονται στο πλέγμα όσο παλιότερα, με τις εφαρμογές που έχουν ανάγκη για μεγάλη υπολογιστική δύναμη να είναι προσκολλημένες σε αυτό. Ο περιορισμός αυτός έχει οδηγήσει τους διαχειριστές και τους πάροχους στην αναζήτηση λύσης ώστε να επεκτείνουν τις υποδομές τους με χαμηλό κόστος. Το πρόβλημα με αυτό είναι ότι το πλέγμα είναι κυρίως στατικό μη παρέχοντας την κατάλληλη ευελιξία. Λύση σε αυτό το πρόβλημα έρχεται να δώσει το ιδιωτικό cloud μετατρέποντας το πλέγμα από στατικό σε δυναμικό δίνοντας την επιθυμητή ευελιξία. [3] Τα ιδιωτικά cloud κατασκευάζονται για την αποκλειστική χρήση από ένα (client) πελάτη, παρέχοντας μεγαλύτερο έλεγχο των δεδομένων, ασφάλεια και ποιότητα των υπηρεσιών. Η εταιρία ενός ιδιωτικού cloud έχει τον έλεγχο σε αυτό μέσω δικών της προσαρμοσμένων εφαρμογών (ειδικά φτιαγμένες για το σκοπό αυτό). Βρίσκουν χρήση συνήθως σε datacenter. [4]




**Εικόνα 2 [4]**

## 1.2.2 Δημόσιο Σύννεφο

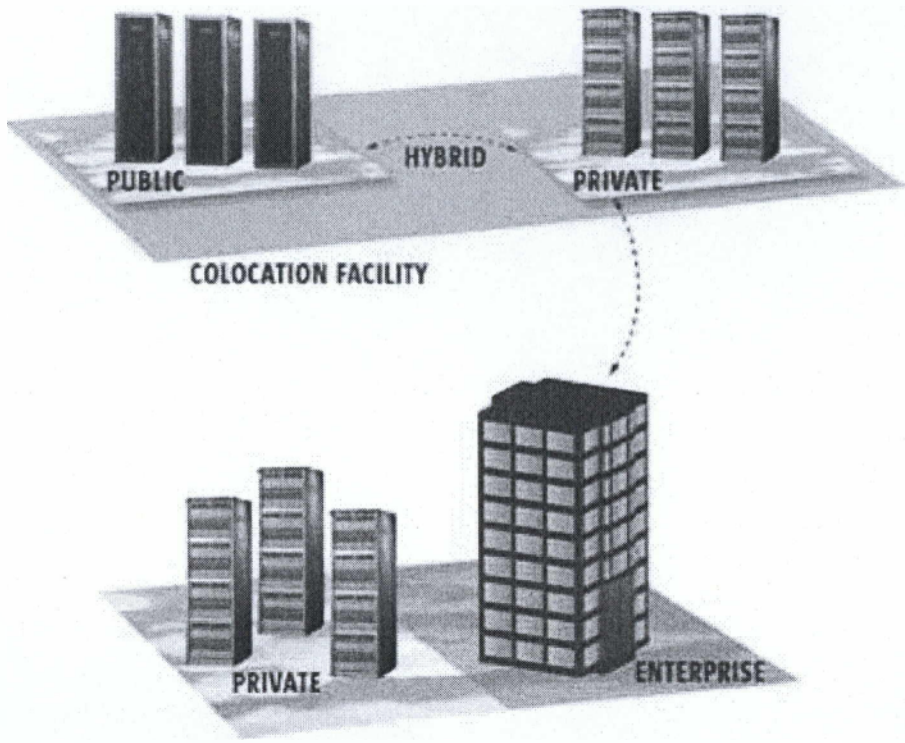
Τα δημόσια cloud μπορούν να εκτελέσουν εφαρμογές από διαφορετικούς πελάτες συνδυάζοντας τις δυνατότητες που παρέχουν οι cloud servers, τα συστήματα αποθήκευσης και τα δίκτυα (Εικόνα 3 [4]). Τα δημόσια clouds συχνά βρίσκονται σε διαφορετικές τοποθεσίες από τους πελάτες, παρέχοντας έναν τρόπο ώστε να μειωθεί ο κινδύνου και το κόστος που έχουν οι επιχειρήσεις, δίνοντας έτσι μια πιο ευέλικτη και ίσως προσωρινή επέκταση της υποδομής της επιχείρησης. [4]




Εικόνα 3 [4]

### 1.2.3 Υβριδικό Σύννεφο

Τα υβριδικά cloud είναι ένας συνδυασμός από δημόσια και ιδιωτικά cloud (Εικόνα 4 [4]). Συμβάλουν στην παροχή μιας άμεσης εξωτερικής επέκτασης όποτε αυτή χρειαστεί. Έχει την δυνατότητα να αυξάνει τους πόρους ενός ιδιωτικού cloud με τους πόρους ενός δημόσιου cloud. Αυτό μπορεί να χρησιμοποιηθεί για τη διατήρηση των υπηρεσιών σε περίπτωση που ο φόρτος εργασίας υπερβεί τα επιτρεπόμενα όρια. Η πιο συχνή χρήση του γίνεται στα αποθηκευτικά clouds για την υποστήριξη εφαρμογών Web 2.0. Ένα υβριδικό cloud μπορεί επίσης να χρησιμοποιηθεί για να χειριστεί ένα προγραμματισμένο φόρτο εργασίας. [4]


PUBLIC
HYBRID
PRIVATE
COLOCATION FACILITY
PRIVATE
ENTERPRISE

**Εικόνα 4 [4]**

# 1.3 Private Cloud (Πλέγμα)

## 1.3.1 Τι είναι το πλέγμα

Το πλέγμα(Grid) αποτελεί την εξέλιξη των κατανεμημένων συστημάτων. Ο στόχος του πλέγματος είναι η δημιουργία ενός εικονικού, πολύ ισχυρού υπολογιστή που αποτελείται από πολλά διασυνδεδεμένα ετερογενή συστήματα τα οποία μοιράζονται πολλά είδη πόρων. Η δημιουργία κοινών προτύπων για το μοίρασμα των πόρων στο πλέγμα με δεδομένη τη διαθεσιμότητα υψηλού εύρους ζώνης για την επικοινωνία των συστημάτων, έχει οδηγήσει σε ένα επαναστατικό βήμα στην πληροφορική εξίσου σημαντικού με το διαδίκτυο που επίσης προήλθε από την προτυποποίηση στον τομέα των επικοινωνιών για τη διασύνδεση ετερογενών συστημάτων.

## 1.3.2 Η αρχιτεκτονική του Πλέγματος

Τα υπολογιστικά πλέγματα ορίζονται ως συστήματα που συντονίζουν κατανεμημένους πόρους κάνοντας χρήση τυποποιημένων, ανοικτών και γενικού-σκοπού πρωτόκολλων και διεπαφών, για να παραδώσουν αξιόλογης ποιότητας υπηρεσίες.

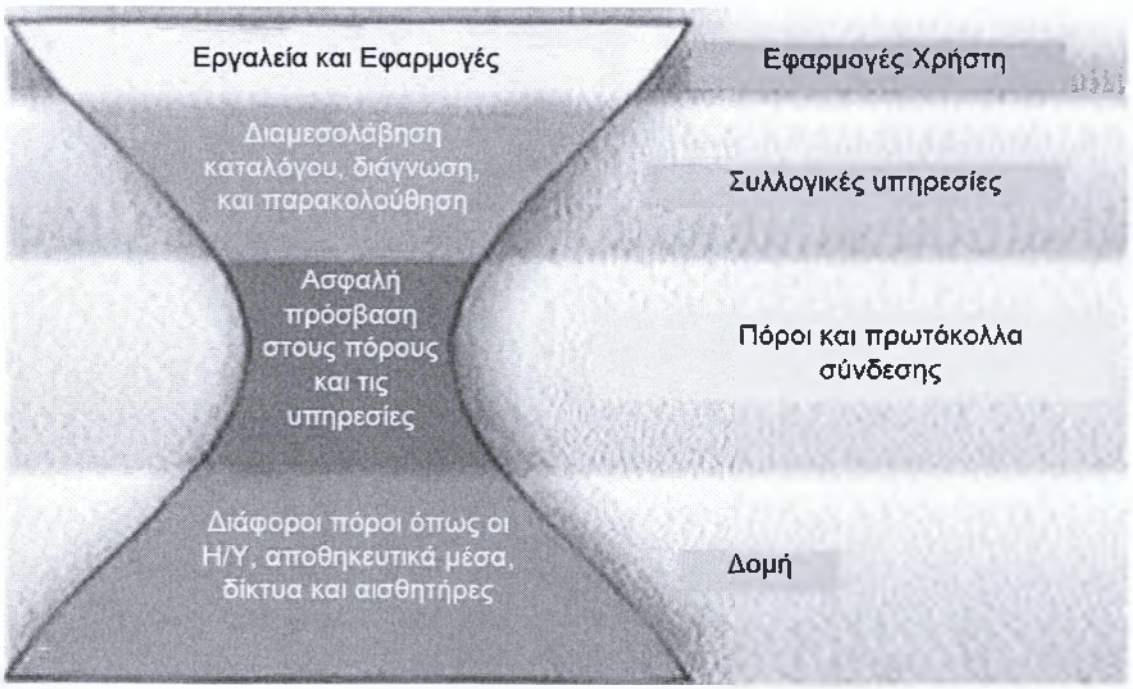
- Συντονισμός κατανεμημένων πόρων: Ένα υπολογιστικό πλέγμα ενώνει και συντονίζει πόρους και χρήστες που βρίσκονται σε διαφορετικές γεωγραφικές περιοχές, και διαχειρίζεται θέματα ασφάλειας, άδειας, πληρωμής, συμμετοχής, και άλλων που προκύπτουν.
- Χρήση τυποποιημένων, ανοικτών και γενικού-σκοπού πρωτοκόλλων και διεπαφών: Ένα υπολογιστικό πλέγμα κατασκευάζεται από πρωτόκολλα και διεπαφές πολλαπλών σκοπών, τα οποία έχουν να κάνουν με θέματα, όπως η πιστοποίηση αυθεντικότητας, εξουσιοδότησης, ανακάλυψη πόρων και πρόσβαση σε αυτούς. Τα πρωτόκολλα αυτά και οι διεπαφές είναι αναγκαίο να είναι ανοικτές και τυποποιημένες, αλλιώς θα έχουμε να κάνουμε με ένα σύστημα εξαρτώμενο από την εκάστοτε εφαρμογή.
- Να παραδώσει αξιόλογες ποιότητες υπηρεσιών: Ένα υπολογιστικό πλέγμα επιτρέπει να χρησιμοποιηθούν οι πόροι που το συνθέτουν με συντονισμένο τρόπο, ώστε να προσφέρει ποιοτικές υπηρεσίες (π.χ. σχετικά με το χρόνο απόκρισης, την απόδοση, τη διαθεσιμότητα, την ασφάλεια), ή και συν-κατανομή διαφορετικών τύπων πόρων, με σκοπό να ικανοποιηθούν οι απαιτήσεις των χρηστών.

Ένα υπολογιστικό πλέγμα δεν είναι μόνο μία συλλογή από υπολογιστικούς πόρους. Είναι επίσης ένα σύνολο υπηρεσιών και πρωτοκόλλων, που στοχεύουν στη συλλογή πληροφοριών , στον εντοπισμό των διαθέσιμων υπολογιστικών πόρων και τη δρομολόγηση των εργασιών, στην επικοινωνία, στη δυνατότητα πρόσβασης στον κώδικα και τα δεδομένα, στην παρακολούθηση της επίδοσης, στην πιστοποίηση των χρηστών και των πόρων, στην εξασφάλιση της ασφάλειας στις επικοινωνίες κλπ. Η πρόσβαση στις υπηρεσίες και στα πρωτόκολλα επιτυγχάνεται μέσω APIs και SDKs. Έτσι με αυτόν τον τρόπο απλοποιείται η ανάπτυξη των εφαρμογών, ορίζοντας μία εικονική μηχανή, την οποία είναι ευκολότερη να χειριστεί ο προγραμματιστής, απ' ότι ένα χαμηλότερου επιπέδου υλικό, όπως ένα τυπικό λειτουργικό σύστημα και οι υπηρεσίες που υλοποιεί διευκολύνουν τη σχεδίαση του

λογισμικού. Οι υπηρεσίες και οι διεπαφές, μαζί με τα αντίστοιχα APIs και SDKs τους, ορίζουν την αρχιτεκτονική των υπολογιστικών πλεγμάτων.

Η αρχιτεκτονικής του πλέγματος ακολουθεί τη λογική των "στρωμάτων", όπου το κάθε στρώμα παρέχει μία συγκεκριμένη λειτουργία. Τα υψηλότερα στρώματα επικεντρώνονται στο χρήστη, ενώ τα χαμηλότερα στα υπολογιστικά συστήματα και στα δίκτυα που τα συνδέουν.

Για τον προσδιορισμό των διαφόρων στρωμάτων ακολουθούνται οι αρχές του "μοντέλου κλεψύδρας". Ο στενός λαιμός της κλεψύδρας περιέχει ένα μικρό σύνολο αφαιρετικών πυρήνων και πρωτοκόλλων (όπως αντίστοιχα τα HTTP, και TCP στην αρχιτεκτονική του διαδικτύου), τα οποία μπορούν να χρησιμοποιηθούν από πληθώρα διαφορετικών υλοποιήσεων υψηλότερων επιπέδων, και να αλληλεπιδράσουν με πολλές διαφορετικές τεχνολογίες κατώτερων στρωμάτων. Ο αριθμός των πρωτοκόλλων στο λαιμό της κλεψύδρας πρέπει να είναι μικρός. Στην αρχιτεκτονική, ο λαιμός της κλεψύδρας αποτελείται από το στρώμα πόρων και το στρώμα συνδεσιμότητας τα οποία διευκολύνουν την κατανομή μεμονωμένων πόρων. Τα πρωτόκολλα σε αυτό το επίπεδο είναι σχεδιασμένα έτσι , ώστε να μπορούν να χρησιμοποιηθούν από διαφορετικούς πόρους, τα οποία ορίζονται στο πιο κάτω δομικό στρώμα, καθώς και να χρησιμοποιηθούν επίσης για να κατασκευαστεί ένα ευρύ φάσμα από υπηρεσίες και εξειδικευμένες εφαρμογές στο πιο πάνω συλλογικό στρώμα, το οποίο ονομάζεται έτσι επειδή περιλαμβάνει τη συντονισμένη (αθροιστική - συλλογιστική) χρήσης πολλαπλών πόρων. [5]




Εικόνα 5 [5]

### 1.3.2.1 Δομικό στρώμα (fabric layer)

Το δομικό στρώμα παρέχει τους πόρους στους οποίους η πρόσβαση επιτυγχάνεται από τη διαμεσολάβηση των πρωτοκόλλων του πλέγματος. Περιλαμβάνει υπολογιστικούς πόρους, συστήματα αποθήκευσης, καταλόγους, δικτυακούς πόρους και αισθητήρες. Ένας πόρος μπορεί να είναι μία λογική οντότητα, όπως ένα κατανεμημένο σύστημα αρχείων, ένας τομέας υπολογιστών (computer cluster), ή μία κατανεμημένη περιοχή υπολογιστών (distributed computer pool). Σε τέτοιες περιπτώσεις, μία υλοποίηση των στοιχείων των πόρων μπορεί να περιλαμβάνει εσωτερικά πρωτόκολλα (π.χ. το πρωτόκολλο πρόσβασης αποθήκευσης NFS ή κάποιο πρωτόκολλο διαχείρισης του συστήματος αρχείων). Διαδικασίες διανομής που υλοποιούνται σε υψηλότερα στρώματα της αρχιτεκτονικής αναθέτουν σε συγκεκριμένους κάθε φορά πόρους του δομικού στρώματος συγκεκριμένες, τοπικές διαδικασίες. Δημιουργείται έτσι μία αλληλεξάρτηση μεταξύ των λειτουργιών που υλοποιούνται στο δομικό στρώμα και των διαδικασιών διανομής που υποστηρίζονται. Όσο πιο πλούσιο είναι το δομικό στρώμα, τόσο πολυπλοκότερες διαδικασίες διανομής μπορούν να υπάρξουν. [5]

### 1.3.2.2 Στρώμα συνδεσιμότητας (connectivity layer)

Το στρώμα συνδεσιμότητας καθορίζει τα πρωτόκολλα επικοινωνίας και πιστοποίησης ταυτότητας που απαιτούνται για τις συγκεκριμένες δικτυακές συναλλαγές στο πλέγμα. Τα πρωτόκολλα επικοινωνίας επιτρέπουν την ανταλλαγή δεδομένων μεταξύ των πόρων του δομικού στρώματος. Τα πρωτόκολλα πιστοποίησης ταυτότητας είναι ενσωματωμένα στις υπηρεσίες επικοινωνίας για να παρέχουν κρυπτογραφημένους ασφαλείς μηχανισμούς για τη εξακρίβωση της ταυτότητας των χρηστών και πόρων. [5]

Το κομμάτι της επικοινωνίας περιλαμβάνει μεταφορά δεδομένων, αντιστοίχιση και διευθυνσιοδότηση. Ενώ υπάρχουν εναλλακτικές λύσεις, χρησιμοποιείται στα πλέγματα η στοίβα πρωτοκόλλων TCP/IP: ειδικότερα, η διευθυνσιοδότηση και ανακάλυψη πόρων (IP, ICMP), η μεταφορά (TCP, UDP), και η εφαρμογή (DNS, OSPF, RSVP, κλπ) που είναι στρώματα της αρχιτεκτονικής του διαδικτύου. [5]

### 1.3.2.3 Στρώμα πόρων (resource layer)

Το στρώμα πόρων εντοπίζεται πάνω από τα πρωτόκολλα επικοινωνίας και πιστοποίησης ταυτότητας του στρώματος συνδεσιμότητας και καθορίζει τα πρωτόκολλα (API's και SDK's) που χρειάζονται για την ασφαλή διαπραγμάτευση, αρχικοποίηση, παρακολούθηση, έλεγχο, χρέωση και πληρωμή των λειτουργιών για κάθε πόρο ξεχωριστά. Οι εφαρμογές αυτών των πρωτοκόλλων, καλούν τις λειτουργίες του δομικού στρώματος για την πρόσβαση και τον έλεγχο των τοπικών πόρων. Τα πρωτόκολλα του στρώματος πόρων ενδιαφέρονται για τον κάθε πόρο χωριστά, και αγνοούν τη συλλογική κατάσταση και τα θέματα λειτουργίας των πόρων εντός των κατανεμημένων συλλογών. Τα πρωτόκολλα αυτά χωρίζονται σε δύο κατηγορίες: [5]

1. Πρωτόκολλα πληροφορίας: Χρησιμοποιούνται για τη λήψη πληροφοριών για τη δομή και την κατάσταση ενός πόρου (π.χ. το τρέχον φορτίο εργασίας, και τη χρήση των υπολογιστικών κύκλων του). [5]

2. Πρωτόκολλα διαχείρισης: Χρησιμοποιούνται για τη διαχείριση της πρόσβασης σε έναν κοινό πόρο, προσδιορίζοντας τις απαιτήσεις που πρέπει να ικανοποιούν, καθώς και τις λειτουργίες που πρέπει να γίνουν σε αυτούς, όπως δημιουργία διεργασιών, ή πρόσβαση σε δεδομένα. [5]

#### 1.3.2.4 Συλλογικό στρώμα (collective layer)

Περιέχει τα πρωτόκολλα και τις διεπαφές (API's και SDK's) που επικεντρώνονται στις αλληλεπιδράσεις συλλογικών πόρων. Τα πρωτόκολλα αυτά εξαρτώνται από την εφαρμογή που τα χρησιμοποιεί και την περιοχή στην οποία εφαρμόζονται, και θα μπορούν να προσαρμοστούν στις απαιτήσεις μίας συγκεκριμένης κοινότητας χρηστών ή ενός εικονικού δικτύου. Για παράδειγμα είναι δυνατό κάποια από αυτά να εφαρμόζονται μόνο σε συγκεκριμένους εικονικούς οργανισμούς. Μερικά παραδείγματα υλοποιήσεων του στρώματος αυτού είναι τα εξής:

- Η κατανομή (co-allocation), η δρομολόγηση και οι υπηρεσίες ανάθεσης εργασιών στους πόρους (brokering services) επιτρέπουν στους χρήστες να ζητήσουν την κατανομή των πόρων σε ένα εικονικό δίκτυο και αφού βρεθεί η ιδανικότερη δρομολόγηση σε αυτό να ξεκινήσει η διαδικασία εκτέλεσης των εργασιών στους επιλεγμένους πόρους. [5]
- Διαγνωστικές υπηρεσίες και υπηρεσίες παρακολούθησης που υποστηρίζουν την παρακολούθηση των εικονικών πόρων για αποτυχία, εχθρική επίθεση (ανίχνευση εισβολέα) υπερφόρτωση και άλλα. [5]
- Οι υπηρεσίες αποθήκευσης αντιγράφων δεδομένων πραγματοποιούν τη διαχείριση των πόρων αποθήκευσης σε ένα εικονικό δίκτυο, με σκοπό να μεγιστοποιηθεί η απόδοση της πρόσβασης σε δεδομένα, με βάση το κόστος, το χρόνο και την αξιοπιστία. [5]

Οι συλλογικές υπηρεσίες είναι δυνατό να υλοποιηθούν ως «αλυσίδες» υπηρεσιών, με τα σχετικά πρωτόκολλα, ή ως SDK's με τα αντίστοιχα API's, ώστε το σύνολό τους να μπορεί να αξιοποιηθεί από μία εφαρμογή. Η υλοποίησή τους μπορεί να βασιστεί πάνω στο στρώμα πόρων ή σε άλλο συλλογικό στρώμα για τη δημιουργία νέων πρωτοκόλλων και API's. Για παράδειγμα, το συλλογικό στρώμα μπορεί να έχει δύο επίπεδα. Στο πρώτο επίπεδο να ορίζεται το API και SDK της συν-κατανομής, που χρησιμοποιεί ένα διαχειριστικό πρωτόκολλο του στρώματος πόρων για να χειριστεί τον κάθε πόρο που υπάρχει σε χαμηλότερο επίπεδο. Στο δεύτερο επίπεδο μπορεί να καθορίζεται ένα πρωτόκολλο υπηρεσιών ανάθεσης εργασιών και να υλοποιείται μία υπηρεσία υποβολής εργασιών που θα μιλά με αυτό. Η υπηρεσία καλεί με τη σειρά της το API της συν-κατανομής για να εφαρμόσει τις διαδικασίες αυτές και ακόμη παρέχει επιπρόσθετες λειτουργίες, όπως η πιστοποίηση ταυτότητας, η ανοχή στα σφάλματα και η πρόσβαση στο σύστημα. Μία εφαρμογή τέλος μπορεί να χρησιμοποιήσει το πρωτόκολλο υπηρεσίας υποβολής εργασιών για να υποβάλει εργασίες από άκρη σε άκρη του δικτύου. [5]

#### 1.3.2.5 Στρώμα Χρήστη (user layer)

Στην κορυφή κάθε συστήματος πλέγματος είναι οι εφαρμογές του χρήστη, οι οποίες κατασκευάζονται καλώντας, τα κατασκευαστικά στοιχεία των άλλων στρωμάτων. [5]

### 1.3.3 Αρχιτεκτονική ενός Middleware Πλέγματος

Το middleware ενός πλέγματος είναι χτισμένο από πακέτα πολλών επίπεδων που αλληλεπιδρούν μεταξύ τους, χρησιμοποιώντας διαφορετικά στελέχη που καλούνται από ένα API, έτσι ώστε οι χρήστες να μην χρειάζεται να γνωρίζουν τη διαφορετική σύνταξη και τις μεθόδους πρόσβασης των συγκεκριμένων πακέτων. [6]

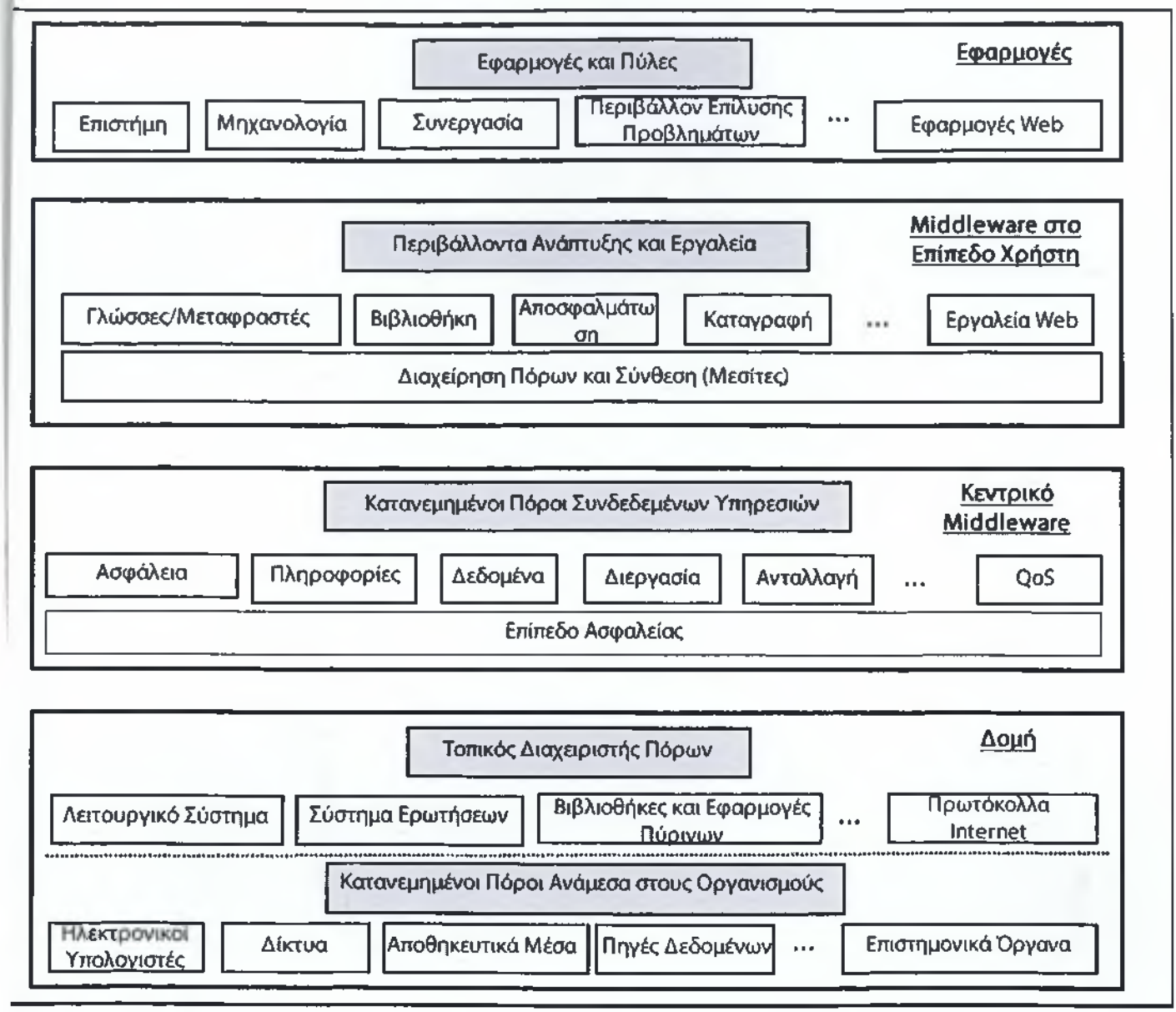
Η παρακάτω εικόνα παρουσιάζει την στοίβα του υλικού και του λογισμικού εντός μιας τυπικής αρχιτεκτονική πλέγματος. Αποτελείται από τέσσερα στρώματα: το δομικό στρώμα, το κεντρικό middleware, το middleware στο επίπεδο χρήστη, καθώς και τις εφαρμογές. [6]

Το Fabric επίπεδο πλέγματος αποτελείται από κατανεμημένους πόρους όπως οι υπολογιστές, τα δίκτυα, συσκευές αποθήκευσης και επιστημονικά όργανα. Η υπολογιστικοί πόροι αντιπροσωπεύουν πολλαπλές αρχιτεκτονικές όπως τα clusters, τους εξυπηρετητές και απλούς προσωπικούς υπολογιστές που τρέχουν διάφορα λειτουργικά συστήματα (όπως UNIX παραλλαγές ή Windows). [6]

Το κεντρικό middleware προσφέρει υπηρεσίες όπως η εξ αποστάσεως διαχείριση των διεργασιών, την από κοινού κατανομή των πόρων, την αποθήκευση, την πρόσβαση, την καταχώρηση και εύρεση των πληροφοριών, την ασφάλεια, Quality of Service (QoS), όπως την κράτηση και την ανταλλαγή των πόρων. Οι υπηρεσίες αυτές αφαιρούν την πολυπλοκότητα και η ετερογένεια του δομικού επιπέδου με την παροχή μιας μεθόδου που επιτρέπει την πρόσβαση στους κατανεμημένους πόρους. [6]

Το middleware στο επίπεδο χρήστη χρησιμοποιεί τις διεπαφές που παρέχονται από το middleware του χαμηλότερου επιπέδου για την παροχή υψηλότερης αφαίρεσης και υπηρεσιών. Αυτές περιλαμβάνουν περιβάλλοντα ανάπτυξης εφαρμογών, εργαλεία προγραμματισμού και μεσίτες πόρων για τη διαχείριση των πόρων και την οργάνωση των εργασιών. [6]

Οι εφαρμογές πλέγματος και οι πύλες συνήθως αναπτύσσονται με τη χρήση μιας γλώσσας πλέγματος. Ένα παράδειγμα μιας εφαρμογής, όπως η προσομοίωση ή ένα πρόβλημα μεγάλης πρόκλησης, θα απαιτήσει αρκετή υπολογιστική ισχύ, πρόσβαση σε απομακρυσμένα δεδομένα, και ενδεχομένως χρειαστεί να αλληλεπιδράσει με τα επιστημονικά όργανα. Οι πύλες πλέγματος προσφέρουν δυνατότητες Web υπηρεσιών, όπου οι χρήστες μπορούν να υποβάλουν και να συλλέξουν τα αποτελέσματα για τις εργασίας τους μέσω του Web. [6]


Εφαρμογές και Πύλες
Εφαρμογές
Επιστήμη
Μηχανολογία
Συνεργασία
Περιβάλλον Επίλυσης Προβλημάτων
...
Εφαρμογές Web
Περιβάλλοντα Ανάπτυξης και Εργαλεία
Middleware στο Επίπεδο Χρήστη
Γλώσσες/Μεταφραστές
Βιβλιοθήκη
Αποσφαλμάτωση
Καταγραφή
...
Εργαλεία Web
Διαχείριση Πόρων και Σύνθεση (Μεσίτες)
Κατανεμημένοι Πόροι Συνδεδεμένων Υπηρεσιών
Κεντρικό Middleware
Ασφάλεια
Πληροφορίες
Δεδομένα
Διεργασία
Ανταλλαγή
...
QoS
Επίπεδο Ασφαλείας
Τοπικός Διαχειριστής Πόρων
Δομή
Λειτουργικό Σύστημα
Σύστημα Ερωτήσεων
Βιβλιοθήκες και Εφαρμογές Πύρινων
...
Πρωτόκολλα Internet
Κατανεμημένοι Πόροι Ανάμεσα στους Οργανισμούς
Ηλεκτρονικοί Υπολογιστές
Δίκτυα
Αποθηκευτικά Μέσα
Πηγές Δεδομένων
...
Επιστημονικά Όργανα

**Εικόνα 6 [6]**

### 1.3.4 Κύριες Χρήσεις των Πλεγμάτων

Αντίθετα με ότι πιστεύεται, το πλέγμα δεν είναι μόνο ένα παράδειγμα υπολογισμού για την παροχή υπολογιστικών πόρων σε απαιτητικές εφαρμογές. Αντίθετα, είναι μια υποδομή που ενοποιεί τους διάφορους απομακρυσμένους πόρους προκειμένου να παράσχει υποστήριξη για ένα ευρύ φάσμα εφαρμογών. Οι διάφοροι τύποι υπολογιστικής υποστήριξης που προσφέρει το πλέγμα μπορούν να κατηγοριοποιηθούν σύμφωνα με τις κύριες προκλήσεις που παρουσιάζονται από την αρχιτεκτονική του πλέγματος. Η κατηγοριοποίηση αυτή είναι η εξής: [7]

-Η κατανεμημένη υποστήριξη υπέρ-υπολογιστών επιτρέπει στις εφαρμογές να χρησιμοποιούν τα πλέγματα ως ζευγάρι υπολογιστικών πόρων, ώστε να μειωθεί ο χρόνος ολοκλήρωσης μιας εργασίας ή για την αντιμετώπιση των προβλημάτων που δεν μπορούν να λυθούν σε ένα ενιαίο σύστημα. Τα κύρια προβλήματα που τίθενται από τις εφαρμογές που απαιτούν τέτοια υποστήριξη είναι η ανάγκη να οργανώσουν τη χρήση των σπάνιων και πολύ δαπανηρών πόρων, την επεκτασιμότητα των πρωτοκόλλων και των αλγορίθμων σε ένα μεγάλο αριθμό κόμβων, την ανοχή στην καθυστέρηση των αλγορίθμων καθώς και την επίτευξη υψηλών επιπέδων απόδοσης. Τυπικές εφαρμογές που απαιτούν την κατανομή τους σε υπέρ-υπολογιστές είναι πρόγνωσης του καιρού και σενάρια στρατιωτικών προσομοιώσεων. [7]

-*Η υψηλής απόδοσης υπολογιστική υποστήριξη επιτρέπει* στις εφαρμογές να χρησιμοποιούν πλέγματα για να θέσουν τους αχρησιμοποίητους κύκλους του επεξεργαστή σε λειτουργία, σε εργασίες που έχουν μικρή εξάρτηση μεταξύ τους ή και καθόλου. Εφαρμογές όπως η προσομοίωση Monte Carlo είναι κατάλληλες για υπολογιστές υψηλών επιδόσεων. [7]

-*Η On-demand υπολογιστική υποστήριξη* επιτρέπει στις εφαρμογές να χρησιμοποιούν τα πλέγματα για να ανακτήσουν εκ νέου πηγές που δεν μπορούν να είναι οικονομικά αποδοτικές ή προσιτές σε τοπικό επίπεδο. Πολλά ζητήματα θέτονται σε πρόκληση, προκειμένου να προσφέρει on-demand υποστήριξη για την τοποθεσία των πόρων, το χρονοπρογραμματισμό, τη διαχείριση κώδικα, τη ρύθμιση των παραμέτρων, την ανοχή στα σφάλματα και την ασφάλεια. Μια οικονομική εφαρμογή που επιτρέπει στους χρήστες να εκτελούν με ακρίβεια αναλύσεις της χρηματιστηριακής αγοράς και την πρόβλεψη των τιμών που τους ενδιαφέρουν από το σπίτι τους είναι ένα αντιπροσωπευτικό παράδειγμα του on-demand computing. [7]

-*Η συνεργατική(Collaborative) υπολογιστική υποστήριξη* επιτρέπει στις εφαρμογές να χρησιμοποιούν το πλέγμα, και να ενισχύουν τις ανθρώπινες αλληλεπιδράσεις, με ένα σύγχρονο ή ασύγχρονο τρόπο, μέσω ενός εικονικού χώρου. Η real-time τις απαιτήσεις που επιβάλλουν οι ανθρώπινες αντιληπτικές ικανότητες, καθώς και το ευρύ φάσμα πολλών διαφορετικών αλληλεπιδράσεων είναι ένα από τα πλέον δύσκολα θέματα της συνεργατικής πληροφορικής. Τυπικό παράδειγμα εφαρμογής που μπορεί να χρησιμοποιήσει τη συνεργατική υπολογιστική υποδομή που παρέχεται από τα πλέγματα είναι οι εφαρμογές multiconferencing. [7]

-*Η Multimedia υπολογιστική υποστήριξη* επιτρέπει στις εφαρμογές να χρησιμοποιούν τα πλέγματα για να μεταφέρουν δεδομένα διασφαλίζοντας άκρο προς άκρο QoS. Κύριες προκλήσεις για την υποστήριξη multimedia υπολογισμών απορρέουν από την ανάγκη

παροχής QoS σε πολλές διαφορετικές μηχανές. Οι εφαρμογές τηλεδιάσκεψης είναι ένα τυπικό παράδειγμα εφαρμογής που απαιτούν multimedia υπολογιστική υποστήριξη. [7]

# 1.4 Public Cloud

## 1.4.1 Εισαγωγή

Η έννοια του cloud computing αποτελεί μια νέα προσέγγιση στο χώρο των κατανεμημένων συστημάτων η οποία χρησιμοποιεί και κάποιες τεχνολογίες που προϋπήρχαν. [8] Σκοπός του είναι η παροχή υπηρεσιών όπως, η υπολογιστική ισχύ και η αποθηκευτική δυνατότητα, στους χρήστες του συστήματος. Το cloud computing θεωρείται κάθε σύστημα το οποίο επιτρέπει την ανάθεση υπολογιστικών και αποθηκευτικών υπηρεσιών εξωτερικά. Οι καταναλωτές έχουν τη δυνατότητα πρόσβασης σε εφαρμογές και δεδομένα από όπου και αν βρίσκονται. Ακόμα είναι σίγουροι ότι η υποδομή του cloud είναι πολύ ισχυρή και θα είναι πάντα διαθέσιμη ανά πάσα στιγμή. Οι υπηρεσίες είναι εξαιρετικά αξιόπιστες, επεκτάσιμες και αυτόνομες. [9]

## 1.4.2 Εξάπλωση του Cloud Computing

Η εξάπλωση του cloud έναντι του πλέγματος τα τελευταία χρόνια είναι ραγδαία. Αυτό οφείλετε κυρίως σε οικονομικούς και επιχειρηματικούς παράγοντες, βάση των επενδύσεων που έχουν κάνει οι επιχειρήσεις πάνω στις τεχνολογίες πληροφορικής.

Οι παγκόσμιοι φορείς πληροφορικής ξόδεψαν $3.4 τρισεκατομμύρια το 2008, με το τελικό αποτέλεσμα να είναι πτωτικό την χρονική περίοδο 2001 - 2010. Στην πραγματικότητα λόγο της ύφεσης και της πιστωτικής κρίσης, επήλθε η μείωση του αρχικού κεφαλαίου και της πίστωσης που διέθεταν οι επιχειρήσεις για τα μεγάλα έργα πληροφορικής. Αυτό είχε σαν αποτέλεσμα πολλοί νεοεισερχόμενοι στην αγορά να χρησιμοποιούν το cloud με τις τιμές υπηρεσιών να μειώνονται λόγο του αυξανόμενου ανταγωνισμού μεταξύ των μεγάλων εταιριών. [10]

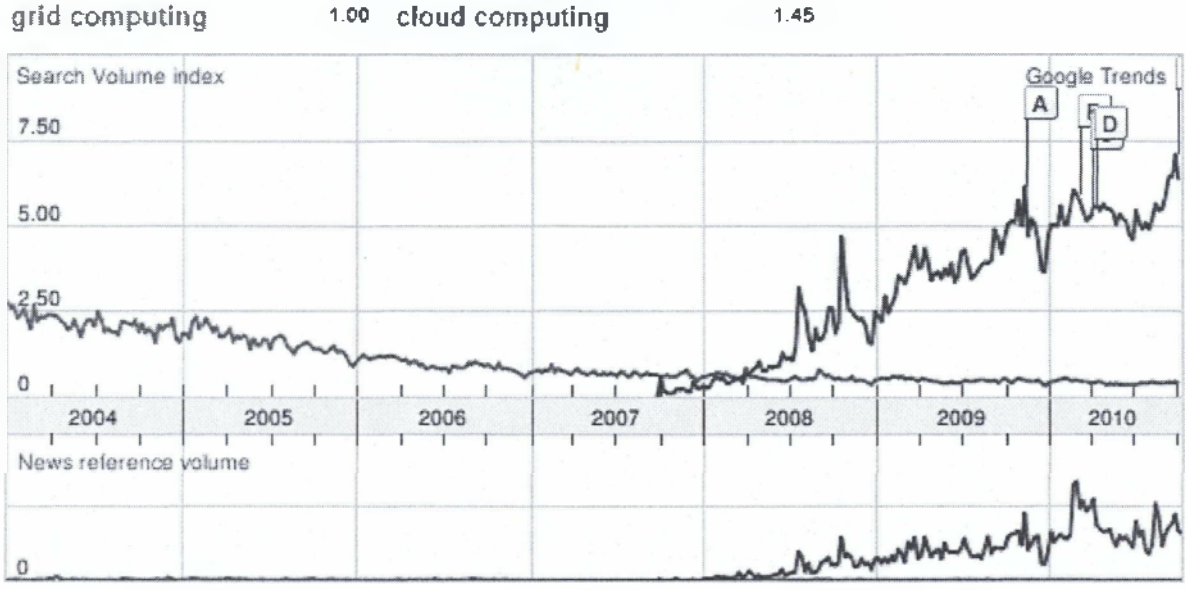
"Η IDC εκτιμά ότι περίπου το δέκα τοις εκατό των περίπου 64 δισεκατομμυρίων δολαρίων που ξοδευτήκαν για επιχειρηματικές εφαρμογές σε όλο τον κόσμο κατά το 2008 δαπανήθηκε σε εφαρμογές του cloud computing. Πολλοί αναλυτές βλέπουν ότι ο ρυθμός ανάπτυξης του cloud computing θα υπερβεί το 20% με 30% ή και περισσότερο μέσα στα επόμενα χρόνια, εκτιμώντας ότι η παγκόσμια αγορά στις υπηρεσίες του cloud computing θα μπορούσε να φθάσει 42 δισεκατομμύρια δολάρια μέχρι το 2012. Η Gartner προβλέπει ότι η αγορά του cloud computing ξεπερνά ήδη τα 40 δισεκατομμύρια δολάρια σήμερα, και ότι θα αυξηθεί σε πάνω από 150 δισεκατομμύρια δολάρια έως το 2013" [10]

"Σύμφωνα με τα αποτελέσματα της έρευνας του 2009 πάνω στο cloud computing, μας δείχνει ότι πάνω από 500 εταιρίες πληροφορικής σκέπτονται τη στροφή τους στο cloud computing και μάς δείχνει ότι όλο και περισσότεροι οργανισμοί επιλέγουν νέες τεχνολογίες για να μειώσουν το κόστος, αντί να μειώσουν τη διάδοση της τεχνολογίας. Επίσης το cloud computing δεν σε αναγκάζει να χρησιμοποιήσεις όλες τις δυνατότητες του. Πράγματι, φαίνεται στην πράξη ότι η διείσδυση του cloud ξεκινά συχνά, όταν οι οργανισμοί

χρησιμοποιούν αρχικά μερικούς πόρους του cloud για ένα μέρος των μη κρίσιμων εφαρμογών τους δοκιμαστικά." [10]

### 1.4.2.1 Η αποδοχή του cloud συμφώνα με το διαδίκτυο

Συμφώνα με την Web εφαρμογή trends της Google τα τελευταία χρόνια η δημοτικότητα του cloud computing έχει ραγδαία αύξηση σε σχέση με το πλέγμα που όπως φαίνεται παραμερίζεται όλο και πιο πολύ. [11]




**Εικόνα 7 [11]**

## 1.4.3 Αρχιτεκτονική του Cloud

Το cloud computing παρέχει υπηρεσίες που μπορούν να ομαδοποιηθούν σε τρεις κατηγορίες:

Λογισμικό ως υπηρεσία, πλατφόρμα ως υπηρεσία και υποδομή ως υπηρεσία. Οι τρεις αυτές υπηρεσίες μαζί δημιουργούν μια δομή από στρώματα όπως φαίνεται στην **Εικόνα 8**. [4]

### 1.4.3.1 Λογισμικό ως υπηρεσία

Το λογισμικό ως υπηρεσία (SaaS) είναι ένα ολοκληρωμένο λογισμικό που προσφέρεται ως υπηρεσία όποτε αυτή ζητηθεί. Ένα στιγμιότυπο του λογισμικού τρέχει στο cloud και εξυπηρετεί πολλαπλούς χρηστές ή οργανισμούς. Το ποιο γνωστό παράδειγμα είναι το Google Apps που προσφέρει βασικές υπηρεσίες σε μια επιχείρηση όπως το email και η επεξεργασία κειμένου. [4]

### 1.4.3.2 Πλατφόρμα ως υπηρεσία

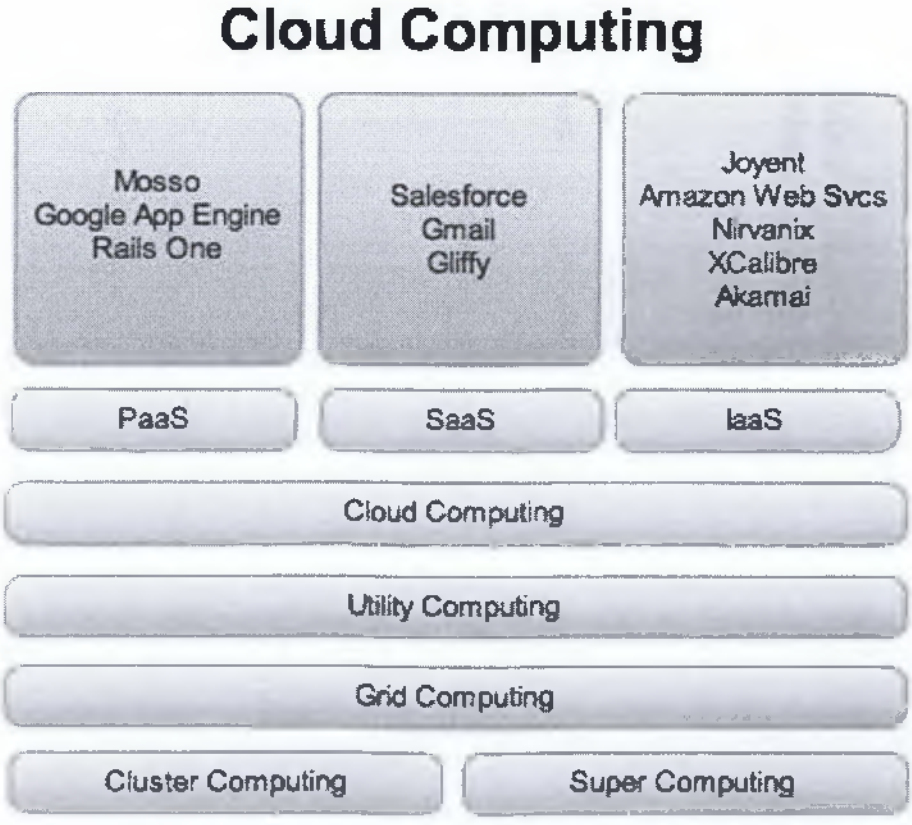
Η πλατφόρμα ως υπηρεσία (PaaS) είναι ένα επίπεδο εφαρμογής που παρέχεται ως υπηρεσία ώστε να κατασκευαστούν υψηλότερου επιπέδου υπηρεσίες. Την πλατφόρμα ως υπηρεσία μπορούμε να τη δούμε σαν δυο κατηγορίες ανάλογα με το δημιουργό ή το χρηστή της υπηρεσίας.:

- Κάποιος που δημιουργεί μια πλατφόρμα ως υπηρεσία ενσωματώνοντας ένα λειτουργικό σύστημα, ένα middleware, μια εφαρμογή ή και ένα περιβάλλον ανάπτυξης που το παρέχει στο χρήστη ως υπηρεσία.
- Κάποιος που χρησιμοποίει την πλατφόρμα ως υπηρεσία θα μπορεί να έχει πρόσβαση σε αυτή μέσω κάποιου API. Ο πελάτης αλληλεπιδρά με την πλατφόρμα μέσω του API και η πλατφόρμα κάνει όλες τις απαραίτητες λειτουργιές διαχειρίσεις και επέκτασης παρέχοντας ένα επίπεδο υπηρεσιών. Οι εικονικές συσκευές μπορούν να χαρακτηριστούν σαν πλατφόρμα ως υπηρεσία.

Η πλατφόρμα ως υπηρεσία παρέχει όλα τα επίπεδα που χρειάζονται για να αναπτυχτεί και να δοκιμαστεί μια εφαρμογή. Εμπορικά παραδείγματα είναι το Google Apps Engine, που διανέμει εφαρμογές μέσα από τις εγκαταστάσεις της Google. [4]

### 1.4.3.3 Υποδομή ως υπηρεσία

Η Υποδομή ως υπηρεσία (IaaS) παρέχει υπολογιστικές και αποθηκευτικές δυνατότητες ως βασικές υπηρεσίες μέσω του δικτύου. Εξυπηρετητές, συστήματα αποθήκευσης, switches, routers, και άλλα συστήματα συγκεντρώνονται ώστε να μπορέσουν να χειριστούν το φόρτο εργασίας μιας μικρής ή και μεγάλης σε απαιτήσεις εφαρμογής. [4]




Εικόνα 8 [12]

### 1.4.4 Πλεονεκτήματα του Cloud

Για να επωφεληθούμε στο μέγιστο βαθμό από τις υπηρεσίες που προσφέρει το cloud οι προγραμματιστές πρέπει να τροποποιήσουν κατάλληλα τις εφαρμογές τους ώστε να γίνει καλύτερη εκμετάλλευση της αρχιτεκτονικής και της υποστήριξης που προσφέρει το cloud. Τα οφέλη που έχουμε είναι η μείωση του χρόνου εκτέλεσης, χρόνου απόκρισης, του ρίσκου που έχουμε όταν τοποθετούμε τις εφαρμογές στη φυσική υποδομή και τη μείωση κόστους ανάπτυξης. [4]

#### 1.4.4.1 Μείωση Χρόνου εκτέλεσης και απόκρισης

Οι εφαρμογές που χρησιμοποιούν το cloud για να εκτελέσουν εργασίες μπορούν να χρησιμοποιήσουν χιλίους υπολογιστές ώστε να τελειώσει η εργασία στο 1/1000 του χρόνου που θα χρειάζονταν ένας και μόνο υπολογιστής. Οι εφαρμογές που χρειάζεται να έχουν γρήγορο χρόνο απόκρισης στους πελάτες, αναδημιουργούνται ώστε οι εργασίες που έχουν ανάγκη για περισσότερη επεξεργαστική δύναμη να αποστέλλονται στους "εργάτες". Οι "εργάτες" είναι πολλοί εικονικοί υπολογιστές που βελτιστοποιούν το χρόνο απόκριση ανάλογα με τις απατήσεις του πελάτη. [4]

#### 1.4.4.2 Χαμηλότερο κόστος εισαγωγής

Υπάρχουν πολλοί λόγοι που το cloud μας βοηθάει να μειώσουμε το κόστος εισαγωγής σε μια νέα αγορά:

- "Επειδή η υποδομή ενοικιάζεται, και δεν αγοράζεται, το κόστος είναι ελεγχόμενο και η αρχική μας επένδυση μπορεί να είναι μηδενική. Η μείωση του κόστους αγοράς υπολογιστικών κύκλων και αποθηκευτικού χώρου γίνεται δυνατή από το μεγάλο αριθμό παροχών cloud που βοηθούν περαιτέρω στη μείωση του κόστους εισαγωγής." [4]
- "Οι εφαρμογές αναπτύσσονται συναρμολογώντας τα μέρη της εφαρμογής χωρίς να χρειάζεται να κάνουμε χρήση προγραμματισμού. Αυτό επιτρέπει τη ραγδαία ανάπτυξη των εφαρμογών, βοηθώντας να μειωθεί ο χρόνος κυκλοφορίας στην αγορά, κάτι που εκμεταλλεύονται οι επιχειρήσεις ώστε να υπερισχύσουν έναντι του ανταγωνισμού." [4]

#### 1.4.4.3 Αυξανόμενος ρυθμός της καινοτομίας

Το cloud computing βοηθάει ώστε να αυξηθεί ο ρυθμός καινοτομίας. Το χαμηλό κόστος που προσφέρει του επιτρέπει την είσοδο σε νέες αγορές και έτσι βοηθά ώστε να εξασφαλιστούν ισότιμοι όροι ανταγωνισμού, επιτρέποντας στις νεοϊδρυόμενες επιχειρήσεις να αναπτύξουν νέα προϊόντα πιο γρήγορα και με χαμηλότερο κόστος. Αυτό δίνει τη δυνατότητα στις μικρές επιχειρήσεις να ανταγωνιστούν αποτελεσματικότερα τους παραδοσιακούς οργανισμούς των οποίων η διαδικασία ανάπτυξης εφαρμογών στα επιχειρησιακά datacenters είναι σημαντικά μεγαλύτερη. Η αύξηση του ανταγωνισμού βοηθά και στο να αυξηθεί ο ρυθμός καινοτομίας με το καλύτερο παράδειγμα να είναι η χρήση λογισμικού ανοικτού κώδικα, ώστε το σύνολο της βιομηχανίας να επωφελείται από αυτό. [4]

## 1.4.5 Περίπτωση χρήσης του Cloud

Μια προσπάθεια για την ανάπτυξη εφαρμογών σε cloud έχει γίνει από την Yahoo! η οποία έθεσε σαν στόχο την προσφορά αποθηκευτικών και υπολογιστικών υπηρεσιών, μέσου του διαδικτύου, στους χρήστες. Προκειμένου να γίνεται πιο εύκολη η ανάπτυξη και η συντήρηση των εφαρμογών χρησιμοποιώντας το μοντέλο του cloud, οι νέες εφαρμογές που προσθέτονται στο σύστημα τοποθετούνται στην κορυφή του cloud. Με αυτό τον τρόπο επιτυγχάνεται η πιο εύκολη παραγωγή εφαρμογών. Επιπλέον χρησιμοποιώντας μια οριζόντια αρχιτεκτονική υπηρεσιών δεν απαιτείται πλήρη γνώση από τους προγραμματιστές σε θέματα όπως η παροχή των υπηρεσιών, ασφάλειας και εισαγωγής νέων εξυπηρετητών για την κάλυψη του φορτίου. Έτσι λοιπόν η Yahoo έχει θέσει ως στόχο την παροχή μιας ισχυρή υποδομή για της υπηρεσίες που προσφέρει προκειμένου να μπορέσει να υποστηρίξει τις εφαρμογών που αναπτύσσονται μέσα σε αυτή. [13]

### 1.4.5.1 Απαιτήσεις.

Προκειμένου να επιτευχθούν οι στόχοι που έχουν τεθεί και αξιοποιώντας την προηγούμενη εμπειρία της η Yahoo! θέτει μια σειρά από απαιτήσεις.

**Multitenancy**: οι υπηρεσίες που παρέχει το cloud πρέπει να υποστηρίζουν διαφορετικές εφαρμογές. Οι εφαρμογές μπορούν να διαμοιράζονται πληροφορίες αλλά εκτελούνται απομονωμένα.

**Elasticity**: η εφαρμογές πρέπει να είναι σε θέση να διαπραγματεύονται και να λαμβάνουν επιπλέον πόρους προκειμένου να καλύπτουν τις διαρκώς αυξανόμενες ανάγκες τους σε υπολογιστική ισχύ και αποθηκευτική δυνατότητα

**Scalability**: το σύστημα πρέπει να ανακατανέμει τα δεδομένα σε περίπτωσης εισαγωγής νέου υλικού.

**Load and Tenant Balancing:** το σύστημα πρέπει να έχει την δυνατότητα μεταφοράς φορτίου ανάμεσα στους εξυπηρετητές για να αποφευχθεί υπερφόρτωση. Δηλαδή σε περίπτωση που για κάποιον λόγο προκύψει πολλαπλασιασμός του φορτίου σε κάποια εφαρμογή θα πρέπει με κάποιον τρόπο να υποστηριχθεί αυτό το νέο φορτίο.

**Availability:** το σύστημα πρέπει να συνεχίζει τη λειτουργία του ακόμα και σε περίπτωση υψηλών ποσοστών αποτυχίας. Δηλαδή σε περίπτωση αποτυχίας εξυπερετητών ή των δικτύων οι υπηρεσίες του cloud πρέπει να παραμένουν ανεπηρέαστες και να παρέχονται χωρίς κανένα πρόβλημα.

**Security**: κρίσιμο σημείο για την παραβίαση της ασφάλειας του συστήματος θα προκαλέσει πρόβλημα και στις εφαρμογές.

**Operability**: η λειτουργικότητα συστημάτων και διασύνδεση των συστημάτων που εκτελούνται στο cloud πρέπει να είναι όσο το δυνατό πιο εύκολη στην διαχείριση τους.

**Metering**: η παρακολούθηση των πόρων από τις εφαρμογές ώστε να γίνεται η λήψη των αποφάσεων και υπολογισμό του κόστους.

**Global**: τοποθέτηση των υπηρεσιών «κοντά» στον χρήστη για την μείωση καθυστερήσεων.

**Simple APIs**: ώστε να γίνεται πιο εύκολη η ανάπτυξης των εφαρμογών που χρησιμοποιούν το cloud. [13]

### 1.4.6 Σύγκριση Grid και Cloud Computing

Υπάρχει μια σύγχυση που επικρατεί γύρω από το Cloud σε σχέση με το Grid Computing. Η διαφορά τους δεν είναι εύκολα διακριτή γιατί τα Clouds και τα Grids μοιράζονται κοινά χαρακτηριστικά:

Τη μείωση του κόστους και αύξηση της ευελιξίας και της αξιοπιστίας με τη χρήση υλικού που προέρχεται από διαφορετικούς κατασκευαστές. Θα συγκρίνουμε το Grid με τους βασικούς ορισμούς του cloud. [8]

| | |
|---|---|
| Resource Heterogeneity | Υποστηρίζουν την συνάθροιση ετερογενών πόρων |
| User Access | Πρόσβαση χρηστών στους πόρους με διαφανή τρόπο |

**Πίνακας 1: κοινά χαρακτηριστικά cloud και grid [8]**

| | Grid | Cloud |
|---|---|---|
| Virtualization | Καλύπτει την ετερογένεια των πόρων μέσω Virtualization δεδομένων και πόρων. | Επεκτείνει τη διαδικασία και στο υλικό. |
| Security | Αυθεντικοποίηση μέσω πιστοποιητικών. | Κάθε χρήστης έχει μοναδική πρόσβαση σε κάθε εικονικό περιβάλλον (απομόνωση). |
| Scalability | Μέσω της αύξηση του αριθμού των κόμβων. | Αναπροσαρμογή των πόρων με αυτοματοποιημένο τρόπο. |
| SelfManagement | Πιο εύκολη όταν υπάρχει ενιαία διαχείριση του συστήματος. | Αυτοματοποιημένη διαδικασία, προβλέπονται δυσκολίες σε περίπτωση απουσίας ενιαία διαχείριση. |
| QoS Guarantees | Περιορισμένες εγγυήσεις συχνά μόνο αυτή της υπηρεσίας βέλτιστης προσπάθειας. Οι εφαρμογές αναγκάζονται να καλύψουν αυτόν τον τομέα. | Περιορισμένες, οι εφαρμογές κληρονομούν από την πλατφόρμα τις εγγυήσεις. |

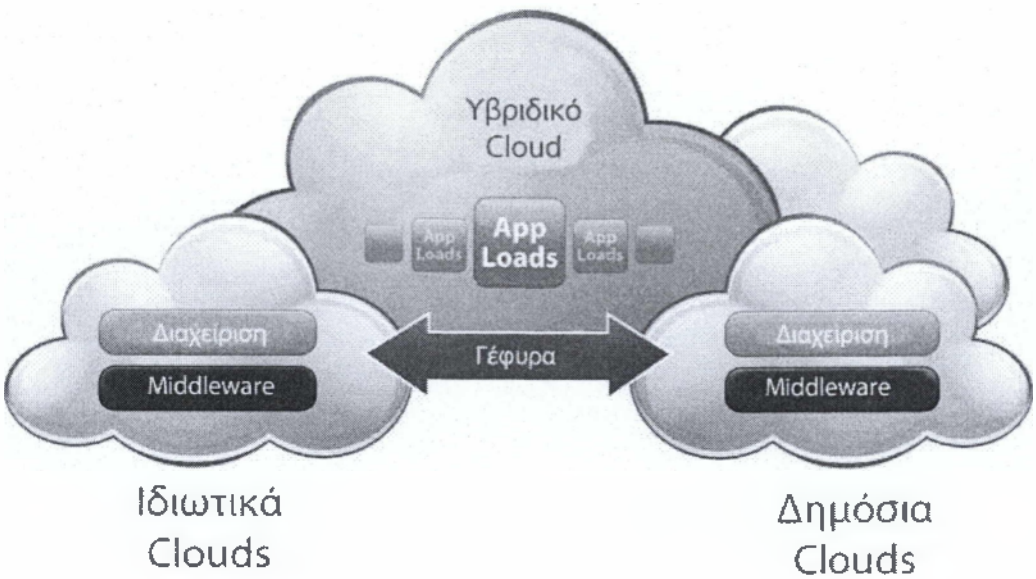
**Πίνακας 2: κοινά χαρακτηριστικά διαφορετική προσέγγιση [8]**

| | Grid | Cloud |
|---|---|---|
| Resource Sharing | Υποστηρίζει κοινή χρήση των πόρων. | Δεν υποστηρίζεται λόγο της απομόνωσης μέσου της Virtualization. |
| High Level Services | Πληθώρα υπηρεσιών όπως υπηρεσίες μεταφοράς δεδομένων και εύρεση μέσο μεταδεδομένων. | Έλλειψη που πιθανόν οφείλεται στο χαμηλό επίπεδο ωριμότητας. |
| Architecture | Προσανατολισμένη στις υπηρεσίες. | Αρχιτεκτονικές που επιλέγονται από το χρήστη. |
| Software Dependencies | Εξάρτηση από την περιοχή εφαρμογής. | Ανεξαρτησία από την περιοχή εφαρμογής. |
| Platform Awareness | Απαραίτητη η γνώση του λογισμικού-πελάτη σχετικά με grid λειτουργίες. | Ο SP δεν προαπαιτεί γνώσει. |
| Software Workflow | Οι εφαρμογές απαιτούν μια προκαθορισμένη ροή εργασιών που να συντονίζει τις υπηρεσίες. | Η ροη εργασιών δεν παίζει σημαντικό ρόλο για τις εφαρμογές. |
| Usability | Μικρότερος βαθμός ευχρηστίας. | Μεγαλύτερη ευχρηστία μέσω της απόκρυψης λεπτομερειών. |
| Standardization | Υπάρχουν πρότυπα μέσο των οποίο πετυχαίνεται η διαλειτουργικότητα. | Ανάγκη για πρότυπα στην αποθήκευση, ποιότητα υπηρεσιών, καθορισμού των διεπαφών. |
| Payment Model | Ανελαστικό, τιμολόγηση βάση ενός σταθερού ποσού ανά υπηρεσία | Ευέλικτο, τιμολόγηση βάση της χρήση της υπηρεσίας. |

**Πίνακας 3: Διαφορές [8]**

## 1.5 Υβριδικό Cloud

Τα δημόσια clouds έγιναν περισσότερο δημοφιλή τα τελευταία χρόνια. Ένα από τα πρώτα δημόσια cloud ήταν το EC2 της Amazon που πρόσφερε πολλά σε αυτήν την ανάπτυξη. Όμως δεν μεγαλώνει γρήγορα μονό η κοινότητα του cloud αλλά και οι κριτικές που δέχεται σε θέματα όπως η ασφάλεια των δεδομένων και το προσωπικό απόρρητο. Μια προφανής αλλά όχι και τόσο αξιόλογη λύση σε αυτά τα προβλήματα είναι η χρήση του ιδιωτικού και του δημόσιου μοντέλου μαζί ώστε να δημιουργήσουν ένα νέο μοντέλο, το υβριδικό. [14] Το υβριδικό cloud μπορούμε να το διακρίνουμε όταν ένας οργανισμός υιοθέτει τις πρακτικές που προσφέρει του ιδιωτικού και του δημόσιου cloud. Αυτό γίνεται έτσι ώστε οι εφαρμογές που αναπτύσσονται να ωφελούνται από τα πλεονεκτήματα που διαθέτουν αυτά τα δυο μοντέλα cloud. [15]




**Εικόνα 9 [16]**

### 1.5.1 Ευκαιρίες

Στο υβριδικό cloud μια εταιρία μπορεί να έχει το δικό της ιδιωτικό cloud και να το επεκτείνει με τη βοήθεια του δημόσιου cloud ανάλογα με της ανάγκες της. Μετακινώντας τα δεδομένα από το κεντρικό σύστημα δεδομένων στο υβριδικό cloud έχει πολλά πλεονεκτήματα: [14]

- Βέλτιστη χρήση: Σε ένα τυπικό κέντρο δεδομένων γίνεται χρήση μονό στο 5% με 20% των διαθέσιμων πόρων. Επειδή ο μέγιστος φόρτος εργασίας είναι έως και δέκα φόρες υψηλότερος από το μέσο φόρτο, οι εξυπηρετητές είναι τον περισσότερο χρόνο αδρανής αυξάνοντας το κόστος άσκοπα. Τα υβριδικά clouds μπορούν να διαχειριστούν καλύτερα το φόρτο εργασίας κάνοντας χρήση εξωτερικών πόρων όποτε αυτό είναι αναγκαίο.
- Μεταφορά κίνδυνου: Όσες εταιρίες είναι υπεύθυνες για την καλή λειτουργία των κέντρων συστημάτων που διαθέτουν, έχουν την δυνατότητα να μεταβιβάσουν τον κίνδυνο στις εταιρίες που παρέχουν τα δημόσια clouds έτσι ώστε να υπάρχει υψηλή ασφάλεια στις υπηρεσίες τους.
- Διαθεσιμότητα: Το να εξασφαλίσουμε τη διαθεσιμότητα σε ένα εταιρικό περιβάλλον είναι αρκετά δύσκολο και δαπανηρό, γιατί χρειάζεται να διαθέσουμε ένα μεγάλο αριθμό εργαζομένων όπως επίσης και τη δημιουργία αντίγραφων ασφαλείας. Στο υβριδικό περιβάλλον το δημόσιο cloud επεκτείνεται ή παίρνει τον έλεγχο όλων των εργασιών όταν συμβεί κάποιο σφάλμα ή κάποια επίθεση άρνησης εξυπηρέτησης.

### 1.5.2 Προκλήσεις

Ακόμα κι αν τα υβριδικά cloud είναι μια συμφέρουσα πρόταση ως προς το κόστος και τις δυνατότητες που προσφέρουν ο αριθμός των προκλήσεων και των προβλημάτων που μπορεί να προκύψουν είναι μεγάλος. Ειδικά λόγω της συνεχόμενης εξέλιξης που το χαρακτηρίζει, το cloud computing έχει πολλά άλυτα οικονομικά και τεχνικά ζητήματα. [14]

#### 1.5.2.1 Κόστος

Ένα από τα πιο προφανή εμπόδια, είναι το γεγονός ότι τα υβριδικά cloud απαιτούν την ύπαρξη υποδομών τοπικά αλλά και επιπλέον απομακρυσμένους πόρους από έναν πάροχο. Όμως στις υποδομές του υβριδικού cloud θα πρέπει να υπολογίζονται το κόστος εγκατάστασης και το κόστος λειτουργίας ενός κέντρου δεδομένων (π.χ. hardware, ισχύ, ψύξη, συντήρηση) καθώς και η χρήση του με βάση το κόστος που έχει ο πάροχος. Ανάλογα με τη χρήση, και την κίνηση δεδομένων στο cloud, οι επιχειρήσεις πρέπει να αποφασίσουν αν η χρήση του είναι κερδοφόρα. [14]

Έτσι προτείνεται ένα μοντέλο για να έχουμε την καλύτερη απόδοση όσον αφορά τα κέρδη βάση των χαρακτηριστικών που επηρεάζουν τις αποφάσεις που πρέπει να ληφθούν για την δημιουργία του υβριδικού μοντέλου: [14]

- Πληρωμή ξεχωριστά ανά πόρο: οι περισσότερες εφαρμογές δεν χρησιμοποιούν τους διαθέσιμους πόρους ισόμερος, αλλά χρησιμοποιούν μερικούς από αυτούς εκτενώς. Κάποιες εφαρμογές χρησιμοποιούν περισσότερο τους επεξεργαστές και κάποιες περισσότερο τον αποθηκευτικό χώρο ή το εύρος ζώνης. Εξαρτάται από τον πάροχο σε ποιο τομέα προσφέρει καλύτερη υποστήριξη ώστε το τοπικό μας σύστημα να προσφέρει σε κάποιο άλλο τομέα καλύτερα.

- Ανάλογα με το πόσο ακριβό είναι το ιδιωτικό data center, οι τοπικές εφαρμογές θα πρέπει να έχουν την ανάλογη αποδοτικότητα και σε κόστος ρεύματος και ψύξης.

- Κόστος λειτουργίας: Το περιβάλλον του cloud έχει χαμηλότερο υλικό κόστος, επειδή τα κέντρα δεδομένων είναι εικονικά και ο κινδύνους αποτυχίας είναι θέμα των παρόχων. Οι δαπάνες λειτουργίας και διαχείρισης λογισμικού, ωστόσο παραμένουν η ίδιες.

- Αξιοποίηση: τα κέρδη και το κόστος είναι ανάλογα του κέντρου δεδομένων. Ενώ οι εξωτερικοί πάροχοι αναλαμβάνουν τις δαπάνες λειτουργίας και χρήσης.

Ανάλογα με τη διαθεσιμότητα των πόρων του τοπικού κέντρου δεδομένων και το κόστος χρήσης του cloud, οι επιχειρήσεις πρέπει να αποφασίσουν πόσους πόρους θα δεσμεύσουν ακόμα από το cloud.

# Κεφάλαιο 2

# Τεχνολογίες ανάπτυξης εφαρμογών Cloud

# 2.1 MPI

## 2.1.1 Πρότυπο παράλληλου προγραμματισμού με πέρασμα μηνυμάτων

Στο πρότυπο παράλληλου προγραμματισμού με πέρασμα μηνυμάτων (message passing model), ο παράλληλος υπολογισμός αποτελείται από έναν αριθμό διεργασιών κάθε μία από τις οποίες επεξεργάζεται κάποια «τοπικά» δεδομένα. Κάθε διεργασία έχει τις δικές της τοπικές μεταβλητές, και δεν υπάρχει η δυνατότητα κάποια διεργασία να έχει άμεση πρόσβαση στην τοπική μνήμη μιας άλλης διεργασίας. Η επικοινωνία μεταξύ των διεργασιών και η ανταλλαγή δεδομένων, προκειμένου να συνεργαστούν στην επίλυση ενός προβλήματος, επιτυγχάνεται με την αποστολή και την παραλαβή μηνυμάτων.

Θα πρέπει να παρατηρήσουμε ότι το πρότυπο

- αναφέρεται σε διεργασίες οι οποίες δεν τρέχουν αναγκαστικά, σε διαφορετικούς επεξεργαστές ή διαφορετικές διεργασίες μπορούν τρέχουν σε διαφορετικούς επεξεργαστές.
- μπορεί να υλοποιηθεί σε μια ποικιλία διαφορετικών μηχανών όπως για παράδειγμα, σε πολυεπεξεργαστές με κοινή μνήμη, σε δίκτυα σταθμών εργασίας (networks of workstations), και ακόμη σε μηχανές με ένα μόνον επεξεργαστή.
- επιτρέπει περισσότερο έλεγχο πάνω στη θέση και τη ροή των δεδομένων μέσα σε ένα παράλληλο πρόγραμμα από ότι ένα πρότυπο κοινής μνήμης. Έτσι, τα προγράμματα τα οποία χρησιμοποιούν πέρασμα μηνυμάτων, συνήθως, έχουν καλύτερη απόδοση.

## 2.1.2 Τι είναι το MPI

Το MPI (Message Passing Interface) είναι μία βιβλιοθήκη συναρτήσεων (σε C ή σε C++) ή υπορουτινών (σε Fortran) για τη συγγραφή παράλληλων προγραμμάτων με πέρασμα μηνυμάτων σε μια μεγάλη ποικιλία συστημάτων. Περιλαμβάνει ένα μεγάλο αριθμό συναρτήσεων οι οποίες διαχειρίζονται την επικοινωνία μεταξύ των επεξεργαστών. [17]

Ένα πρόγραμμα MPI αποτελείται από πολλές διεργασίες που εκτελούνται παράλληλα. Κάθε διεργασία εκτελεί ένα τμήμα του ίδιου προγράμματος και χρησιμοποιεί τα δικά της δεδομένα για να κάνει τους υπολογισμούς της. Άρα, κάθε διεργασία χρησιμοποιεί το δικό της χώρο διευθύνσεων χωρίς αυτό να σημαίνει ότι δεν υπάρχουν υλοποιήσεις του MPI για επεξεργαστές με διαμοιραζόμενη μνήμη. Ο αριθμός των διεργασιών που δημιουργούνται σε ένα πρόγραμμα MPI, είναι σταθερός και καθορίζεται από τον προγραμματιστή. Κατά τη διάρκεια της εκτέλεσης δεν μπορεί να δημιουργήσει καινούργιες διεργασίες ή να καταστρέψει κάποια από τις υπάρχουσες. Δηλαδή, το πρόγραμμα πρέπει να εκτελεστεί από την αρχή για να οριστεί ξανά ο καινούργιος αριθμός διεργασιών.

Η μεταφορά δεδομένων από τις μεταβλητές μιας διεργασίας στις μεταβλητές μιας άλλης διεργασίας γίνεται με πέρασμα μηνυμάτων από τη μία διεργασία στην άλλη. Την αποστολή των μηνυμάτων την αναλαμβάνει το MPI. Ο προγραμματιστής καλεί τις συναρτήσεις αποστολής και λήψης (ή παραλαβής) μηνυμάτων μέσα από το πρόγραμμά του αλλά, δεν

ασχολείται με τις λεπτομέρειες της επικοινωνίας μεταξύ των διεργασιών. Ωστόσο, ο προγραμματιστής όταν εκτελεί την εφαρμογή, μπορεί να δώσει ως παράμετρο τον αριθμό των διεργασιών που θα δημιουργηθούν για την επίλυση του προβλήματος. [18]

Το πρότυπο MPI-1 ορίσθηκε την Άνοιξη του 1994. Σημειώνεται ότι: [19]

- Το πρότυπο καθορίζει τα ονόματα, τις παραμέτρους κλήσεις και τις εξόδους συναρτήσεων και υπορουτινών προκειμένου να καλούνται από C και Fortran κώδικα, αντίστοιχα.
- Η λεπτομερής υλοποίηση της βιβλιοθήκης αφήνεται στους κατασκευαστές, οι οποίοι είναι έτσι, ελεύθεροι να παράγουν βέλτιστες εκδόσεις για τις μηχανές τους.
- Το πρότυπο έχει γίνει αποδεκτό από πολλούς φορείς και ερευνητές τόσο στη βιομηχανία των υπολογιστών όσο και στα πανεπιστήμια. Υλοποιήσεις του υπάρχουν διαθέσιμες για μια μεγάλη ποικιλία από πλατφόρμες.

Το πρότυπο MPI-2 έχει επίσης, ορισθεί το 1996. Είναι σημαντικό να αναφέρουμε ότι το MPI-2 είναι περισσότερο ένα υπερσύνολο του MPI-1, ωστόσο κάποιες λειτουργίες έχουν αποσυρθεί. Τα προγράμματα MPI-1.2 μπορούν να λειτουργήσουν κάτω από MPI υλοποιήσεις του MPI-2 standard. Έτσι το MPI-2: [19]

- Έχει χαρακτηριστικά και δυνατότητες που δεν περιλαμβάνονται στο MPI-1, όπως, εργαλεία για παράλληλο I/O, διασύνδεση με τη C++ και τη Fortran 90, και διαχείριση δυναμικών διεργασιών.
- Έχει διεπαφές που παρέχουν εικονική τοπολογία, συγχρονισμό και λειτουργίες επικοινωνίας ανάμεσα σε ένα σύνολο διεργασιών (οι οποίες έχουν ταξινομηθεί σε κόμβους, servers, υπολογιστές) ανεξαρτήτου γλώσσας, με συγκεκριμένο συντακτικό γλώσσας, συν κάποια χαρακτηριστικά τα οποία είναι ξεχωριστά σε κάθε γλώσσα.

### 2.1.3 Οι σκοποί του MPI

Ο βασικός σκοπός του MPI είναι να δώσει στους προγραμματιστές ένα καλά ορισμένο πρότυπο παρέχοντας ένα σύνολο συναρτήσεων για τη συγγραφή παραλλήλων προγραμμάτων με πέρασμα μηνυμάτων. Τα MPI προγράμματα είναι μεταφέρσιμα σε πολλές αρχιτεκτονικές υπολογιστών και ο εκτελέσιμος κώδικας που παράγεται, είναι αρκετά αποδοτικός. Η μεγάλη ποικιλία συναρτήσεων που παρέχει το MPI, οδηγεί σε ευέλικτα προγράμματα που μπορούν εύκολα να τροποποιηθούν και να συντηρηθούν. Το MPI παρέχει: [18]

- Διαφορετικούς τύπους επικοινωνίας μεταξύ των διεργασιών. Αποτελεσματική επικοινωνία μεταξύ των διεργασιών.
- Ένα ασφαλές πρότυπο επικοινωνίας μεταξύ των διεργασιών (blocking send-receive).
- Ειδικές ρουτίνες για κοινές «συλλογικές» (collective) λειτουργίες.
- Δυνατότητα χειρισμού τοπολογιών και τύπων δεδομένων που ορίζονται από το χρήστη.

- Δυνατότητα ανάπτυξης και εκτέλεσης του κώδικα σε παράλληλα συστήματα με διαφορετική αρχιτεκτονική καθώς επίσης, και σε ανομοιογενή δίκτυα σταθμών εργασίας χωρίς προβλήματα συμβατότητας.
- Ανεξαρτησία από τη γλώσσα προγραμματισμού που χρησιμοποιούμε. Οι συναρτήσεις του MPI δεν εμπλέκονται με τις συναρτήσεις της γλώσσας. Για παράδειγμα, η συνάρτηση printf της C εξακολουθεί να λειτουργεί κανονικά, όταν εκτελείται ένα MPI πρόγραμμα.
- Εύκολα μεταφέρσιμο κώδικα. Αρκεί να υπάρχει η βιβλιοθήκη του MPI για το συγκεκριμένο σύστημα και ένας μεταγλωττιστής της C, C++ ή της FORTRAN.

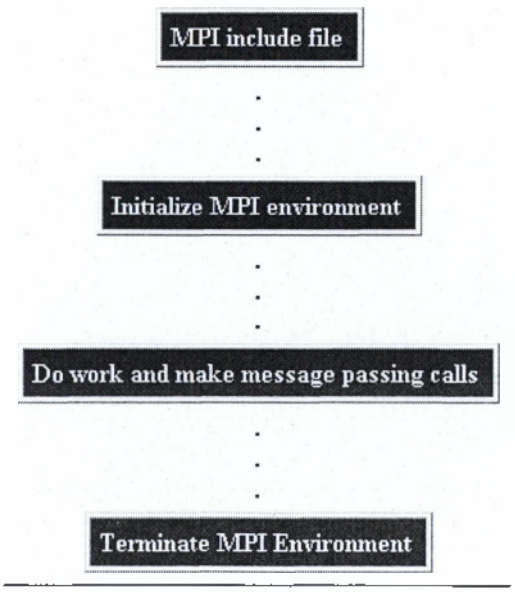
### 2.1.4 Τι παρέχεται με το MPI

Το MPI παρέχει υποστήριξη για τα παρακάτω: [18]

- Επικοινωνία κόμβου με κόμβο (point-to-point).
- Συλλογική επικοινωνία.
- Οργάνωση των διεργασιών σε ομάδες (group).
- Οργάνωση των επικοινωνιών σε communicators.
- Τοπολογίες επικοινωνίας διεργασιών.
- Δυνατότητα διασύνδεσης με C, C++ και FORTRAN.

### 2.1.5 Δομή Προγράμματος MPI

Η δομή προγράμματος του MPI φαίνεται στην Εικόνα 10. Πρώτα θα πρέπει να περιλαμβάνει το header file που μας προσφέρει το MPI, έπειτα έχουμε την αρχικοποίηση του περιβάλλοντος, στη συνέχεια την οποία δουλεία που έχουμε να εκτελέσουμε και τέλος το κλείσιμο του MPI περιβάλλοντος. [20]




Εικόνα 10. Δομή προγράμματος MPI [20]

# 2.2 MapReduce: Απλοποιημένη Επεξεργασία Δεδομένων σε Μεγάλα Συστήματα

Το MapReduce, είναι ένα προγραμματιστικό μοντέλο που επιτρέπει την επεξεργασία και την παραγωγή μεγάλων ποσοτήτων δεδομένων. Οι χρήστες ορίζουν μία συνάρτηση map η οποία επεξεργάζεται ένα ζεύγος key/value ώστε να παράγει ένα ενδιάμεσο ζεύγος key/value, και μία συνάρτηση reduce η οποία συγχωνεύει όλα τα ενδιάμεσα values που σχετίζονται με τα ίδια ενδιάμεσα keys. Προγράμματα τα οποία γράφονται με αυτό τον τρόπο, εκτελούνται σε ένα μεγάλο cluster αυτόματα και παράλληλα. [21]

## 2.2.1 Εισαγωγή

Η ανάγκη για επεξεργασία πάρα πολύ μεγάλων όγκων δεδομένων ταυτόχρονα σε κατανεμημένα συστήματα όπου υπάρχουν παράλληλες μηχανές και ο τρόπος διαχείρισης των πιθανών σφαλμάτων, απασχόλησε για μεγάλο χρονικό διάστημα τους ερευνητές. Στόχος ήταν η εύρεση μίας εύκολης λύσης, όπου θα υπήρχε εκτέλεση υψηλού επιπέδου εφαρμογών και αποδέσμευση των προγραμματιστών από πολύπλοκες υλοποιήσεις.

Η περίπτωση της μέτρησης της συχνότητας πρόσβασης μιας URL, που είναι δυνατό προσπελαύνεται από χιλιάδες χρήστες του διαδικτύου, ή της αυτόματης ταξινόμησης εγγραφών σε κατανεμημένο σύστημα που δέχεται συνεχώς αιτήσεις από διάφορους clients, ή του ανεστραμμένου ευρετηρίου μιας μηχανής αναζήτησης, η οποία εμπλουτίζεται συνεχώς με νέα έγγραφα, είναι ζητήματα για τα οποία η λύση της σειριακής επεξεργασίας θα ήταν απαγορευτική.

Προηγούμενες προσεγγίσεις προσπάθησαν να παρουσιάσουν συστήματα για παράλληλο υπολογισμό, περιορισμένων όμως δυνατοτήτων και χωρίς ανοχή σφαλμάτων, αφού αφήναν τις λεπτομέρειες διαχείρισης των αποτυχιών της μηχανής στους προγραμματιστές. Από την άλλη πλευρά, το MapReduce προτείνει παράλληλες αναγνώσεις για την επεξεργασία όγκου δεδομένων σε clusters από υπολογιστές γενικής χρήσης. Η συμβολή αυτού του προγραμματιστικού μοντέλου αφορά:

a) τη δημιουργία ενός framework που τρέχει σε μεγάλους clusters από εμπορικές μηχανές, υψηλής επεκτασιμότητας, και που δύναται να ανακάμψει από εσφαλμένες καταστάσεις,
b) την ευκολία και ευελιξία της χρήσης του συστήματος από τους προγραμματιστές, αφού όλες οι πολύπλοκες διαδικασίες γίνονται στο παρασκήνιο και οι χρήστες προσαρμόζουν τις συναρτήσεις map() και reduce() σύμφωνα με τις εκάστοτε ανάγκες τους. [21]

## 2.2.2 Προσέγγιση και αξιολόγηση

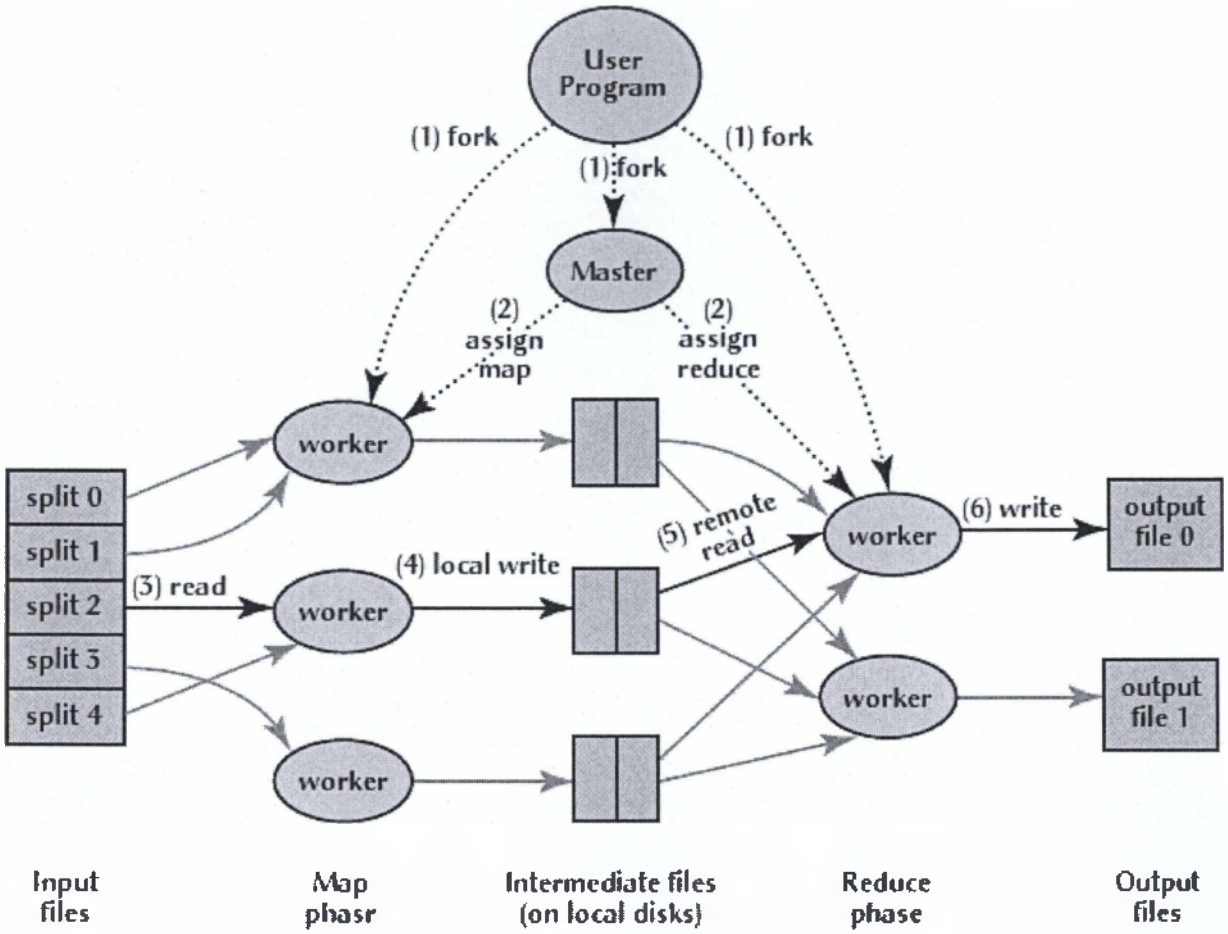
### *2.2.2.1 Παρουσίαση του MapReduce*

2.2.2.1.1 Τρόπος λειτουργίας

Η κεντρική ιδέα είναι η υλοποίηση ενός συστήματος που θα μπορεί να επεξεργάζεται παράλληλα και αποτελεσματικά πάρα πολύ μεγάλο όγκο δεδομένων που ολοένα και αυξάνεται, προσπαθώντας να αυξήσει την τοπικότητα της όλης διαδικασίας. Ο τρόπος λειτουργίας του MapReduce είναι ο ακόλουθος:

i. Η MapReduce βιβλιοθήκη στο πρόγραμμα του χρήστη, χωρίζει τα δεδομένα εισόδου σε M κομμάτια, και στη συνέχεια, αντίγραφα του προγράμματος επεξεργασίας αρχίζουν να τρέχουν σε μία ομάδα υπολογιστών.
ii. Ένα από αυτά τα αντίγραφα ονομάζεται αφέντης, ενώ τα υπόλοιπα εργάτες. Ο αφέντης είναι υπεύθυνος σε πρώτη φάση για την απόδοση των M (map) και R (reduce) εργασιών σε καθένα από τους εργάτες.
iii. Ένας εργάτης που του έχει ανατεθεί μία map εργασία, διαβάζει τα δεδομένα από το τμήμα που του αναλογεί και εξάγει ενδιάμεσα ζεύγη του τύπου key/value, τα οποία αποθηκεύονται στη μνήμη.
iv. Περιοδικά τα ενδιάμεσα ζεύγη key/value αποθηκεύονται στον τοπικό δίσκο, όπου μέσω συνάρτησης διαχωρισμού μοιράζονται σε R τμήματα. Οι τοποθεσίες αυτών των τμημάτων δίνονται στον αφέντη και αυτός με τη σειρά του τις δίνει στους R reduce εργάτες.
v. Ο reduce εργάτης διαβάζει επαναληπτικά την ταξινομημένη λίστα ζευγών key/value και για κάθε ενδιάμεσο key περνάει στη συνάρτηση reduce το key και τις ενδιάμεσες τιμές που σχετίζονται με αυτό. Η έξοδος της συνάρτησης reduce επισυνάπτεται σε ένα αρχείο εξόδου.
vi. Όταν οι παραπάνω διαδικασίες ολοκληρωθούν, ο αφέντης αφυπνίζει/ενημερώνει το πρόγραμμα του χρήστη.

Από την παραπάνω περιγραφή, παρατηρούμε ότι ο αφέντης χρειάζεται να πάρει O(M+R) προγραμματιστικές αποφάσεις και διατηρεί O(M*R) καταστάσεις στη μνήμη. [21]




Εικόνα 11 [22]

### 2.2.2.2 Παράδειγμα

Π.Χ. ένα πρόβλημα καταμέτρησης των λέξεων σε μια μεγάλη συλλογή των εγγράφων.

```
map(String key, String value):
        // key: document name
        // value: document contents
        for each word w in value:
                EmitIntermediate(w, "1");
reduce(String key, Iterator values):
        // key: a word
        // values: a list of counts
        int result = 0;
        for each v in values:
                result += ParseInt(v);
        Emit(AsString(result));
```

### 2.2.2.3 Ανοχή σφαλμάτων

Σε έναν σύστημα όπου υπάρχουν εκατοντάδες ή χιλιάδες μηχανές να τρέχουν παράλληλα, οι πιθανότητες να προκύψουν σφάλματα κατά την εκτέλεση είναι ένα λογικό σενάριο

Αποτυχία εργάτη: Ο αφέντης επικοινωνεί περιοδικά με κάθε εργάτη. Αν ο εργάτης δε απαντήσει μέσα σε συγκεκριμένο χρόνο θεωρεί ότι ο εργάτης απέτυχε. Σε αυτό το σημείο διακρίνουμε τρεις περιπτώσεις:

a) Ο εργάτης προτού αποτύχει είχε ολοκληρώσει μία map εργασία, τότε η εργασία ακυρώνεται και επαναλαμβάνεται από κάποιον άλλο εργάτη(τα αποτελέσματά της βρίσκονταν αποθηκευμένα στον τοπικό δίσκο του εργάτη που απέτυχε),
b) Ο εργάτης προτού αποτύχει είχε ολοκληρώσει την reduce εργασία, τότε η εργασία δεν εκτελείται ξανά διότι τα αποτελέσματά της έχουν ήδη αποθηκευτεί στο global σύστημα,
c) Ο εργάτης τη στιγμή που απέτυχε εκτελούσε map ή reduce εργασία, τότε η εργασία ακυρώνεται και επαναλαμβάνεται από την αρχή από κάποιον άλλο εργάτη.

Οι reduce εργασίες που βρίσκονται σε εξέλιξη ενημερώνονται κατάλληλα σε περίπτωση που η map εργασία από την οποία προμηθεύονται τα δεδομένα εκτελεσθεί από την αρχή από κάποιον άλλο εργάτη.

Αποτυχία αφέντη: Σε αυτή την περίπτωση όλες οι υπολογιστικές διαδικασίες του MapReduce ακυρώνονται αν και η αποτυχία του κόμβου αφέντη είναι σπάνιο φαινόμενο.

Επιπλέον, η σημασία της παρουσίας σφαλμάτων έχει να κάνει και με τον τρόπο καθορισμού των map και reduce συναρτήσεων από το χρήστη, δηλαδή αν αυτές θα είναι ντετερμινιστικές ως προς τις εισαγόμενες τιμές τους ή μη-ντετερμινιστικές. Στην πρώτη περίπτωση, η κατανεμημένη εφαρμογή παράγει συνεχώς το ίδιο αποτέλεσμα όπως θα παραγόταν από μία μη-λανθασμένη σειριακή εκτέλεση του συνολικού προγράμματος. Αντίθετα, στη δεύτερη περίπτωση, το αποτέλεσμα εξαρτάται από κάθε φορά από τα δεδομένα. [21]

### 2.2.2.4 Εργασίες Backup

Μία συνηθισμένη αιτία που προκαλεί αύξηση του συνολικού χρόνου εκτέλεσης είναι ο "straggler": μία μηχανή που χρειάζεται ασυνήθιστα μεγάλο χρόνο για να ολοκληρώσει μία από τις λίγες τελευταίες map ή reduce εργασίες υπολογισμού της. Η λύση που προτείνεται είναι η εξής: όταν μια λειτουργία MapReduce κοντεύει στο τέλος της, ο αφέντης προγραμματίζει backup εκτελέσεις των υπολοίπων εργασιών που βρίσκονται σε εξέλιξη. Η εργασία μαρκάρεται ως ολοκληρωμένη όταν η αρχική ή η backup εκτέλεση ολοκληρωθεί. Μετρήσεις κατέγραψαν έως και 44% μείωση του χρόνου ολοκλήρωσης της εργασίας χρησιμοποιώντας την backup τεχνική. [21]

#### 2.2.2.5 Επεκτάσεις

Οι επεκτάσεις στο ήδη υπάρχον σύστημα προσφέρονται ως επιπλέον λύσεις στις ανάγκες των προγραμματιστών, και τους δίνουν τις δυνατότητες: [21]

- τροποποίησης της συνάρτησης διαχωρισμού,
- εύκολης δημιουργίας ταξινομημένου εξερχόμενου αρχείου ανά τμήμα,
- προσδιορισμού προαιρετικής Combiner συνάρτησης που κάνει μερική συγχώνευση των δεδομένων πριν την αποστολή τους μέσω δικτύου,
- υποστήριξης για ανάγνωση δεδομένων διαφόρων μορφών,
- παραγωγής βοηθητικών αρχείων ως επιπρόσθετα outputs των map ή/και reduce λειτουργιών,
- παράλειψης εγγραφών που μπορεί να προκαλέσουν προβλήματα στη λειτουργία του συστήματος,
- τοπικής εκτέλεσης όλων των εργασιών μιας MapReduce λειτουργία για ευκολία debugging, profiling και μικρής έκτασης testing,
- αναφοράς κατάστασης συστήματος μέσω status pages,
- καταμέτρησης γεγονότων προκληθέντων από διάφορα συμβάντα.

#### 2.2.2.6 Πειράματα και αποτελέσματα

Για την πειραματική αξιολόγηση του συστήματος έγιναν δύο υπολογισμοί (αναζήτησης και ταξινόμησης) που εκτελέστηκαν σε ένα μεγάλο cluster μηχανών με περίπου 1TB δεδομένα.

Αναζήτηση: Το grep πρόγραμμα σαρώνει 1010 100-byte εγγραφές, αναζητώντας ένα σχετικά σπάνιο τριών-χαρακτήρων pattern. Το input χωρίζεται σχεδόν σε 64MB κομμάτια και ολόκληρο το output τοποθετείται σε ένα αρχείο. Ο ρυθμός μεταφοράς δεδομένων σε σχέση με το χρόνο αυξάνεται σταδιακά κατά τη διάρκεια των map εργασιών και μέχρι την ολοκλήρωσή τους, ενώ έπειτα μειώνεται γρήγορα μέχρι που μηδενίζεται. Η ολοκλήρωση του υπολογισμού απαιτεί περίπου 150sec.

Ταξινόμηση: Το sort πρόγραμμα ταξινομεί 1010 100-byte εγγραφές. Μία map συνάρτηση εξάγει ένα 10-byte ταξινομημένο key και εκπέμπει το πραγματικό κείμενο ως το ενδιάμεσο ζεύγος key/value. Γίνεται χρήση μίας Identity συνάρτησης κατά τη reduce λειτουργία, η οποία περνάει τα ενδιάμεσα μη-αλλαγμένα ζεύγη key/values ως output ζεύγη key/values. Το τελικό ταξινομημένο output γράφεται σε ένα σετ 2-τρόπων κατασκευασμένα GFS αρχεία. Ο input ρυθμός είναι υψηλότερος από το ρυθμό ανακατάταξης και τον output ρυθμό λόγω της βελτιστοποίησης της τοπικότητας, ενώ ο ρυθμός ανακατάταξης είναι υψηλότερος από τον output ρυθμό γιατί η output φάση γράφει δύο αντίγραφα των ταξινομημένων δεδομένων. [22]

### 2.2.3 Πλεονεκτήματα - Μειονεκτήματα

#### 2.2.3.1 Δυνατά σημεία

Το MapReduce αποτελεί μία καινοτομία στο χώρο της παράλληλης επεξεργασίας πολύ μεγάλων όγκων δεδομένων, με πιο δυνατά σημεία τα εξής:

1. Επιτρέπεται η εκτέλεση λειτουργιών map/reduce με χρήση κατανεμημένης επεξεργασίας, δεδομένου ότι η κάθε λειτουργία map που γίνεται είναι ανεξάρτητη από άλλες, όλο το mapping μπορεί να γίνει παράλληλα.
2. Μπορεί να γίνεται επεξεργασία πολύ μεγαλύτερου όγκου δεδομένων από αυτόν που ένας απλός εξυπηρετητής μπορεί να επεξεργαστεί.
3. Σε περίπτωση που ένας εξυπηρετητής είναι αργός, ή αδυνατεί να επεξεργαστεί τα δεδομένα που του ανατέθηκαν, γίνεται εκ νέου προγραμματισμός (rescheduling) της εργασίας του αυτόματα.

#### 2.2.3.2 Αδύνατα σημεία

Εν συνεχεία, αναφέρουμε κάποια αδύνατα σημεία που εντοπίσαμε στα δύο άρθρα:

1. Το MapReduce θα μπορούσε να χαρακτηριστεί λιγότερο αποδοτικός σε σχέση με άλλους αλγορίθμους που είναι περισσότερο γραμμικοί.
2. Στην αξιολόγηση του συστήματος δίνονται μετρήσεις χρόνου εκτέλεσης για πολλούς διαφορετικούς αλγόριθμους που έχουν γραφτεί χρησιμοποιώντας το MapReduce, αλλά δεν υπάρχει τιμή αναφοράς που να συγκρίνει αυτές τις μετρήσεις.

# Κεφάλαιο 3

# Σύγκριση των grid/cloud computing frameworks

Υπάρχουν διάφορα frameworks ανοικτού κώδικα, τα οποία μπορούν να χρησιμοποιηθούν σχεδόν χωρίς καμιά ρύθμιση. Απλά κατεβάζουμε τη βιβλιοθήκη, δημιουργούμε τον πηγαίο κώδικα και να παρατηρούμε πόσο γρήγορα επιλύονται τα προβλήματά. Αλλά ποιο framework θα πρέπει να χρησιμοποιήσουμε; Η απάντηση δεν είναι απλή και εξαρτάται σε μεγάλο βαθμό από τις ατομικές μας ανάγκες. Δεν μπορούμε να πούμε, ότι, για παράδειγμα το Hadoop θα είναι πάντα η καλύτερη επιλογή, αλλά μπορούμε να επισημάνουμε τα πλεονεκτήματα και τα μειονεκτήματα των διαφόρων framework, έτσι θα είμαστε σε θέση να επιλέξουμε το καλύτερο για τις ανάγκες μας. Έχουμε αποφασίσει να δοκιμάσουμε τα ακόλουθα frameworks:

- Hadoop [23]
- GridGain [24]

## 3.1 Μεθοδολογία

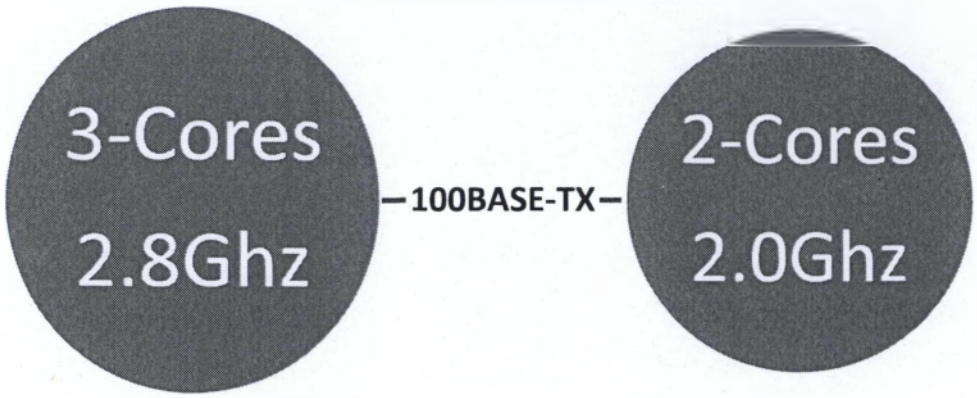
Στο benchmark που πραγματοποιήσαμε βασιστήκαμε στο μαθηματικό πρόβλημα της λειτουργιάς καταμέτρησης δυαδικών μονότονων (counting monotone boolean functions) επίσης γνωστό ως το πρόβλημα του Dedekind. Γιατί επιλέξαμε αυτό το πρόβλημα; Επειδή: [25]

- Καταναλώνει αρκετή από τη δύναμη της cpu (μια cpu χρειάζεται περισσότερες από 800 ώρες ώστε να κάνει καταμέτρηση σε όλα δυαδικά μονότονα για N=8)
- Δεν χρειάζεται να λύσουμε όλο το πρόβλημα στο benchmark (διαλέγουμε ένα κομμάτι ώστε μια cpu να μπορέσει να το υπολογίσει σε περισσότερες από 3 ώρες.
- Είναι εύκολο να διαιρεθεί σε ένα αυθαίρετο αριθμό εργασιών.
- Δημιουργεί εργασίες που έχουν διαφορετικές υπολογιστικές ανάγκες (είναι μια τέλεια περίπτωση για εξισορρόπηση φορτίου (load balance)

Με ένα τέτοιο ευέλικτο πρόβλημα στη διάθεση μας, αποφασίσαμε να πραγματοποιήσουμε 3 τεστ:

- Ένα υπολογιστικό πρόβλημα διαιρεμένο σε 800 εργασίες (CMBF με παραμέτρους : n=3, επίπεδο = 1000)
- Ένα υπολογιστικό πρόβλημα διαιρεμένο σε 64 εργασίες (CMBF με παραμέτρους : n=3, επίπεδο = 10000)
- Ένα υπολογιστικό πρόβλημα διαιρεμένο σε 10 εργασίες (CMBF με παραμέτρους : n=3, επίπεδο = 100000)

Όλα τα τεστ τα επαναλάβαμε δέκα φόρες ώστε να αποφύγουμε κάποια λανθασμένη μέτρηση. Χρησιμοποιήσαμε δυο υπολογιστές με πέντε πυρήνες στο σύνολο.




Εικόνα 12

## 3.2 Hadoop 0.20.2

Το Hadoop είναι μια πλατφόρμα λογισμικού η οποία μπορεί να χρησιμοποιηθεί για την ανάλυση και επεξεργασία μεγάλου αριθμού δεδομένων επιπέδου petabyte. Αυτό που κάνει το hadoop είναι κατανέμει τα δεδομένα και τη διαδικασία ανάλυση τους σε ομάδες υπολογιστών (Clusters) ώστε να επεξεργαστούν παράλληλα τα δεδομένα, επιταχύνοντας έτσι τις διαδικασίες.

Το Hadoop χρησιμοποιεί το προγραμματιστικό μοντέλο MapPeduce το οποίο αναπτύχθηκε πρώτα από την Google για την υποστήριξη του συστήματος της στην ανάλυση δεδομένων. Αυτό που κάνει αυτή η τεχνική είναι, να μεταβιβάσει τα δεδομένα και το πρόβλημα στον υπολογιστή master του cluster, και αυτός στη συνέχεια θα διασπάσει το πρόβλημα σε μικρότερα προβλήματα και κάθε μικρότερο πρόβλημα θα το προωθήσει σε κάθε ένα από τους υπόλοιπους υπολογιστές του Cluster. Ο κάθε υπολογιστής του cluster θα επιλύσει το δικό του υποπρόβλημα που του έχει ανατεθεί και θα επιστρέψει τη λύση στον master υπολογιστή ο οποίος θα συνδυάσει τις λύσεις των υποπροβλημάτων για να βρει τη λύση στο πρόβλημα που το δόθηκε.

Το Hadoop βασίζεται στην παραπάνω τεχνική με επιπρόσθετα πλεονεκτήματα όπως ότι καταφέρνει να ανακτήσει δεδομένα σε περίπτωση που ένας υπολογιστής του cluster πάθει ζημιά και να μεταβιβάσει το υποπρόβλημα σε άλλον υπολογιστή. Ήδη η Yahoo χρησιμοποιεί το hadoop και πιο πρόσφατα το εγκατέστησε και το facebook.

Το Hadoop 0.20.2 διαφέρει από το άλλο framework που δοκιμαστικέ γιατί διαιρεί το πρόβλημα από μονό του. Χρησιμοποιήσαμε το mapred-site.xml με τις ακόλουθες ρυθμίσεις

- dfs.replication = 1
- mapred.map.tasks = 20
- mapred.reduce.tasks = 10

Αυτό μας επιτρέπει να χρησιμοποιήσουμε όλες τις διαθέσιμες υπολογιστικές μονάδες. Το Hadoop είναι ένα κεντρικοποιημενο framework (με ειδικούς κύριους κόμβους), γι' αυτό πρέπει επίσης να τρέξουμε πρόσθετες διαδικασίες σε μία από τις μηχανές. Μπορούμε να δούμε την αρχιτεκτονική του περιβάλλοντος δοκιμών στον ακόλουθο κώδικα:

### 3.2.1 Κώδικας

Το Hadoop λειτουργεί με υπολογιστικές μονάδες που ονομάζονται εργασίες (jobs). Οι εργασίες χρησιμοποιούν ειδικές κλάσεις που ονομάζονται **Mappers** και **Reducers**. Χρησιμοποιούμε το Mapper για να διαιρέσουμε το πρόβλημα σε μικρότερα καθήκοντα:

```
public static class CMBFMapper extends Mapper<IntWritable, IntWritable, Text,
FastBigInt128> {
  @Override
  public void map(IntWritable i, IntWritable lvl, Context context)
    throws IOException, InterruptedException {
    for ( final String[] imageDesc : new Worker().generateImages(i.get(), lvl.get())) {
      StringBuilder buffer = new StringBuilder(imageDesc[0]);
      for (int j = 1; j < imageDesc.length; ++j) {
        buffer.append("^");
        buffer.append(imageDesc[j]);
      }
      context.write(new Text(buffer.toString()), new FastBigInt128());
    }
  }
}
```

Μετά από αυτό χρησιμοποιούμαι το Reducer ώστε να υπολογίσουμε το αποτέλεσμα:

```
public static class CMBFReducer extends Reducer<Text, FastBigInt128,
WritableComparable<?>, Writable> {
  @Override
  public void reduce(Text imageDesc, Iterable<FastBigInt128> values, Context context)
    throws IOException, InterruptedException {
    FastBigInt128 res = new Worker().countInImage(imageDesc.toString().split("\\^"));
    context.write(imageDesc, res);
  }
}
```

### 3.2.2 Αποτελέσματα

Μπορούμε να δούμε τα αποτελέσματα με τον αριθμό των εργασιών και το μέσο χρόνο τους στον ακόλουθο πινάκα:

| | Tasks: 800 | Tasks: 64 | Tasks: 10 |
|---|---|---|---|
| 1 | 81261 | 84049 | 81089 |
| 2 | 80014 | 85093 | 82042 |
| 3 | 85179 | 85027 | 82048 |
| 4 | 88146 | 84002 | 81087 |
| 5 | 82122 | 85035 | 81061 |
| 6 | 82045 | 85168 | 81059 |
| 7 | 81064 | 85107 | 82102 |
| 8 | 80125 | 84979 | 81071 |
| 9 | 81016 | 85033 | 82086 |
| 10 | 81059 | 85156 | 82015 |
| **Average** | **82207.1** | **84864.9** | **81566** |

## 3.3 GridGain 2.1.1

Οι GridGain 2.1.1 κόμβοι είναι multi-threaded. Λόγω αυτού, πρέπει να ξεκινήσουμε μία μόνο διεργασία σε κάθε μηχάνημα, ώστε να χρησιμοποιήσει πλήρως τις διαθέσιμες μονάδες επεξεργασίας. Το GridGain είναι πλήρως κατανεμημένο, γι 'αυτό δεν χρειάζεται να ξεκινήσουμε κάποιες επιπλέον διεργασίες. Μπορούμε να δούμε την αρχιτεκτονική του περιβάλλοντος δοκιμών στον ακόλουθο κώδικα:

### 3.3.1 Κώδικας

Ετοιμάσαμε δυο εκδόσεις των δοκίμων με το GridGain:

- με GridTasks
- με ExecutorService

### 3.3.2 Δοκιμή I - Χρησιμοποιώντας GridTasks

Το GridGain λειτούργει βασισμένο στα GridTasks. Η εκτέλεση μιας τέτοιας εργασίας είναι πολλή απλή:

```
Grid grid = GridFactory.getGrid();
GridTaskFuture<FastBigInt128> future = grid.execute(CMBFtask.class, args);
log.info("Final result = " + future.get());
```

In order to perform computations, we had to divide problem into small tasks in the split method:

```
public Collection<? extends GridJob> split(int gridSize, String[] arg) throws GridException {
  List<GridJob> jobs = new ArrayList<GridJob>();
  int n = arg.length > 0 ? Integer.parseInt(arg[0]) : 3;
  int level = arg.length > 1 ? Integer.parseInt(arg[1]) : 1000;
  int k = 1;
  Worker cmbfWorker = new Worker();
  log.info("Generating tasks (n=" + n + ", level=" + level + ")");
  for (int i = 1; i <= n; i++) {
    for (final String[] imageDesc : cmbfWorker.generateImages(i, level)) {
      jobs.add(new GridJobAdapter<Integer>(k++) {
        public Serializable execute() throws GridException {
          int k = getArgument();
          Worker cmbfWorkr = new Worker();
          FastBigInt128 result = cmbfWorkr.countInImage(imageDesc);
          log.info("Image #" + k + " (" + sentTasks + "), result = " + result);
          return result;
        }
      });
      sentTasks++;
    }
  }
  log.info("Sent " + sentTasks + " tasks. Receiving results...");
  return jobs;
}
```

Μετά πρέπει να συλλέξουμε τα αποτελέσματα στην μέθοδο reduce:

```
public FastBigInt128 reduce(List<GridJobResult> results) throws GridException {
  FastBigInt128 result = new FastBigInt128(0);
  for (GridJobResult res : results) {
    FastBigInt128 charCnt = res.getData();
    result.add(charCnt);
  }
  return result;
}
```

### 3.3.3 Δόκιμη II - Χρησιμοποιώντας ExecutorService

Η κατανεμημένη υπηρεσία εκτέλεσης(executor service) του GridGain λειτούργει χρησιμοποιώντας τη Callable/Runnable διεπαφή. Η κεντρική εργασία γίνεται στην κλάση **Agent** που υλοποίει τη Callable διεπαφή:

```
public class Agent implements Callable<FastBigInt128>, Serializable {
   private static final long serialVersionUID = 1L;
   private String[] imageDesc;
   private int z;
   public Agent(String[] imageDesc, int z) {
      this.imageDesc = imageDesc;
      this.z = z;
   }
   public FastBigInt128 call() {
      Worker cmbfWorker = new Worker();
      FastBigInt128 result = cmbfWorker.countInImage(imageDesc);
      System.out.println("\tResult from task #" + z + ", " + "\tvalue: " + result);
      return result;
   }
}
```

Για να γίνουν οι υπολογισμοί, πρέπει να διαιρέσουμε το πρόβλημα σε μικρότερες διεργασίες και να τις υποβάλουμε στην υπηρεσία εκτέλεσης:

```
for (int i = 1; i <= n; i++) {
   for (final String[] imageDesc : cmbfWorker.generateImages(i, level)) {
      Future<FastBigInt128> future = executorService.submit(new Agent(imageDesc,
++count));
      tasks.add(future);
   }
}
After that we had to gather results:
for (Future<FastBigInt128> w : tasks) {
   results.add(w.get());
}
```

### 3.3.4 Αποτελέσματα

Μπορούμε να δούμε όλα τα αποτελέσματα και τους μέσους όρους των χρονών στους ακολούθους πινάκες:

### 3.3.5 Αποτελέσματα Δοκιμής I - Χρησιμοποιώντας GridTasks

| | Tasks: 800 | Tasks: 64 | Tasks: 10 |
|---|---|---|---|
| 1 | 62810 | 56047 | 49753 |
| 2 | 61496 | 56382 | 62872 |
| 3 | 60048 | 55348 | 50590 |
| 4 | 60790 | 57091 | 56643 |
| 5 | 61269 | 55330 | 56376 |
| 6 | 60893 | 56827 | 50849 |
| 7 | 59621 | 55239 | 57563 |
| 8 | 61696 | 54847 | 50268 |
| 9 | 61931 | 55222 | 57253 |
| 10 | 62316 | 56911 | 56690 |
| Average | **61287** | **50824.4** | **54885.7** |

Πίνακας 4

### 3.3.6 Αποτελέσματα Δοκιμής II - Χρησιμοποιώντας ExecutorService

| | Tasks: 800 | Tasks: 64 | Tasks: 10 |
|---|---|---|---|
| 1 | 62296 | 51769 | 64335 |
| 2 | 67217 | 50524 | 65650 |
| 3 | 64443 | 63061 | 49634 |
| 4 | 61252 | 50985 | 54921 |
| 5 | 58787 | 66548 | 50477 |
| 6 | 64637 | 64753 | 61256 |
| 7 | 66276 | 65860 | 71488 |
| 8 | 67417 | 53761 | 52453 |
| 9 | 61814 | 67561 | 47007 |
| 10 | 60965 | 63301 | 69109 |
| Average | **63510.4** | **59812.3** | **58633** |

Πίνακας 5

## 3.4 Συμπεράσματα

Λόγο των περιορισμών του περιβάλλοντος (μονό 5 επεξεργαστές), και τα 2 framework παρουσίασαν σχεδόν παρόμοια αποτελέσματα. Ωστόσο οι κατανεμημένες εργασίες του Hadoop ήταν πιο αργές από το GridGain, αλλά το Hadoop έχει σχεδιαστεί για να χειρίζεται μεγάλα σύνολα δεδομένων, όποτε τα παραπάνω αποτελέσματα είναι απολύτως κατανοητά. Ωστόσο υπάρχουν κάποια σημεία που πρέπει να σημειώσουμε. Το Hadoop χρειάστηκε περισσότερες τροποποιήσεις στα αρχεία ρυθμίσεων του framework αλλά και στα αρχεία ρυθμίσεων του λειτουργικού συστήματος. Αντιθέτως το GridGain το μονό που χρειάστηκε για να λειτουργήσει ήταν η συγγραφή του κώδικα. Το ξεκίνημα των κόμβων ήταν αρκετά εύκολο (εκτελούμε ένα script αρχείο) ενώ το Hadoop είχε ανάγκη το ξεκίνημα πολλών και διαφορετικών διεργασιών σε κάθε κόμβο.

# Κεφάλαιο 4

## GridGain

# GridGain

Το GridGain είναι μια ανοικτή πλατφόρμα Cloud. Με την εικονικοποίηση (virtualization) του hardware που προσφέρουν οι cloud πάροχοι το GridGain παρέχει στους προγραμματιστές μια δυναμική τεχνολογία για την ανάπτυξη και εκτέλεση των εφαρμογών σε cloud. Το GridGain έχει αναπτυχθεί σε Java και αποτελεί μια επέκταση στις τελευταίες μεθοδολογίες ανάπτυξης με java συμπεριλαμβανομένου των annotations, του Spring framework και την εκτέλεση των εφαρμογών σε πλέγμα βασιζόμενο στο AOP.

Το GridGain παρέχει ένα middleware σε Java που επιτρέπει την ανάπτυξη πολύπλοκων εφαρμογών grid σε μια υποδομή cloud με απλό τρόπο.

## 4.1 Η Αρχιτεκτονική του GridGain

### 4.1.1 Public API

Στην κορυφή του GridGain βρίσκεται το public Grid API με το οποίο αλληλεπιδρά ο χρήστης. Μέσω αυτού ο χρήστης έχει τη δυνατότητα να εκτελεί εργασίες(tasks), να πάρει ειδικές πληροφορίες για το πλέγμα, όπως η τοπολογία ή γεγονότα, να στείλει άμεσα μηνύματα στους απομακρυσμένους κόμβους, κ.λπ..

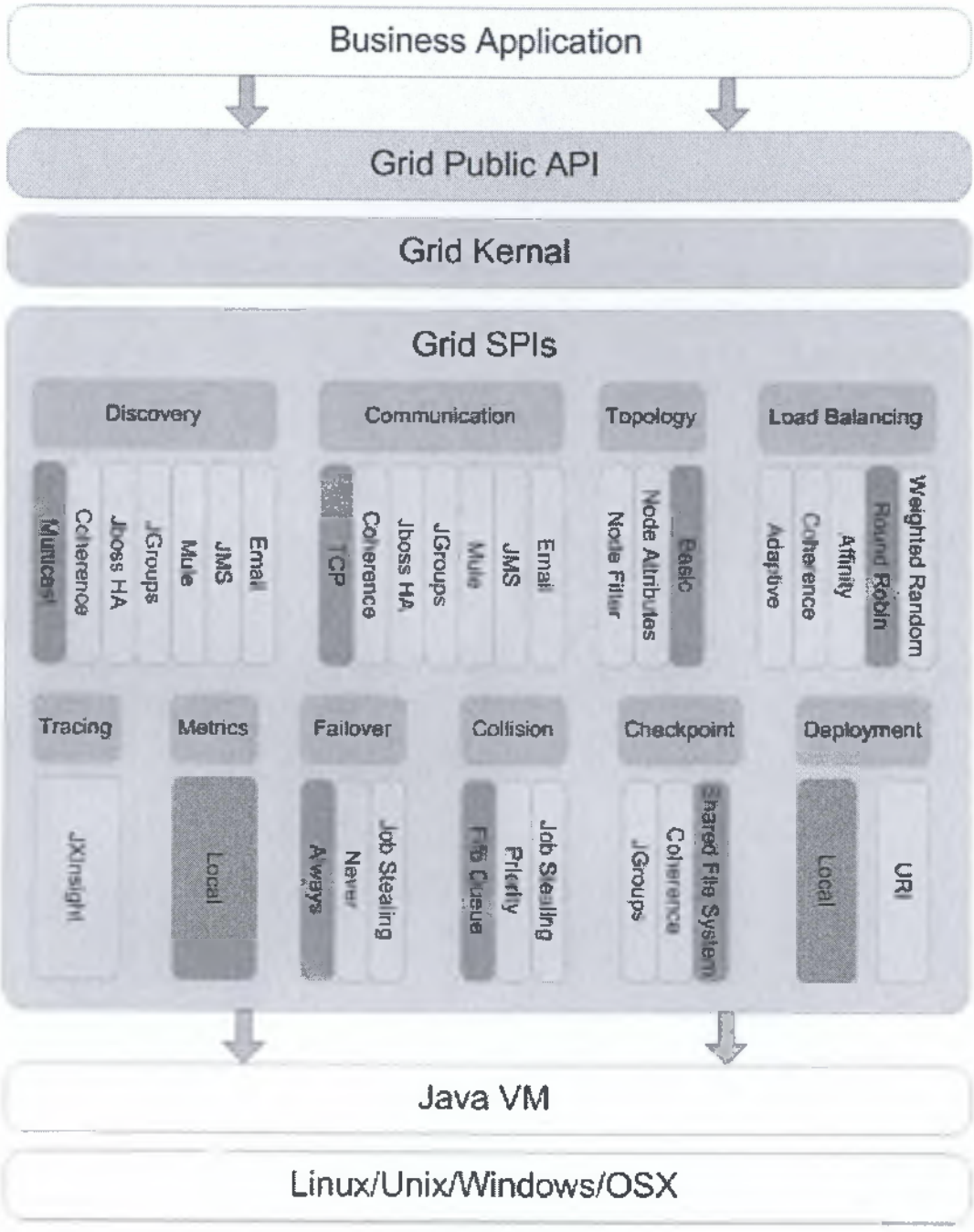
### 4.1.2 Grid Kernal

"Κάτω από το public API βρίσκεται ο Grid Kernal. Να σημειώσουμε ότι είναι επίτηδες λανθασμένη η ορθογραφία, Kernal αντί του πυρήνα (Kernel). Ο πυρήνας GridGain Kernal ονομάστηκε προς τιμήν του λειτουργικού συστήματος του παλαιού 8-bit Commodore υπολογιστή. Η ευθύνη του Kernal είναι να παρέχει μια υλοποίηση για το GridGain public API και τη μεταβίβαση της απαραίτητης λειτουργικότητας στο SPI Layer." [24]

### 4.1.3 Service Provider Interfaces

SPI (Service Provider Interface) Layer είναι ένα σύνολο από Java διεπαφές (interfaces). Το GridGain περιέχει πολλά SPI. Τα SPIs χρησιμοποιούνται για να παρέχουν διάφορες διεπαφές για την εύρεση των κόμβων, την επικοινωνία, την ανάλυση της τοπολογία, σημεία ελέγχου κάποιας εργασία, τον προγραμματισμό και την ανάλυση τυχόν συγκρούσεων ανάμεσα στις εργασίες, εξισορρόπηση φορτίου, fail-over, κ.λπ... Τα SPIs χρησιμοποιούνται από το GridGain εσωτερικά και δεν είναι ανοιχτά για δημόσια χρήση. Επιτρέπουν στο GridGain να κάνει εύκολη εναλλαγή, μεταξύ της εύρεσης με Multicast και την εύρεση με βάση τα JGroups απλά αλλάζοντας μερικές γραμμές στο αρχείο ρυθμίσεων.

Το παρακάτω διάγραμμα απεικονίζει τα αρχιτεκτονικά στρώματα του GridGain και τις υλοποιήσεις SPI που παρέχονται. Τα SPI που είναι χρωματισμένα με γκρι χρώμα είναι αυτά που ξεκινάμε από προεπιλογή χωρίς αλλαγές στης ρύθμισης των παραμέτρων του GridGain.


Business Application
Grid Public API
Grid Kernal
Grid SPIs
Discovery
Communication
Topology
Load Balancing
Email
JMS
Mule
JGroups
Jboss HA
Coherence
Multicast
Email
JMS
Mule
JGroups
Jboss HA
Coherence
TCP
Node Attributes
Node Filter
Basic
Weighted Random
Round Robin
Affinity
Coherence
Adaptive
Tracing
Metrics
Failover
Collision
Checkpoint
Deployment
JXInsight
Local
Job Stealing
Never
Always
Job Stealing
Priority
FIFO Queue
Shared File System
Coherence
JGroups
Local
URI
Java VM
Linux/Unix/Windows/OSX

**Εικόνα 13 [24]**

## 4.2 Τι είναι το GridGain

Το GridGain είναι ένα προϊόν υπολογιστικού πλέγματος. Μας επιτρέπει την παράλληλη εκτέλεση κώδικα σε ένα σύνολο υπολογιστικών πόρων. Πόροι μπορεί να είναι από ένα laptop ένα desktop computer , ένας server, ένα mainframe ή όποια συσκευή υποστηρίζει την java 5 ή μεταγενέστερη. Το σύνολο των υπολογιστικών πόρων μπορεί να είναι ομοιογενές (ίδιο είδος) ή ετερογενής (διαφορετικά είδη), μπορούν να βρίσκονται σε τοπικό επίπεδο, στο εσωτερικό των επιχειρήσεων, σε παγκόσμιο επίπεδο ή οποιοδήποτε συνδυασμό των παραπάνω. [24]

## 4.3 Τι δεν είναι το GridGain

### 4.3.1 Το GridGian δεν είναι μια λύση εικονοποίησης υλικού

Παρά το γεγονός ότι GridGain μπορεί να προσφέρει ένα ενοποιημένο προγραμματιστικό και υπολογιστικό μοντέλο σε ένα πολύ διαφορετικό σύνολο υπολογιστικών πόρων, δεν είναι λύση virtualization υλικού. Συνήθως όταν αναφερόμαστε στο virtualization υλικού συνήθως εννοούμε κάτι σαν το VMWare και το Xen, το λογισμικό που μπορεί να χρησιμοποιεί πολλαπλά λειτουργικά συστήματα ταυτόχρονα και προσομοιώνει με αυτό τον τρόπο το υλικό στον χρήστη. Το GridGain προσφέρει virtualization σε ένα διαφορετικό επίπεδο. Είναι εικονοποιημένο σε ένα σύνολο από πόλων διαφορετικών υπολογιστικών πόρων μέσω της παροχής ενός μοντέλου προγραμματισμού Java για τη σύνταξη ενός λογισμικού ανεξαρτήτου κλίμακας υπολογιστών (αν πρόκειται να τρέχει σε έναν υπολογιστή ή στο εκατοντάδες υπολογιστές). [24]

### 4.3.2 Το GridGain δεν είναι μια ESB λύση

Τα ESBs (Enterprise Service Bus) συνδέουν συνήθως πολλούς υπολογιστές παρέχοντας μια ενιαία αποστολή μηνυμάτων και κλήση υπηρεσιών και, συνεπώς, συχνά συγχέεται με το Grid Computing. Αυτές οι δύο κατηγορίες middleware επιλύουν διαφορετικά προβλήματα. Συχνά, αν όχι στις περισσότερες περιπτώσεις, το ESB χρησιμοποιείται μαζί με το Grid Computing, το ESB ενεργεί σαν ένας ενδιάμεσος (ή δίαυλος) με τον οποίο οι κομβόι του πλέγματος ανταλλάσουν δεδομένα. [24]

### 4.3.3 Το GridGain δεν είναι μια λύση κατανεμημένων δεδομένων

Το GridGain είναι ένα υπολογιστικό cloud. Επιτρέπει την παραλληλοποίηση των εργασιών. Τα cloud δεδομένων κάνουν το ίδιο με τα δεδομένα - παραλληλοποίουν τα δεδομένα μέσω του δικτύου παρέχοντας υψηλή διαθεσιμότητα μέσω του αναδιπλασιασμού (ή με caching) σε ένα σύνολο αποθηκευμένων πόρων. Όπως μπορούμε να δούμε οι δύο κατηγορίες προϊόντων είναι εντελώς διαφορετικές στην επίλυση των προβλημάτων. Παρά τις διαφορές τους, τόσο τα υπολογιστικά cloud όσο και τα cloud δεδομένων σχεδόν πάντα χρησιμοποιούνται μαζί. Πράγματι, για να εκτελεστεί σχεδόν οποιαδήποτε εργασία που θα χρειαστεί μερικά δεδομένα. Έτσι τα πλέγματα δεδομένων παρέχουν τον πιο αποτελεσματικό τρόπο για να πάρουμε τα δεδομένα σε ένα κατανεμημένο περιβάλλον. [24]

## 4.5 Grid Computing

Το πλέγμα συνήθως ορίζεται από δύο κατηγορίες: τα πλέγματα δεδομένων και τα υπολογιστικά πλέγματα. Με λίγα λόγια, τα πλέγματα δεδομένων διανέμουν δεδομένα και τα υπολογιστικά πλέγματα είναι υπεύθυνα για τη διανομή της επεξεργασίας. Και οι δύο τύποι του πλέγματος προσπαθούν να βελτιώσουν τις επιδόσεις ή και την επεκτασιμότητα της εφαρμογής που τρέχουν στο πλέγμα. Το GridGain είναι ένα framework υπολογιστικού πλέγματος . [24]

### 4.5.1 Επεξεργαστικά Grids

Τα υπολογιστικά πλέγματα κατέχουν ένα μεγάλο ποσοστό της χρήσης του Grid. Με λίγα λόγια, η ιδέα πίσω από αυτό είναι πολύ απλή: αν έχουμε μια εργασία που εκτελείτε για πολύ μεγάλο χρονικό διάστημα, μπορούμε να χωρίσουμε το έργο αυτό σε πολλαπλά καθήκοντα εκτελώντας τα επιμέρους έργα παράλληλα σε ξεχωριστούς υπολογιστές, και στην συνεχεία συνδυάζοντας τα αποτελέσματα από τα υπό-καθήκοντα παίρνουμε το αποτέλεσμα O (N)-φορές ταχύτερα, όπου N είναι ο αριθμός των υπό-καθηκόντων. Τα υπολογιστικά πλέγματα κατέχουν ένα μεγάλο μερίδιο από τις εφαρμογές που εκτελούνται στο πλέγμα [24]

### 4.5.2 Grids Δεδομένων

Τα πλέγματα δεδομένων επιτρέπουν το διαχωρισμό των αποθηκευμένων δεδομένων σε πολλούς υπολογιστές. Όπως τα υπολογιστικά πλέγματα που διαχωρίζουν μια εργασία, επιτρέπουν την τοποθέτηση των δεδομένων σε πολλούς υπολογιστές η σε αποθηκευτικές συστοιχίες και τη διάθεση τους σαν ένα ενιαίο σύνολο.

Αυτή η λειτουργιά παρέχει επίσης και τη δυνατότητα κατανεμημένης προσωρινής αποθήκευσης (caching).

## 4.6 Map/Reduce επίσης γνωστό ως Split και Aggregate

Όπως και τα παραδοσιακά συστήματα HPC (όπως το MPI), έχουν την ικανότητα να διαχωρίζουν μια διαδικασία σε μια σειρά από υπό-διαδικασίες, εκτελώντας αυτές τις υπό-διαδικασίες παράλληλα, και τα επιμέρους αποτελέσματα να ενώνονται τελικά σε ένα αποτέλεσμα (σημειώστε ότι η διάσπαση μπορεί συμβεί κατ 'επανάληψη).

Η διάσπαση και ομαδοποίηση (γνωστή και ως Map /Reduce) επιτρέπει την παράλληλη χρήση της διαδικασίας εκτέλεσης των εργασιών κερδίζοντας σε επιδόσεις ή / και σε επεκτασιμότητα. Με δύο κόμβους στο δίκτυο μπορούμε να επιτύχουμε έως και 2 φορές την αύξηση των επιδόσεων, με 3 κόμβους - έως 3 φορές την απόδοση αύξηση, και ούτω καθεξής.

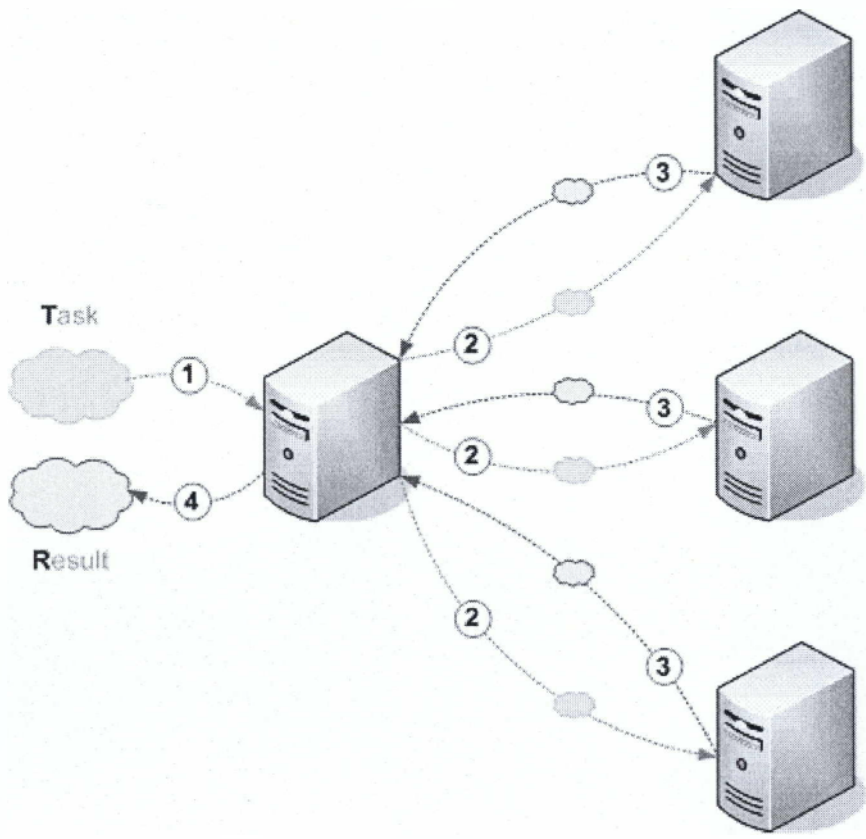
Στο GridGain μια εργασία (task) πλέγματος είναι υπεύθυνη για τη διάσπαση και την ομαδοποίηση με τη δουλειά (job) να κατανέμει της μονάδες εργασίας. Ουσιαστικά μια εργασία πλέγματος χωρίζεται σε δουλειές πλέγματος που εκτελούνται στους κόμβους. Το ακόλουθο διάγραμμα παρουσιάζει τη διάσπαση και την ομαδοποίηση:

1. Η εργασία κάνει αίτηση για εκτέλεση

2. Το έργο διασπάτε (Map) σε "δουλειές"

3. Επιστρέφεται το αποτέλεσμα της εκτέλεσης

4. Ομαδοποίηση (Reduce) των αποτελεσμάτων σε ένα αποτέλεσμα




**Εικόνα 14 [24]**

## 4.8 Πλεονεκτήματα που προσφέρει το GridGain

### Κόστος - Είναι δωρεάν

Το GridGain είναι δωρεάν είτε για ένα κόμβο ή για χίλιους. Όσο βρισκόμαστε εντός της άδειας LGPL δεν κοστίζει τίποτα η χρήση του με όποιο τρόπο θέλουμε.

### Πηγαίος κώδικας - Είναι Open Source

Διαθέτει ένα πλήρως σχολιασμένο πηγαίο κώδικα που είναι απαραίτητος για ένα προϊόν middleware. Όχι μόνο προσφέρετε ο πηγαίος κώδικα λόγο της LGPL αλλά το πιο σημαντικό είναι ότι αυξάνει την ευκολία και την παραγωγικότητα για τον προγραμματιστή έχοντας τη δυνατότητα και δει τον κώδικα και να εντοπίσει την υλοποίηση και την συμπεριφορά των διαφόρων χαρακτηριστικών που προσφέρει το framework αντί να προσφύγει σε χρονοβόρες κλήσεις υποστήριξης ή σε εικασίες.

### 4.8.1 Υποστήριξη - Enterprise Level

Το GridGain Systems παρέχει μια πλήρη γκάμα επιλογών για τους πελάτες που χρειάζονται βοήθεια στην επίλυση ζητημάτων με το προϊόν GridGain, από τους ίδιους τους μηχανικούς που το κατασκευάζουν

### 4.8.2 Java – Δημιουργήθηκε σε Java

Το GridGain έχει δημιουργηθεί σε Java. Αλλά αυτό που έχει σημασία είναι ότι δημιουργήθηκε για Java προγραμματιστές.

### 4.8.3 AOP – Καινοτόμος ενεργοποίηση του πλέγματος με την AOP

Ένα από τα μοναδικά χαρακτηριστικά του GridGain είναι η ενεργοποίηση του πλέγματος μέσω της AOP(Aspect Oriented Programming) που επιτρέπει σε μία εργασία να εκτελεστεί στο πλέγμα.

Με την AOP μας επιτρέπεται απλά να προσθέσουμε ένα annotation σε μια μέθοδο και την επόμενη φορά που θα καλεστεί η μέθοδος αυτή να εκτελεστεί στο πλέγμα. Έχουμε τον πλήρη έλεγχο για το πώς θα εκτελεστεί στο πλέγμα. Ένα από τα ενδιαφέροντα χαρακτηριστικά της προσέγγισης αυτής είναι ότι σε πολλές περιπτώσεις δεν χρειάζεται να έχουν τον πηγαίο κώδικα για το σύστημα που θέλουμε να εκτελεστεί στο πλέγμα - εφόσον γνωρίζουμε, την υπογραφή της μέθοδος που πρόκειται να εκτελεστεί. [24]

### 4.8.4 Απλοποιημένο – Εύκολο στη χρήση

Με το GridGain μπορούμε να δημιουργήσουμε μια εφαρμογή σε πλέγμα μέσα σε λίγα λεπτά.

### 4.8.5 Χαρακτηριστικά – Τα καλύτερα χαρακτηριστικά που διαθέτει το Grid Computing

Το μεγαλύτερο πλεονέκτημα του προϊόντος είναι ότι γίνεται συνδυασμός ενός απλού μοντέλου προγραμματισμού με τα στοιχεία της υπολογιστικής τεχνολογίας του πλέγματος. Παρέχει ένα πλήρες εύρος των στάνταρ και αρκετές προχωρημένες λειτουργίες όπως διαχείριση της τοπολογίας, failover και διαχείριση των συγκρούσεων μέσω των SPIs, έξυπνη και πλήρως προγραμματιζόμενα map / reduce (γνωστό και ως split/aggregate ή mcast-reduce, στη γλώσσα του MPI), pluggable deployment, σημεία ελέγχου, επικοινωνία, εντοπισμό , γεγονότα και εύρεση μέσω των SPIs, αυτόματη peer-to-peer deployment, κλπ. [24]

### 4.8.6 Ενσωμάτωση – Ολοκληρωμένη υποστήριξη από τον κατασκευαστή

GridGain έρχεται από τον κατασκευαστή με υποστήριξη για: [24]

- JBoss
- Spring
- Spring AOP
- JBoss AOP
- AspectJ
- JGroups
- Coherence
- GigaSpaces
- JXInsight
- Weblogic
- Websphere
- Mule

### 4.8.7 Agile – Δημιουργημένο φιλικά προς τον προγραμματιστή

Εδώ είναι μερικά από τα χαρακτηριστικά που εφαρμόζονται από το GridGain ώστε να είναι φιλικό προς τον προγραμματιστή, ιδανικό για ομάδες και προσανατολισμένο στη γρήγορη ανάπτυξη: [24]

- Ρυθμίσεις βασισμένες στο IoC μέσω του Spring
- Υποστήριξη τοπικών δοκίμων
- Μπορούμε να τρέξουμε πολλούς κόμβους σε ένα υπολογιστή ή σε μια JVM και να κάνουμε αποσφαλμάτωση τοπικά

- Προεπιλεγμένο peer-to-peer τοπολογία με αυτόματη γρήγορη επαναφόρτιση ώστε να εξαλείψει τη δαπανηρή διαδικασία της κατασκευής –απλά κάνουμε ξανά μεταγλώττιση τοπικά και εκτελούμε
- Λειτουργεί όπως μια τυπική εφαρμογή τοποθετημένη εσωτερικά μέσα σε ένα περιβάλλον όπως ο JBoss
- Ξεκάνει από οποιοδήποτε IDE (Eclipse, IDEA, NetBeans) σε δευτερόλεπτα
- Υποστηρίζει την κατασκευή μέσω του Ant
- Κατασκευασμένο με αυξημένες εσωτερικές πληροφορίες στον πηγαίο κώδικα σε όλες τις εξαιρέσεις (exceptions) και τους ισχυρισμούς (assertions)

# Κεφάλαιο 5

## Quicksort

# Quicksort

Η quicksort είναι ένας αλγόριθμος ταξινόμησης που χρησιμοποιείται περισσότερο από κάθε άλλο αλγόριθμο ταξινόμησης. Ο βασικός αλγόριθμος εφευρέθηκε το 1960 από τον C. A. R. Hoare, και έχει μελετηθεί από πολλούς ανθρώπους από την εποχή εκείνη. Η Quicksort είναι δημοφιλής, διότι δεν είναι δύσκολο να εφαρμοστεί, λειτουργεί καλά για μια ποικιλία από διαφορετικά είδη δεδομένων εισόδου, και καταναλώνει λιγότερους πόρους από οποιαδήποτε άλλη μέθοδο ταξινόμησης στις περισσότερες περιπτώσεις.

Το βασικό πλεονέκτημα αυτού του αλγορίθμου είναι ότι χρησιμοποιεί μόνο μια μικρή βοηθητική στοίβα, απαιτεί χρόνο ανάλογο με N log N κατά μέσο ορό για την ταξινόμηση N αντικείμενων, και έχει ένα πολύ σύντομο εσωτερικό βρόχο. Το μειονεκτήματα του είναι ότι δεν είναι σταθερός, χρειάζεται περίπου $N^2$ εργασίες στη χειρότερη περίπτωση, είναι εύθραυστος με την έννοια ότι ένα απλό λάθος στην εφαρμογή του μπορεί να περάσει απαρατήρητο και μπορεί να προκαλέσει προβλήματα.

Μια προσεκτικά ρυθμισμένη έκδοση της γρήγορης ταξινόμησης είναι πιθανό να τρέχει πολύ πιο γρήγορα στους περισσότερους υπολογιστές από οποιαδήποτε άλλη μέθοδο ταξινόμησης, καθώς ο αλγόριθμος αυτός χρησιμοποιείται ευρέως ως ένα εργαλείο ταξινόμησης σε βιβλιοθήκες και σε άλλες σοβαρές εφαρμογές ταξινόμησης. Η κύρια βιβλιοθήκη ταξινόμησης της Java είναι υλοποιημένη με τη quicksort. [26]

## 5.1 Ο βασικός αλγόριθμος

Η Quicksort είναι μια διαίρεί και βασίλευε, μέθοδος ταξινόμησης. Λειτουργεί με τον τεμαχισμό (partitioning) ενός πίνακα σε δύο μέρη, και την ταξινόμηση των μερών αυτών ανεξάρτητα. Η ακριβής θέση του διαμερίσματος εξαρτάται από την αρχική διάταξη των στοιχείων στο αρχείο εισόδου. Η κύρια λειτουργία της μεθόδου είναι η διαδικασία τεμαχισμού, η οποία αναδιατάσσει τον πινάκα για να δημιουργήσει τις ακόλουθες τρεις προϋποθέσεις: [26]

- Το στοιχειό a[i] είναι στη τελική του θέση μέσα στο πινάκα για κάποιο i.
- Κανένα από τα στοιχειά στις a[l], ..., a[i-1] είναι μεγαλύτερο από a[i].
- Κανένα από τα στοιχειά στις a[i+1], ..., a[r] είναι μικρότερο από a[i].

### 5.1.2 Quicksort

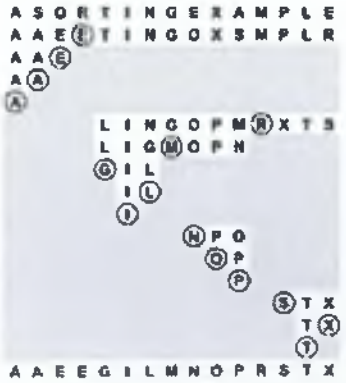
Αν ο πίνακας έχει ένα ή και λιγότερα στοιχεία, δεν κάνει τίποτα. Σε αντίθετη περίπτωση, ο πίνακας επεξεργάζεται από τη διαδικασία partition, η οποία θέτει το a[i] σε θέση για κάποιο i μεταξύ l και r και αναδιατάσσει τα άλλα στοιχεία, έτσι ώστε οι αναδρομικές κλήσεις τελειώσουν σωστά την ταξινόμηση.

```
static void quicksort(ITEM[] a, int l, int r)
 {
   if (r <= l) return;
   int i = partition(a, l, r);
   quicksort(a, l, i-1);
   quicksort(a, i+1, r);
 }
```

Όταν επιτύχουμε μια πλήρες ταξινόμηση με τη διαδικασία τεμαχισμού, εφαρμόζουμε αναδρομικά τη μέθοδο αυτή στα υπό-αρχεία. Επειδή η διαδικασία τεμαχισμού βάζει τουλάχιστον ένα στοιχείο στη θέση του, είναι μια επίσημη απόδειξη ότι η αναδρομική μέθοδος εκτελεί μια σωστή ταξινόμηση που δεν είναι δύσκολη να υλοποιηθεί. Η Quicksort είναι ένα αναδρομικό πρόγραμμα που υλοποιεί αυτή την ιδέα. [26]

#### 5.1.2.1 Quicksort παράδειγμα

Η Quicksort είναι μια επαναληπτική διαδικασία τεμαχισμού: Τεμαχίζουμε ένα αρχείο τοποθετώντας κάποιο στοιχείο (το στοιχείο διαχωρισμού), στη θέση του και αναδιοργανώνουμε την σειρά έτσι τα ώστε τα μικρότερα στοιχεία να είναι στα αριστερά του στοιχείου διαχωρισμού και τα μεγαλύτερα στοιχεία στα δεξιά. Στη συνέχεια, ταξινομούμε το αριστερό και δεξί μέρος του πίνακα, αναδρομικά. Κάθε γραμμή στην Εικόνα 15 απεικονίζει το αποτέλεσμα του τεμαχισμού εμφανίζοντας τα υπό-αρχεία χρησιμοποιώντας το κυκλωμένο στοιχείο. Το τελικό αποτέλεσμα είναι ένα πλήρως ταξινομημένο αρχείο.




Εικόνα 15 [26]

Χρησιμοποιούμε την ακόλουθη στρατηγική για την υλοποίηση του τεμαχισμού. Επιλέγουμε αυθαίρετα ένα a[r] να είναι το στοιχειό διαχωρισμού το οποίο θα πάει στην τελική του θέση. Στη συνέχεια, σαρώνουμε από το αριστερό άκρο του πίνακα, μέχρι να βρούμε κάποιο στοιχείο μεγαλύτερο από το στοιχείο διαχωρισμού, και σαρώνουμε από το δεξιό άκρο του πίνακα, μέχρι να βρούμε ένα στοιχείο μικρότερο από το στοιχείο διαχωρισμού. Τα δύο στοιχεία που σταμάτησαν την σάρωση είναι προφανώς εκτός θέσης στο τελικό τεμαχισμένο πινάκα, γι' αυτό και τα ανταλλάσουμε. Συνεχίζοντας με αυτόν τον τρόπο, διασφαλίζουμε ότι κανένα στοιχειό του πινάκα στα αριστερά του αριστερού ευρετηρίου δεν είναι μεγαλύτερο από το στοιχείο διαχωρισμού, και ότι κανένα στοιχειό του πινάκα στα δεξιά του δεξιού ευρετηρίου δεν είναι μικρότερο από το στοιχείο διαχωρισμού, όπως απεικονίζεται στη παρακάτω εικόνα.


**Εικόνα 16 [26]**

Όταν οι δείκτες της σάρωσης διασταυρωθούν, το μόνο που χρειάζεται για να ολοκληρωθεί η διαδικασία τεμαχισμού είναι να ανταλλάξουμε το a[r] με το αριστερότερο στοιχείο του δεξιού υπό-πινάκα (το στοιχείο υποδεικνύεται από το αριστερό ευρετήριο). [26]

### 5.1.3 Τεμαχισμός

Η μεταβλητή v έχει την τιμή του αρχείου διαχωρισμού a[r], και i και j είναι ο αριστερός και δεξιός δείκτης της σάρωσης, αντίστοιχα. Ο βρόγχος τεμαχισμού αυξάνει το i και μειώνει το j, όσο διατηρεί αναλλοίωτη την ιδιότητα ότι κανένα στοιχειό αριστερά του i δεν είναι μεγαλύτερο από το v και ότι κανένα στοιχειό δεξιά του j δεν είναι μικρότερο από το v. Όταν οι δείκτες συναντηθούν, έχουμε ολοκληρώσει τον τεμαχισμό ανταλλάσοντας το a[i] και το a[r], που τοποθετεί το v στο a[i], με κανένα στοιχειό δεξιά του v και κανένα μικρότερο στοιχειό στα αριστερά του. [26]

Ο βρόγχος τεμαχισμού υλοποιείτε σαν ένας ατέρμον βρόγχος που κάνει break όταν οι δείκτες διασταυρωθούν. Η συνθήκη j==l μας προστατεύει από την περίπτωση που τα στοιχειό διαχωρισμού είναι το μικρότερο στοιχειό στο αρχείο. [26]

```
static int partition(ITEM a[], int l, int r)
 { int i = l-1, j = r; ITEM v = a[r];
   for (;;)
    {
      while (less(a[++i], v)) ;
      while (less(v, a[--j])) if (j == l) break;
      if (i >= j) break;
      exch(a, i, j);
    }
   exch(a, i, r);
   return i;  }
```

# Κεφάλαιο 6

# Benchmarks

## 6.1 Μέτρηση επίδοσης (benchmarking)

Τα benchmarks είναι ειδικά σχεδιασμένα προγράμματα που χρησιμοποιούνται για τον έλεγχο της επίδοσης και της συμπεριφοράς των υπολογιστικών συστημάτων κάτω από συγκεκριμένες και ελεγχόμενες συνθήκες [27]. Η μέτρηση επίδοσης θεωρείται η διαδικασία της εκτέλεσης κάποιων Benchmarks σε ένα συγκεκριμένο σύστημα και περιλαμβάνει τα βήματα της εκτέλεσης, μέτρησης και αναφοράς των αποτελεσμάτων. Τα benchmarks χρησιμοποιούνται, τόσο από τους κατασκευαστές συστημάτων προκειμένου να σχεδιάζουν συστήματα τα οποία θα αποκρίνονται καλύτερα σε αυτά [2], όσο και από τους χρήστες ως ένα μέσο αξιολόγησης των συστημάτων τους. Προφανώς κανένα benchmark δεν γίνεται να δώσει ακριβείς προβλέψεις σε σχέση για παράδειγμα με την κανονική εκτέλεση ενός προγράμματος. Παρόλα αυτά οι πληροφορίες που μπορούν να αντληθούν είναι πολύ σημαντικές και χρήζουν μελέτης. Επιπλέον υπάρχουν κάποιες οδηγίες που πρέπει να ακολουθούνται όταν δημιουργούνται και εκτελούνται κάποια benchmarks και παρουσιάζονται παρακάτω [28]

- Τα benchmarks πρέπει να είναι «δίκαια». Δεν πρέπει να είναι μεροληπτικά ως προς μια συγκεκριμένη υλοποίηση ή ένα σύστημα.
- Τα benchmarks πρέπει να είναι εύχρηστα. Ένα εύχρηστο και διαθέσιμο benchmark γίνεται πιο εύκολα αποδεκτό από τους χρήστες.
- Οι προδιαγραφές (specifications) των benchmarks πρέπει να είναι γνωστές και εύκολα κατανοητές από τους χρήστες. Είναι σημαντικό να γίνεται εύκολα αντιληπτό τί είναι αυτό που μετράει το εκάστοτε benchmark δηλαδή ποιες είναι οι μετρικές (metrics) που φέρει.
- Το κόστος της διαχείρισης ενός benchmark, το οποίο περιλαμβάνει την εκτέλεση, την αποθήκευση αποτελεσμάτων και την εμφάνιση αυτών, να είναι όσο το δυνατό μικρό.

Τα benchmarks πρέπει να είναι «ρεαλιστικά». Τα benchmarks πρέπει να προσομοιώνουν όσο είναι δυνατό μια πραγματική κατάσταση που θα βρεθεί το σύστημα που μελετάται.

## 6.2 Μέτρηση επίδοσης στο Cloud Computing

Σε αυτή την ενότητα θα γίνει αναφορά στα προβλήματα και τις δυσκολίες που προκύπτουν από τη μέτρηση επίδοσης στο υπολογιστικό cloud. Ορίζουμε ως μέτρηση επίδοσης στο υπολογιστικό cloud (Cloud Benchmarking), τη χρήση των κατάλληλων benchmarks με σκοπό τον επαρκή και περιεκτικό χαρακτηρισμό της επίδοσης διαφόρων πτυχών της υποδομής του υπολογιστικού cloud [29].

Τα παραδοσιακά benchmarks δεν μπορούν απευθείας να εκτελεστούν στο υπολογιστικό cloud, επειδή υπάρχει ένα επιπλέον σημαντικό κόστος από τη χρήση των υπηρεσιών του cloud καθώς επίσης και λόγω των διαφόρων ζητημάτων που προκύπτουν από τη δομή και την αρχιτεκτονική της υποδομής του cloud computing.

Για παράδειγμα η επίδοση του υπολογιστικού πλέγματος μπορεί να επηρεαστεί από την επίδοση των μεμονωμένων στοιχείων που την αποτελούν. Από την επίδοση του υλικού, την επίδοση του δικτύου, την αξιοπιστία του ενδιάμεσου λογισμικού που χρησιμοποιείται και την επίδοση των βιβλιοθηκών και των υπηρεσιών που χρησιμοποιούνται.

Λόγω όλων αυτών των παραγόντων είναι επιτακτική η ανάγκη τα ίδια benchmarks να εκτελούνται πολλές φόρες προκειμένου τα αποτελέσματα να είναι όσο το δυνατόν πιο αξιόπιστα και έτσι η εξαγωγή των αποτελεσμάτων να έχει νόημα. Επίσης τα benchmarks πρέπει να ορίζονται και να περιγράφονται με τέτοιο τρόπο ώστε να μπορεί να εξασφαλιστεί η μεταφορά τους σε οποιοδήποτε ενδιάμεσο λογισμικό με ελάχιστο κόστος [28].

## 6.3 Κατηγορίες στη μέτρηση επίδοσης στο υπολογιστικό πλέγμα

Το τελευταίο καιρό αναπτύχθηκαν πολλά ερευνητικά προγράμματα τα οποία προσπαθούν να επικεντρωθούν σε διάφορες πτυχές της μέτρησης απόδοσης του cloud με την χρήση διαφόρων εργαλείων και τεχνικών. Θα αναφερθούν κάποιες από αυτές [28]

- Μέτρηση επίδοσης στην υποδομή του υπολογιστικού cloud.
  Σε αυτή την περίπτωση επικεντρωνόμαστε στο ανώτερο επίπεδο αρχιτεκτονικής της υποδομής του πλέγματος και χρησιμοποιούμε benchmarks τα οποία προσομοιώνουν workloads προκειμένου να ελέγξουν την συμπεριφορά του συστήματος σε ακραίες καταστάσεις.
- Μέτρηση επίδοσης στις υπηρεσίες του υπολογιστικού cloud.
  Προκειμένου να ελεγχθούν οι υπηρεσίες του cloud έχουν αναπτυχθεί benchmarks τα οποία μετρούν το χρόνο απόκρισης αυτών. Για παράδειγμα δημιουργούνται ακολουθίες ερωτήσεων στο σύστημα πληροφοριών (information system) και αξιολογούν το χρόνο απόκρισης.

## 6.4 Κατηγορίες των benchmarks

Όπως γίνεται εύκολα αντιληπτό ο σχεδιασμός των benchmarks πρέπει να γίνεται με τέτοιο τρόπο ώστε να μπορεί να εξασφαλιστεί ο έλεγχος της επίδοσης όλης της υποδομής του υπολογιστικού cloud. Από τις CPUs των υπολογιστών, μέχρι τις υπηρεσίες, το δίκτυο κτλ. Με βάση αυτά μπορεί να γίνει ο διαχωρισμός των benchmarks σε τρεις κατηγορίες. Κάθε κατηγορία παρέχει εν τέλει πληροφορίες για διαφορετικές πτυχές της υποδομής του υπολογιστικού πλέγματος [29].

- Micro benchmarks
  Σε αυτή την κατηγορία απομονώνουμε κάποια οντότητα του cloud και τη μετράμε σαν να ήταν τελείως αυτόνομη σε σχέση με ολόκληρη την υποδομή. Για παράδειγμα μετράμε την επίδοση των επεξεργαστών (πχ σε πράξεις κινητής υποδιαστολής ανά δευτερόλεπτο).

- Micro kernels
  Σε αυτή την κατηγορία μετράμε την επίδοση ενός συνόλου οντοτήτων που εντάσσονται στην υποδομή του cloud. Για παράδειγμα η μέτρηση της επίδοσης στην παράλληλη επικοινωνία των CPUs στα πλαίσια μια τοποθεσίας (grid site).
- Application kernels και Grid applications [20]
  Σε αυτή την περίπτωση μετράμε την επίδοση μιας οντότητας του cloud σε όσο το δυνατό περισσότερο ρεαλιστικές συνθήκες. Για παράδειγμα εκτελούμε ένα κομμάτι (demo) μιας πραγματικής εφαρμογής και μετράμε τα δευτερόλεπτα που θέλει για να εκτελεστεί.

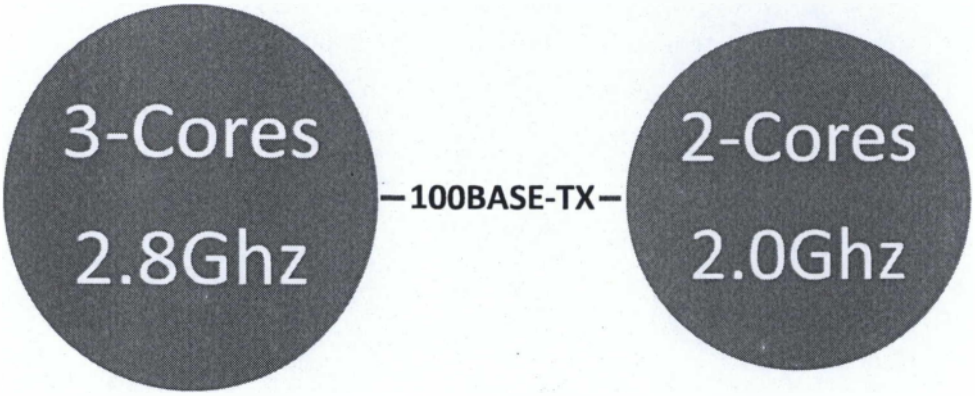
Στην παρούσα διπλωματική εργασία αναπτύχθηκαν και ενσωματώθηκαν στο σύστημα benchmarks που ανήκουν στην κατηγορία Application kernels και Grid applications.

## 6.5 Πειραματικά αποτελέσματα

Προκειμένου να ελέγξουμε την αποδοτικότητα των αλγορίθμων ταξινόμησης που δημιουργήσαμε για cloud και για πολυνηματικούς επεξεργαστές, αλλά και να επιβεβαιώσουμε τα χαρακτηριστικά της κάθε υλοποίησης, όπως επίσης να επιβεβαιώσουμε τα όσα αναλύθηκαν θεωρητικά στα παραπάνω κεφάλαια, υλοποιήσαμε ένα σύστημα δημιουργίας πινάκων και εκτελέσαμε ένα αριθμό πειραμάτων με βάση το μέγεθος του κάθε πίνακα. Τα αποτελέσματα των πειραμάτων αυτών μας επιτρέπουν να καταλήξουμε σε χρήσιμα συμπεράσματα, καθώς και να αξιολογήσουμε τα πλεονεκτήματα που προκύπτουν από τη χρήση των διαδικασιών που παρουσιάστηκαν σε ένα πραγματικό σύστημα.

Για την εκτέλεση των πειραμάτων χρησιμοποιήσαμε μια συστοιχία 2 υπολογιστών. Προκειμένου να εξασφαλίσουμε τη σωστή λειτουργία του συστήματος, ένας από τους κόμβους αυτούς είχε το ρόλο του Master-συντονιστή του συστήματος όσον αφορά τη λειτουργία διάσπαση και της επανένωσης αλλά και σαν υπολογιστικός κόμβος. Οι υπόλοιποι 2 κόμβοι χρησιμοποιούνται για τους υπολογισμούς που έχουμε θέση. Οι υπολογιστές που χρησιμοποιήσαμε αποτελούνται από ένα επεξεργαστή AMD PHENOM II X3 στα 2,8 GHz με 1,5Mb L2 Cache και ένα INTEL Core2 Duo στα 2 GHz με 4 Mb L2 Cache. Το κάθε σύστημα έχει από 2 Gb μνήμη Ram. Ως λειτουργικό σύστημα χρησιμοποιήσαμε το Linux και συγκεκριμένα την διανομή Ubuntu 10.04 x32 bit. Ακόμα αυξήσαμε το μέγεθος του Java Heap για τους reducers προκειμένου να μπορέσουν να διαχειριστούν το μεγάλο όγκο δεδομένων, χωρίς να υπάρχουν προβλήματα μνήμης.

Για τα τεστ που κάναμε χρησιμοποιήσαμε πίνακες που περιέχουν τρεις από τους βασικούς τύπους της java (int, long, double) και ένα πίνακα που περιέχει αντικείμενα τύπου BigInteger. Ο κάθε πίνακας ταξινομήθηκε με την χρήση της πολυνηματικής quicksort και της quicksort υλοποιημένης για cloud computing. Η συνολική δύναμη σε επεξεργαστική ισχύ του cloud συστήματος που χρησιμοποιήσαμε ήταν 5 πυρήνες όπως φαίνεται στην Εικόνα 17.



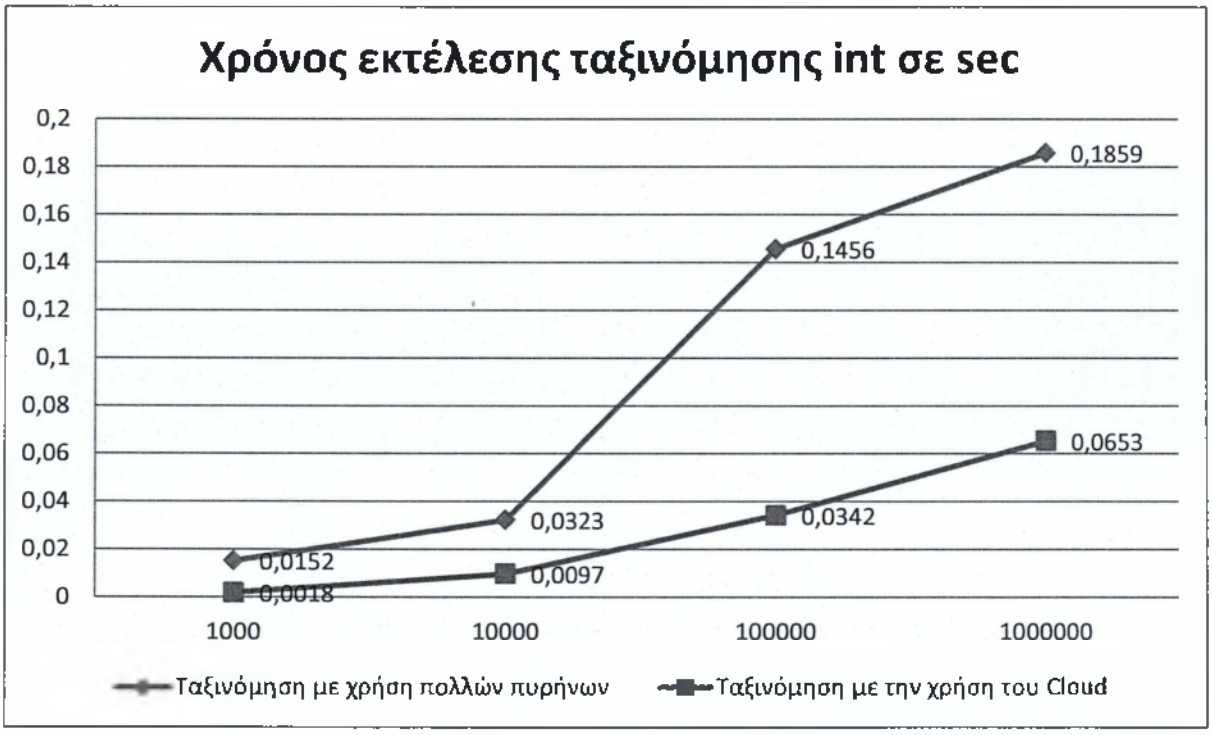

Εικόνα 17

Για τα τεστ δημιουργήσαμε 4 πίνακες μεγέθους 1000, 10000, 100000 και 1000000 στοιχείων. Κάθε πίνακας ταξινομήθηκε δέκα φορές και με βάση τον μέσο όρο βγήκαν τα αποτελέσματα που χρησιμοποιήθηκαν στην σύγκριση.

### 6.5.1 Ταξινόμηση πινάκα με στοιχεία τύπου Integer

| Multithreaded Quicksort | | | | |
|---|---|---|---|---|
| | **1000** | **10000** | **100000** | **1000000** |
| **1** | 0.009 | 0.016 | 0.094 | 0.19 |
| **2** | 0.02 | 0.04 | 0.056 | 0.22 |
| **3** | 0.022 | 0.047 | 0.13 | 0.169 |
| **4** | 0.009 | 0.047 | 0.051 | 0.215 |
| **5** | 0.026 | 0.035 | 0.099 | 0.156 |
| **6** | 0.014 | 0.041 | 0.111 | 0.192 |
| **7** | 0.01 | 0.021 | 0.116 | 0.168 |
| **8** | 0.01 | 0.046 | 0.65 | 0.198 |
| **9** | 0.026 | 0.014 | 0.072 | 0.19 |
| **10** | 0.006 | 0.016 | 0.077 | 0.161 |
| **Average** | | | | |
| | **1000** | **10000** | **100000** | **1000000** |
| | 0.0152 | 0.0323 | 0.1456 | 0.1859 |

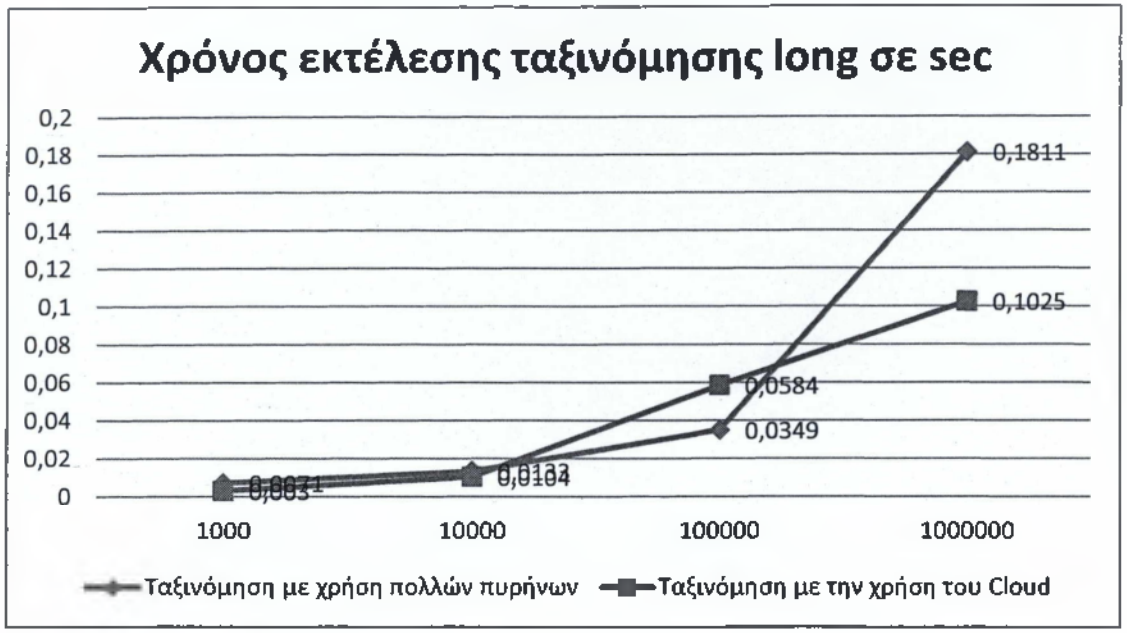
| Grid Computing | | | | |
|---|---|---|---|---|
| | 1000 | 10000 | 100000 | 1000000 |
| 1 | 0.002 | 0.003 | 0.02 | 0.025 |
| 2 | 0.001 | 0.007 | 0.024 | 0.033 |
| 3 | 0.001 | 0.008 | 0.027 | 0.044 |
| 4 | 0.001 | 0.009 | 0.03 | 0.053 |
| 5 | 3.00E-003 | 0.01 | 0.033 | 0.061 |
| 6 | 0.002 | 0.011 | 0.036 | 0.07 |
| 7 | 0.001 | 0.012 | 0.039 | 0.079 |
| 8 | 0.001 | 0.012 | 0.042 | 0.088 |
| 9 | 0.004 | 0.012 | 0.044 | 0.096 |
| 10 | 0.002 | 0.013 | 0.047 | 0.104 |
| Average | | | | |
| | 1000 | 10000 | 100000 | 1000000 |
| | 0.0018 | 0.0097 | 0.0342 | 0.0653 |



## 6.5.2 Ταξινόμηση πίνακα με στοιχεία τύπου Long

| Multithreaded Quicksort | | | | |
|---|---|---|---|---|
| | 1000 | 10000 | 100000 | 1000000 |
| 1 | 0.019 | 0.027 | 0.137 | 0.327 |
| 2 | 0.002 | 0.012 | 0.04 | 0.184 |
| 3 | 0.002 | 0.006 | 0.02 | 0.193 |
| 4 | 0.002 | 0.005 | 0.014 | 0.141 |
| 5 | 0.003 | 0.005 | 0.017 | 0.197 |
| 6 | 0.001 | 0.047 | 0.016 | 0.178 |
| 7 | 0.005 | 0.007 | 0.028 | 0.136 |
| 8 | 0.023 | 0.009 | 0.019 | 0.141 |
| 9 | 0.008 | 0.008 | 0.026 | 0.15 |
| 10 | 0.006 | 0.007 | 0.032 | 0.164 |
| Μέσος Όρος | | | | |
| | 1000 | 10000 | 100000 | 1000000 |
| | 0.0071 | 0.0133 | 0.0349 | 0.1811 |

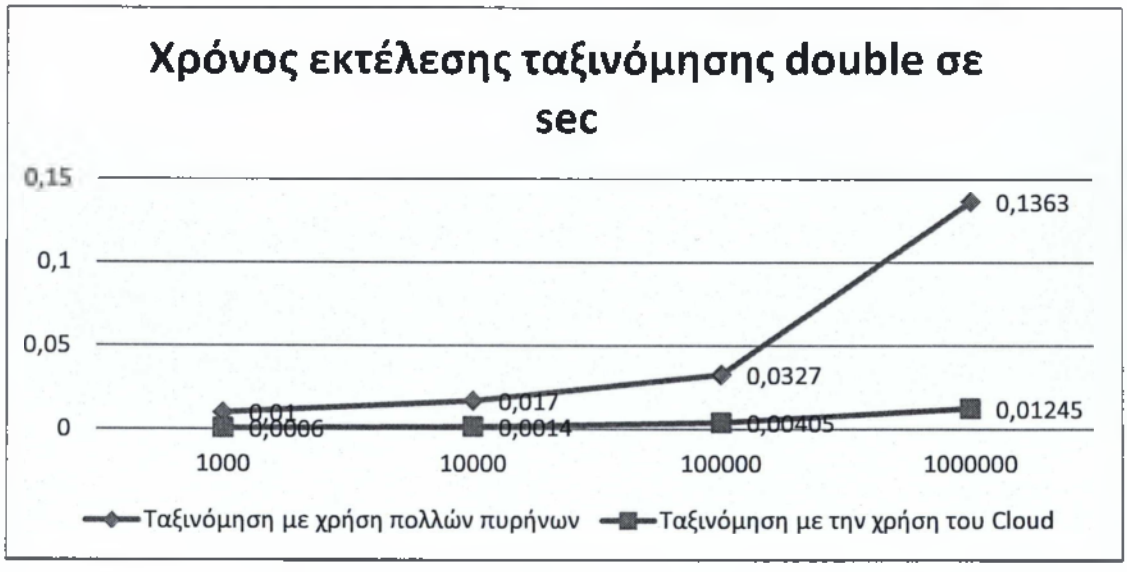
| Grid Computing | | | | |
|---|---|---|---|---|
| | 1000 | 10000 | 100000 | 1000000 |
| 1 | 0.002 | 0.005 | 0.039 | 0.035 |
| 2 | 0.002 | 0.009 | 0.046 | 0.05 |
| 3 | 0.002 | 0.009 | 0.047 | 0.065 |
| 4 | 0.003 | 0.01 | 0.052 | 0.08 |
| 5 | 0.003 | 0.01 | 0.056 | 0.095 |
| 6 | 0.003 | 0.011 | 0.059 | 0.11 |
| 7 | 0.003 | 0.012 | 0.064 | 0.125 |
| 8 | 0.004 | 0.012 | 0.069 | 0.14 |
| 9 | 0.004 | 0.013 | 0.074 | 0.155 |
| 10 | 0.004 | 0.013 | 0.078 | 0.17 |
| Μέσος Όρος | | | | |
| | 1000 | 10000 | 100000 | 1000000 |
| | 0.003 | 0.0104 | 0.0584 | 0.1025 |




### 6.5.3 Ταξινόμηση πίνακα με στοιχεία τύπου Double

| Multithreaded Quicksort | | | | |
|---|---|---|---|---|
| | 1000 | 10000 | 100000 | 1000000 |
| 1 | 0.006 | 0.063 | 0.046 | 0.223 |
| 2 | 0.004 | 0.022 | 0.053 | 0.113 |
| 3 | 0.003 | 0.005 | 0.04 | 0.098 |
| 4 | 0.011 | 0.009 | 0.07 | 0.123 |
| 5 | 0.002 | 0.014 | 0.049 | 0.152 |
| 6 | 0.003 | 0.032 | 0.016 | 0.117 |
| 7 | 0.004 | 0.011 | 0.014 | 0.144 |
| 8 | 0.041 | 0.004 | 0.016 | 0.116 |
| 9 | 0.011 | 0.005 | 0.012 | 0.139 |
| 10 | 0.015 | 0.005 | 0.011 | 0.138 |
| Μέσος Όρος | | | | |
| | 1000 | 10000 | 100000 | 1000000 |
| | 0.01 | 0.017 | 0.0327 | 0.1363 |

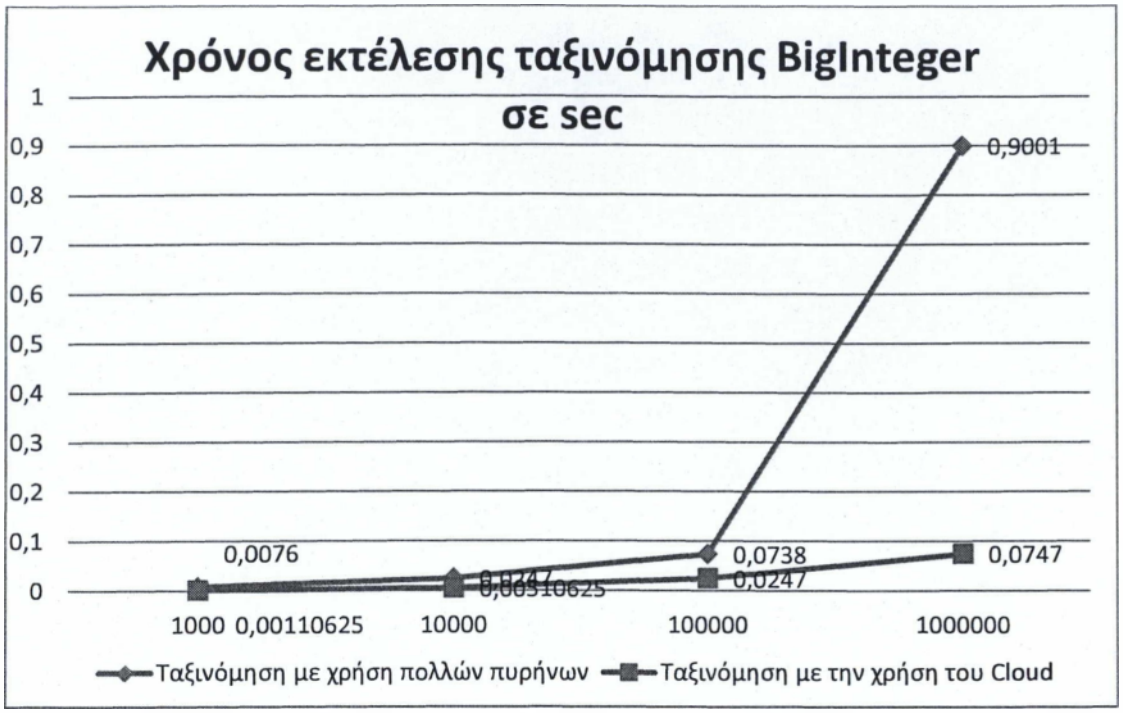
| Grid Computing | | | | |
|---|---|---|---|---|
| | 1000 | 10000 | 100000 | 1000000 |
| 1 | 0.001 | 0.005 | 0.01 | 0.029 |
| 2 | 0 | 0.004 | 0.001 | 0.0115 |
| 3 | 0.001 | 0.001 | 0.003 | 0.011 |
| 4 | 0 | 0 | 0.0045 | 0.011 |
| 5 | 0.001 | 0.001 | 0.004 | 0.011 |
| 6 | 0.001 | 0 | 0.002 | 0.01 |
| 7 | 0.001 | 0.001 | 0.003 | 0.011 |
| 8 | 0 | 0.001 | 0.005 | 0.01 |
| 9 | 0 | 0 | 0.005 | 0.01 |
| 10 | 0.001 | 0.001 | 0.003 | 0.01 |
| Μέσος Όρος | | | | |
| | 1000 | 10000 | 100000 | 1000000 |
| | 0.0006 | 0.0014 | 0.00405 | 0.01245 |



### 6.5.4 Ταξινόμηση πίνακα με στοιχεία τύπου BigInteger

| Multithreaded Quicksort | | | | |
|---|---|---|---|---|
| | 1000 | 10000 | 100000 | 1000000 |
| 1 | 0.022 | 0.038 | 0.085 | 0.821 |
| 2 | 0.003 | 0.04 | 0.043 | 1.244 |
| 3 | 0.001 | 0.033 | 0.054 | 0.759 |
| 4 | 0.004 | 0.015 | 0.108 | 0.911 |
| 5 | 0.003 | 0.039 | 0.055 | 1.003 |
| 6 | 0.003 | 0.018 | 0.108 | 0.809 |
| 7 | 0.003 | 0.023 | 0.051 | 0.685 |
| 8 | 0.005 | 0.03 | 0.095 | 0.573 |
| 9 | 0.003 | 0.005 | 0.074 | 0.926 |
| 10 | 0.029 | 0.006 | 0.065 | 1.27 |
| Μέσος Όρος | | | | |
| | 1000 | 10000 | 100000 | 1000000 |
| | 0.0076 | 0.0247 | 0.0738 | 0.9001 |

| Grid Computing | | | | |
|---|---|---|---|---|
| | 1000 | 10000 | 100000 | 1000000 |
| 1 | 0.0010625 | 0.003 | 0.028 | 0.229 |
| 2 | 0.001 | 0.006 | 0.027 | 0.057 |
| 3 | 0.001 | 0.003 | 0.031 | 0.05 |
| 4 | 0.002 | 0.003 | 0.024 | 0.059 |
| 5 | 0.001 | 0.0030625 | 0.024 | 0.05 |
| 6 | 0.001 | 0.003 | 0.017 | 0.064 |
| 7 | 0.001 | 0.008 | 0.03 | 0.059 |
| 8 | 0.001 | 0.004 | 0.025 | 0.047 |
| 9 | 0.001 | 0.007 | 0.017 | 0.073 |
| 10 | 0.001 | 0.011 | 0.024 | 0.059 |
| Μέσος Όρος | | | | |
| | 1000 | 10000 | 100000 | 1000000 |
| | 0.00110625 | 0.00510625 | 0.0247 | 0.0747 |




## 6.6 Συμπεράσματα

Όπως βλέπουμε και στα διαγράμματα η quicksort υλοποιημένη σε cloud computing έδωσε πολύ καλύτερους χρόνους ταξινόμησης από την πολυνυματική quicksort. Από τα παρακάτω διαγράμματα φαίνεται ότι όταν το μέγεθος του πίνακα είναι μικρό ο χρόνος ταξινόμησης είναι σχετικά ίδιος. Όσο αυξάνουμε το μέγεθος του πίνακα η διάφορα των χρόνων γίνεται μεγαλύτερη. Για δέκα φορές μεγαλύτερο πίνακα, έχουμε μια αύξηση 184% ή και περισσότερο στο χρόνο ταξινόμησης. Αυτό οφείλεται κυρίως στους περιορισμούς της μνήμης που διαθέτουν τα συστήματα που κάναμε τις δοκιμές. Όπως αναφέραμε στο προηγούμενο κεφάλαιο η quicksort είναι ένας αναδρομικός αλγόριθμος για να εκτελεστή η ταξινόμηση σε μεγάλες ποσότητες δεδομένων χρειάζεται αρκετή μνήμη. Αυτό είναι και ο λόγος που όταν εκτελούμε ταξινόμηση σε ένα μεγάλο αριθμό δεδομένων τα προγράμματα που δημιουργούμε σταματάμε την λειτουργία τους γιατί δεν έχουν την κατάλληλη ποσότητα μνήμης. Έτσι σύμφωνα με τα διαγράμματα βλέπουμε και την δύναμη του cloud computing που ξεπερνά αυτούς τους περιορισμούς έχοντας θεωρητικά άπειρη μνήμη ανάλογα πως έχει διαμορφωθεί το cloud σύστημα.

# Κεφάλαιο 7

## Επίλογος

# Επίλογος

Τα τελευταία χρόνια, τα δεδομένα που είναι διαθέσιμα στο διαδίκτυο αυξάνονται με εκρηκτικούς ρυθμούς, ενώ ταυτόχρονα η συμμετοχή των απλών χρηστών στη δημιουργία νέων δεδομένων, μέσω των social networks για παράδειγμα, έχει ως αποτέλεσμα τη συνεχή ανανέωση του περιεχομένου του διαδικτύου. Γι' αυτό το λόγο είναι πλέον αναγκαία η χρήση του cloud computing.

Το cloud computing, όπως συζητήσαμε στα προηγούμενα κεφάλαια, έρχεται να αλλάξει το τρόπο που αντιμετωπίζουμε τα κόστη και την παραγωγική διαδικασία γενικότερα.

Οι άνθρωποι της διοίκησης των επιχειρήσεων και αυτοί της οικονομικής επιστήμης, διαθέτουν πλέον επαρκή εμπειρία, μετά την υπερμεγέθη ανάπτυξη του κλάδου της πληροφορικής και το πλήγμα από την εκτόνωση της «Φούσκας» του Dot Com στις ΗΠΑ, ώστε στη σημερινή εποχή, να συζητούν με ιδιαίτερη προσοχή τη σχέση κόστους/οφέλους που μπορεί να προκύψει από κάθε επένδυση σε μια νέα καινοτομία.

Η πλαστότητα και η δυνατότητα συνδυαστικών παροχών που προσφέρονται μέσω της πλατφόρμας cloud, μας δίνει ένα ισχυρό εργαλείο να μεταφέρουμε σταθερά κόστη στην κατηγορία των μεταβλητών και έτσι να οδηγηθούμε στη βελτιστοποίηση των χρηματοοικονομικών ροών του οργανισμού.

Επιπρόσθετα, η χρήση διαφορετικών λύσεων cloud computing (IaSS, PaSS, SaSS), μας επιτρέπει να διαμορφώσουμε με μεγαλύτερη ευελιξία τη διαχείριση και διάχυση των διαθεσίμων πόρων για την απόκτηση παγίου κεφαλαίου (με την μεταφορά των Ownership Costs σε Operational costs). Έτσι η εξέλιξη του κόστους συνδέεται άμεσα με την αξία που λαμβάνει ο οργανισμός, επεκτείνοντας στην πραγματικότητα το outsourcing, που εφαρμόζεται από διάφορες εταιρίες

Η Amazon, η Google, και η Microsoft είναι από τους πιο σημαντικούς παρόχους υπηρεσιών cloud. Όπως όμως συμβαίνει και με το λογισμικό, έτσι και στην υπηρεσία cloud υπάρχουν εφαρμογές Open Source, οι οποίες βασίζονται σε συγκεκριμένη πλατφόρμα, και διευκολύνουν τον χρήστη να προσαρμόσει τις ανάγκες του στα μέτρα του. Η open source πλατφόρμα έχει την ευελιξία ότι μπορεί να προσαρμοστεί από τον χρήστη τον ίδιο.

# Amazon S3 (Simple Storage Service)

Η υπηρεσία Amazon S3 είναι μια online διαδικτυακή υπηρεσία αποθήκευσης δεδομένων που προσφέρεται από την Amazon Web Services . Το Amazon S3 παρέχει αποθήκευση δεδομένων μέσα από μια απλή διαδικτυακή διεπαφή (interface). Η Amazon προώθησε στις Ηνωμένες πολιτείες το Μάρτιο του 2006, και στην Ευρώπη το Νοέμβριο του 2007. Ξεκίνησε χρεώνοντας του χρήστες ανάλογα με τη χωρητικότητα που αποθήκευαν αλλά και ανάλογα με το εύρος ζώνης που χρησιμοποιούσαν για κατέβασμα αλλά και για ανέβασμα πληροφορίας. Αυτή τη στιγμή το S3 δίνει κάθε μήνα 99,9% ποσοστό εγγύησης για την υπηρεσία που διαθέτει. [30]

# Google App Engine

Η Google App Engine χρησιμοποιεί επίσης το ακρωνύμιο GAE και είναι μια πλατφόρμα για ανάπτυξη και φιλοξενία διαδικτυακών εφαρμογών σε κέντρα δεδομένων που τα διαχειρίζεται η Google. Είναι μια τεχνολογία υπολογιστικού νέφους (cloud computing ). Εξομοιώνει εφαρμογές ανάμεσα σε πολλαπλούς διακομιστές και κέντρα δεδομένων. Άλλες εφαρμογές υπολογιστικού νέφους περιλαμβάνουν παροχές όπως τις Amazon Web Services & Microsoft's Azure Services Platform . Η διάφορα με του Google App Engine είναι ότι το AWS είναι Iaas ενώ το App Engine είναι Paas. Η Google App Engine είναι δωρεάν μέχρι ενός συγκεκριμένου σημείου χρησιμοποιούμενων πηγών. Αυτή τη στιγμή οι γλώσσες προγραμματισμού που μπορούν να υποστηριχτούν είναι Java, Python αλλά και προεκτάσεις τους όπως άλλες JVM γλώσσες όπως Groovy, JRuby, Scala, Clojure,Jython. [31]

Κάποιες σημαντικές διαφορές συγκρίνοντας την με άλλες κλιμακούμενες υπηρεσίες φιλοξενίας όπως η Amazon EC2, είναι ότι η App Engine παρέχει περισσότερες υποδομές για να γίνει εύκολο να δημιουργηθούν κλιμακούμενες εφαρμογές, οι οποίες βέβαια μπορούν να εκτελέσουν περιορισμένου εύρους εφαρμογές που σχεδιαστήκαν για αυτές τις υποδομές. [31]

# Windows Azure

Η υπηρεσία Windows Azure Platform είναι μια πλατφόρμα υπολογιστικού νέφους που προσφέρεται από την Microsoft, και επιτρέπει στους χρήστες της να αναπτύσσουν εφαρμογές αλλά και δεδομένα μέσα στο cloud. Έτσι η Windows Azure Platform ταξινομείται ως Paas. Η πλατφόρμα αποτελείται από διάφορες υπηρεσίες κατά παραγγελία που φιλοξενούνται σε βάσεις δεδομένων της Microsoft. [32]

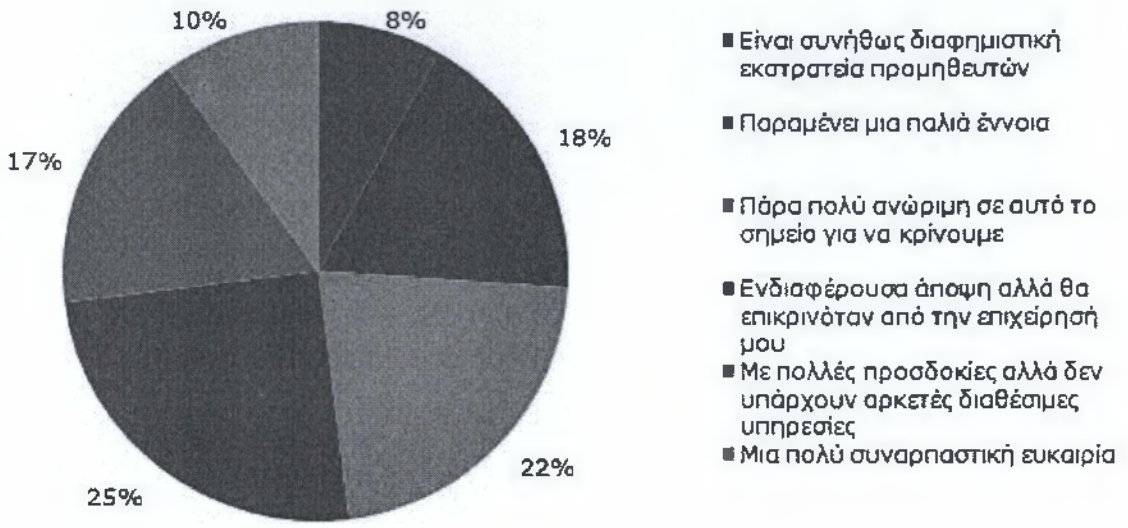
# Προβλέψεις για το cloud

Τριπλάσιες αναμένεται να είναι διεθνώς οι δαπάνες για υπηρεσίες cloud computing μέχρι το 2012, χρόνια κατά την οποία η αξία της συγκεκριμένης αγοράς θα φτάσει τα $42 δις, σύμφωνα με του αναλυτές της IDC. Η IDC υποστηρίζει πως στη σημερινή εποχή που ο περιορισμός του κόστους κυριαρχεί, το cloud computing δεν είναι απλώς μόδα αλλά μια κυρίαρχη τάση. [33]

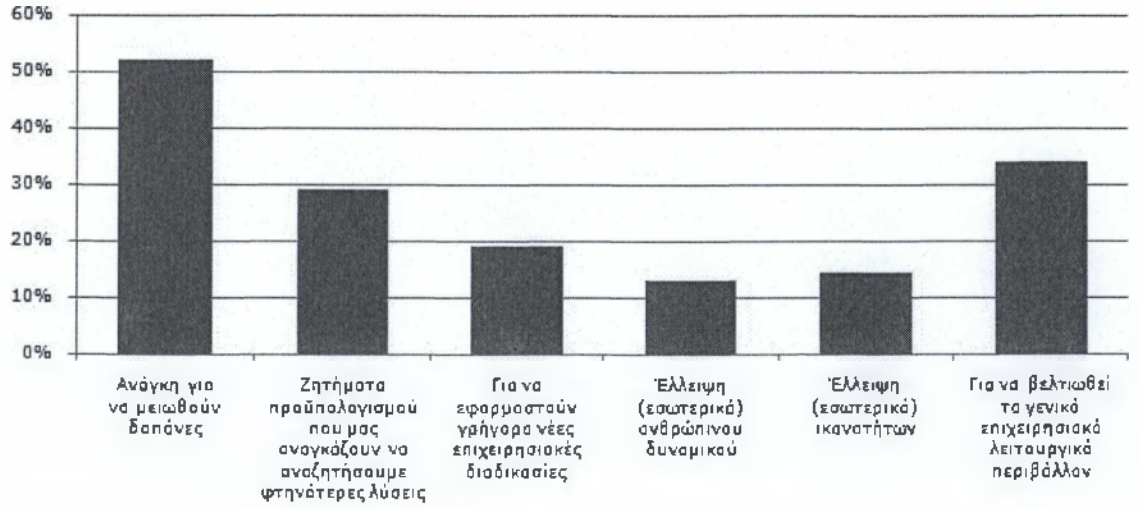
Σύμφωνα με τα στοιχεία πρόσφατης έρευνας που πραγματοποίησε η διεθνής εταιρία με την συμμετοχή 696 CIO' s και υψηλόβαθμων στελεχών από το χώρο του τμήματος πληροφορικής, το 11% των ερωτηθέντων κάνει ήδη χρήση λύσεων cloud computing. Το 41% δηλώνει έτοιμο να προχωρήσει στην υιοθέτηση τέτοιου είδους υπηρεσιών και αρκετοί έχουν ακολουθήσει πιλοτικές εφαρμογές. Από την άλλη πλευρά το 17% των ερωτηθέντων επισημαίνει, πως δεν υπάρχουν αρκετές εφαρμογές προς χρήση παρόλο που το cloud computing είναι πολλά υποσχόμενο. [33]

Σύμφωνα με τον Dr. Patrick Chan, IDC's Chief Technology Advisor for Emerging Technologies in Asia/ Pacific, το μέλλον του Dr. Patrick Chan, IDC's Chief Technology Advisor for Emerging Technologies in Asia/ Pacific φαντάζει λαμπρό. Σε τρία χρόνια από τώρα, οπότε θα έχει

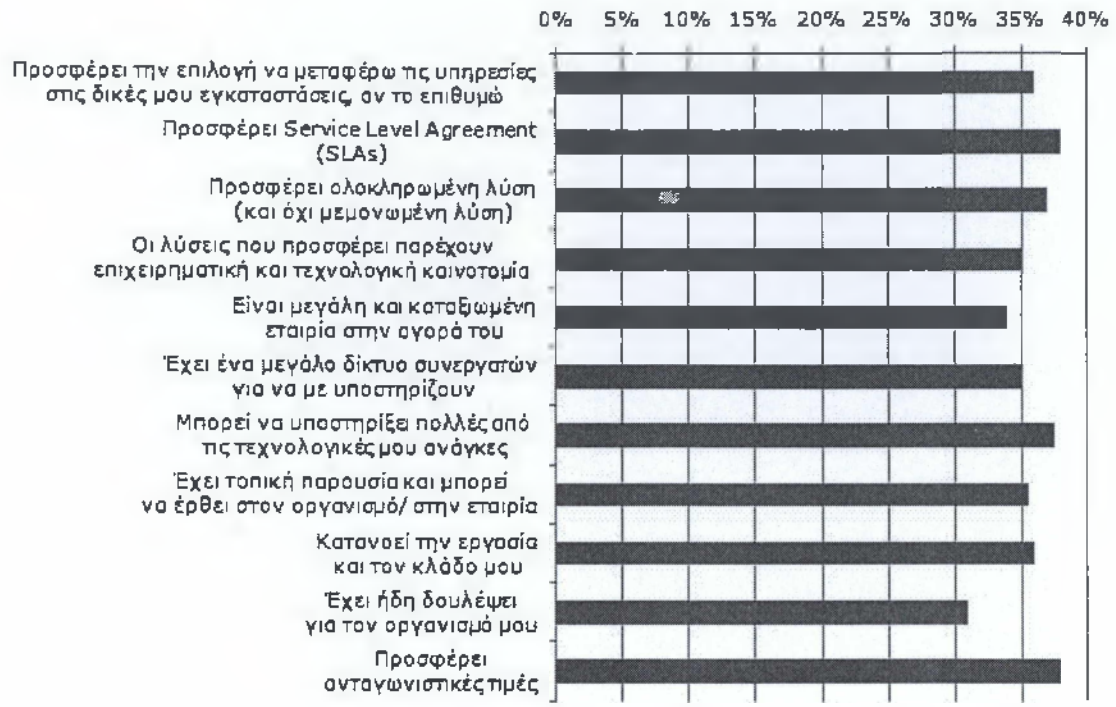
επεκταθεί περαιτέρω η συγκεκριμένη τάση, οι μεγάλοι κατασκευαστές θα πρέπει να έχουν ήδη πάρει θέση με τις κατάλληλες εφαρμογές, αν θέλουν να διεκδικήσουν ηγετική θέση σε μια ανερχόμενη αγορά. Μερικοί μεγάλοι κατασκευαστές έχουν ήδη τοποθετηθεί σωστά στη συγκεκριμένη αγορά. [33]




Εικόνα 18 Ποιά είναι η άποψή σας για την τρέχουσα κατάσταση του Cloud Computing. [33]




Εικόνα 19 Τι οδήγησε τον οργανισμό μας στην επιλογή των υπηρεσιών του Cloud Computing; [33]




Εικόνα 20 Πόσο σημαντικό είναι για τον οργανισμό μας, ένας προμηθευτής Cloud Computing υπηρεσιών να έχει τα παρακάτω χαρακτηριστικά; [33]

# Βιβλιογραφία


[1] Oracle Corporation, Ιανουάριος 2009 . [Ηλεκτρονικό]. Available: http://wikis.sun.com/display/GridEngine/What+Is+Grid+Computing.

[2] IBM, 2004 . [Ηλεκτρονικό]. Available: http://www.ibm.com/developerworks/rational/library/4124.html.

[3] R. Clark, Απρίλιος 2010. [Ηλεκτρονικό]. Available: http://www.zdnet.com/blog/gardner/private-cloud-models-moving-beyond-static-grid-computing-addiction/3562.

[4] Sun Microsystems, Inc., 2009. [Ηλεκτρονικό]. Available: https://www.sun.com/offers/details/CloudComputing.xml.

[5] C. K. Ian Foster, The Grid : Blueprint for a New Computing Infrastructure, 2η Έκδοση επιμ., Morgan Kaufmann, 2004.

[6] R. Buyya, Απρίλιος 2002. [Ηλεκτρονικό]. Available: www.buyya.com/thesis/thesis.pdf.

[7] Y. A. D. a. E. G.-S. Miguel L. Bote-Lorenzo, Φεβρουάριος 2004. [Ηλεκτρονικό]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.9439&rep=rep1&type=pdf.

[8] L. R.-M. J. C. M. L. Luis M. Vaquero, Ιανουάριος 2009. [Ηλεκτρονικό]. Available: http://ccr.sigcomm.org/drupal/files/p50-v39n1l-vaqueroA.pdf.

[9] C. S. Y. S. V. J. B. I. B. Rajkumar Buyya, Ιούνιος 2009. [Ηλεκτρονικό]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.144.8397&rep=rep1&type=pdf.

[10] D. C. Wyld, Νοέμβριος 2009. [Ηλεκτρονικό]. Available: http://airccse.org/journal/ijdms/1011s1.pdf.

[11] Google, [Ηλεκτρονικό]. Available: http://www.google.com/trends?q=grid+computing,+cloud+computing.

[12] M. Fawzy. [Ηλεκτρονικό]. Available: http://www.mohamedfawzy.com/static/cloudComputingLayers.jpg.

[13] E. B. R. F. J. J. K. P. N. N. T. N. R. R. A. S. S. a. R. S. Brian F. Cooper, 2009. [Ηλεκτρονικό]. Available: http://sites.computer.org/debull/A09mar/cooper1.pdf.

[14] P. C. Heckel, Απρίλιος 2010. [Ηλεκτρονικό]. Available: www.philippheckel.com/files/hybrid-

cloud-paper.pdf.

[15] Oracle, Μαίος 2010. [Ηλεκτρονικό]. Available: http://www.oracle.com/dm/offers/fy10/oracle_cloud_computing_final_new.pdf.

[16] VMware, [Ηλεκτρονικό]. Available: http://www.vmware.com/solutions/cloud-computing/.

[17] E. M. J. M. T. H. S. B. M. A. Neil MacDonald. [Ηλεκτρονικό]. Available: http://docs.google.com/viewer?a=v&q=cache:o8RSyANvd18J:citeseerx.ist.psu.edu/viewdoc/download%3Fdoi%3D10.1.1.8.5728%26rep%3Drep1%26type%3Dpdf+Course+notes+for+MPI,+Edinburgh+Parallel+Computing+Center&hl=en&pid=bl&srcid=ADGEESgGYgw-ouF7GYUjEOCJD7Log7sBrmLr.

[18] S. O. S. H.-L. D. W. J. D. Marc Snir, MPI: The Complete Reference, The MIT Press, 1996.

[19] Wikipedia, [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Message_Passing_Interface.

[20] Blaise Barney, Lawrence Livermore National Laboratory, [Ηλεκτρονικό]. Available: https://computing.llnl.gov/tutorials/mpi/.

[21] S. G. Jeffrey Dean, 2004. [Ηλεκτρονικό]. Available: http://labs.google.com/papers/mapreduce-osdi04.pdf.

[22] S. G. Jeffrey Dean, 2008. [Ηλεκτρονικό]. Available: http://portal.acm.org/citation.cfm?id=1327452.1327492&coll=GUIDE&dl=&idx=J79&part=magazine&WantType=Magazines&title=Communications%20of%20the%20ACM.

[23] Apache, [Ηλεκτρονικό]. Available: http://hadoop.apache.org/.

[24] GridGain, [Ηλεκτρονικό]. Available: http://www.gridgain.com/.

[25] Dynamic Agent Computations, [Ηλεκτρονικό]. Available: http://www.dacframe.org/trac/dac/.

[26] R. Sedgewick, Algorithms in Java: Parts 1-4, 3η Έκδοση επιμ., Addison Wesley, 2002.

[27] T. Calzarossa MC, «Benchmarking.Proceedings of Performance,» 2002.

[28] M. D. Dikaiakos, Grid benchmarking: Vision, Challenges, and Current status, 2007.

[29] I. K. C. Foster, The Grid: Blueprint for a New Computing Infrastructure, 2004.

[30] «Amazon Simple Storage Service,» [Ηλεκτρονικό]. Available: http://aws.amazon.com/s3/.

[31] «Google App Engine,» [Ηλεκτρονικό]. Available: http://code.google.com/appengine/.

[32] «Windows Azure,» [Ηλεκτρονικό]. Available: http://www.windowsazure.com/en-us/.

[33] «An IDC Update,» [Ηλεκτρονικό]. Available: http://www.slideshare.net/JorFigOr/cloud-computing-2010-an-idc-update. [Πρόσβαση 2010].

# Ορολογία

# Α

Αδρανές Idle

Αρχιτεκτονική υπολογιστικών πλεγμάτων Grid-architecture

# Δ

Διεπαφές Interfaces

Δομικό Fabric

# Ε

Εικονική μηχανή Virtual machine

Εξυπηρετητές Servers

Εργασίες-πλέγματος Grid jobs

Εύρος ζώνης Bandwidth

# Κ

Κατανεμημένα συστήματα Distributed computing

# Μ

Μεσίτες Brokers

Μποτιλιάρισμα Bottleneck

# Π

Πλέγμα Grid

Πλέγμα δεδομένων Data grid

Πόροι Resources

# Σ

Στοιχεία του πλέγματος Grid components

Συλλογικό στρώμα Collective layer

Συν-κατανομής Co-allocation

# Υ

Υλικό Hardware

Υπηρεσία ανάθεσης εργασιών Job submission service

Υπο-εργασίες Subjobs

# Παράρτημα

**GridSortGui.java**

```
package gr.teikal;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.event.KeyEvent;
import java.util.Random;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;

import org.gridgain.grid.Grid;
import org.gridgain.grid.GridException;
import org.gridgain.grid.GridFactory;

public class GridSortGui extends JFrame
{

        private static final long serialVersionUID = 1L;
        private JPanel jContentPane = null;
        // Κουμπί για τη δημιουργία του πίνακα
        private JButton initializing = null;

        // Εμφανίζει το χρόνο δημιουργίας του πίνακα
        private JLabel initializingTime = null;

        // Κουμπί για να εκτελεστή ο πίνακας στο πλέγμα
        private JButton gridSort = null;

        // Κουμπί για να εκτελέσουμε την quicksort
        private JButton quickSort = null;

        // Ετικέτα που εμφανίζει το χρόνο εκτέλεσης της quicksort
        private JLabel quickSortTime = null;

        // Σταθερά που ορίζει το μέγεθος του πίνακα
        public static final int NUM = 1000000;

        // Δημιουργία νέου πίνακα
        public int[] test = new int[NUM];
```

```
// Μεταβλητή που μετρά το χρόνο που χρειάστηκε να
// ταξινομηθεί ο πίνακας στο πλέγμα
public static double time = 0.0;

/**
 *Η κεντρική μέθοδος που δημιουργεί ένα νεο Thread
 */
public static void main(String[] args)
{

        SwingUtilities.invokeLater(new Runnable()
        {
                public void run()
                {
                        GridSortGui thisClass = new GridSortGui();
                        thisClass.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                        thisClass.setVisible(true);
                }
        });
}

/**
 * O default constructor
 */
public GridSortGui()
{
        super();
        initialize();
}

/**
 * Αυτή η μέθοδος αρχικοποιεί το γραφικό περιβάλλον
 *
 * @return void
 */
private void initialize()
{
        try
        {

                // Ορίζουμε ως θέμα γραφικού περιβάλλοντος
                // το θέμα που έχει το εκάστου
                // λειτουργικό σύστημα
                UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

        } catch (ClassNotFoundException e)
        {
                // TODO Auto-generated catch block
                e.printStackTrace();
        } catch (InstantiationException e)
        {
                                // TODO Auto-generated catch block
```

```
                e.printStackTrace();
            } catch (IllegalAccessException e)
            {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (UnsupportedLookAndFeelException e)
            {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            this.setContentPane(getJContentPane());

            // Δίνουμε ένα τίτλο στο παράθυρο
            this.setTitle("GridSort");

            this.setBounds(new Rectangle(0, 0, 574, 300));

            // Παίρνουμε το μέγεθος της οθόνης
            Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();

            // Υπολογίζουμε που είναι το κέντρο της οθόνης
            int w = this.getSize().width;
            int h = this.getSize().height;
            int x = (dim.width - w) / 2;
            int y = (dim.height - h) / 2;

            // Μετακινούμε το παράθυρο
            this.setLocation(x, y);
        }

        /**
         * Αυτή η μέθοδος δημιουργεί ένα νέο πάνελ.
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJContentPane()
        {
            if (jContentPane == null)
            {
                jContentPane = new JPanel();
                jContentPane.setLayout(null);
                // θέτουμε τα χαρακτηριστικά των ετικετών
                quickSortTime = new JLabel();
                quickSortTime.setText(" ");
                quickSortTime.setLocation(new Point(74, 151));
                quickSortTime.setSize(new Dimension(424, 16));
                initializingTime = new JLabel();
                initializingTime.setText(" ");
                initializingTime.setBounds(new Rectangle(77, 53, 418, 16));
                // Προσθέτουμε τα στοιχειά της
                // εφαρμογής μας στο πάνελ
                jContentPane.add(getInitializing(), null);
```

```
                jContentPane.add(initializingTime, null);
                jContentPane.add(getQuickSort(), null);
                jContentPane.add(getGridSort(), null);
                jContentPane.add(quickSortTime, null);

        }
        return jContentPane;
}

/**
 * Αυτή η μέθοδος αρχικοποιεί το κουμπί initializing
 *
 * @return javax.swing.JButton
 */
private JButton getInitializing()
{
        if (initializing == null)
        {

                // Δημιουργούμε το κουμπί
                initializing = new JButton();

                // Δίνουμε το όνομα που θέλουμε να εμφανίζεται στο κουμπί
                initializing.setText("Initializing");

                // Δίνουμε τις συντεταγμένες
                initializing.setBounds(new Rectangle(243, 13, 90, 26));

                // Προσθέτουμε ένα MouseListener
            initializing.addMouseListener(new java.awt.event.MouseAdapter()
                {
                      public void mouseClicked(java.awt.event.MouseEvent e)
                        {
                                // Παίρνουμε το χρόνο εκκίνησης
                                long start = System.currentTimeMillis();

                                // Δημιουργούμε ένα νέο αντικείμενο Random
                                Random jon = new Random();

                               // Δημιουργούμε το πίνακα με τυχαία στοιχειά
                                for (int i = 0; i < NUM; i++)
                                {
                                        test[i] = jon.nextInt(NUM);
                                }
                                // Παίρνουμε το χρόνο τερματισμού
                                long stop = System.currentTimeMillis();

                             // Κάνουμε τον υπολογισμό για να βρούμε το χρόνο
                             // που χρειάστηκε η αρχικοποίηση του πίνακα
                                double elapsed = (stop - start) / 1000.0;
                                initializingTime.setForeground(Color.red);
```

```
                                        // Εμφανίζουμε το χρόνο
                                        initializingTime
                                    .setText("The time to Initializing the table is "
                                            + String.valueOf(elapsed) + " sec");
                                }
                            });

                }
                return initializing;
        }

        /**
         *Αυτή η μέθοδος αρχικοποιεί το κουμπί gridSort
         *
         * @return javax.swing.JButton
         */
        private JButton getGridSort()
        {
                if (gridSort == null)
                {

                        // Δημιουργούμε το κουμπί
                        gridSort = new JButton();

                        // Δίνουμε το όνομα που θέλουμε να εμφανίζεται στο κουμπί
                        gridSort.setText("GridSort");

                        gridSort.setLocation(new Point(335, 102));
                        gridSort.setMnemonic(KeyEvent.VK_UNDEFINED);
                        gridSort.setSize(new Dimension(81, 26));
                        // Δίνουμε τις συντεταγμένες
                        // Προσθέτουμε ένα MouseListener
                        gridSort.addMouseListener(new java.awt.event.MouseAdapter()
                        {
                                @SuppressWarnings("deprecation")
                              public void mouseClicked(java.awt.event.MouseEvent e)
                                {
                                        try
                                        {

                                    // Δημιουργούμε ένα νέο στιγμιότυπο του πλέγματος
                                                GridFactory.start();
                                                Grid grid = GridFactory.getGrid();

                                        // Εκτελούμε τον πίνακα στο πλέγμα
                                       // βάση την υλοποίηση της κλάσης QuickSortTask
                                          test = (int[]) grid.execute(
                                          QuickSortTask.class.getName(), test).get();

                                    // Εμφανίζουμε το χρόνο
                                    quickSortTime.setForeground(Color.red);
                                    quickSortTime.setText("The time to sort the table is "
```

```
                                                + String.valueOf(time) + " sec");

                                        // Εμφανίζουμε τα 100 πρώτα στοιχειά του πίνακα για
                                        // έλεγχο
                                                for (int i = 0; i < 100; i++)
                                                {
                                                   System.out.print(test[i] + " ");
                                                }
                                                System.out.println();

                                        } catch (GridException ge)
                                        {
                                                System.out.println(ge.toString());
                                        } finally
                                        {
                                                GridFactory.stop(true);
                                        }
                                }
                        });

                }
                return gridSort;
        }

        /**
         * Αυτή η μέθοδος αρχικοποιεί το κουμπί quickSortMouseClicked
         *
         * @return javax.swing.JButton
         */
        private JButton getQuickSort()
        {
                if (quickSort == null)
                {

                        // Δημιουργούμε το κουμπί
                        quickSort = new JButton();

                        // Δίνουμε το όνομα που θέλουμε να εμφανίζεται στο κουμπί
                        quickSort.setText("QuickSort");

                        // Δίνουμε τις συντεταγμένες
                        quickSort.setBounds(new Rectangle(130, 102, 91, 26));

                        // Προσθέτουμε ένα MouseListener
                        quickSort.addMouseListener(new java.awt.event.MouseAdapter()
                        {
                                public void mouseClicked(java.awt.event.MouseEvent e)
                                {

                                        // Δημιουργούμε ένα νέο αντικείμενο
                                        QuickSortThread ms = new QuickSortThread();
```

```
                                        // Παίρνουμε το χρόνο εκκίνησης
                                        long start = System.currentTimeMillis();

                                        // Καλούμε την sort και περνάμε
                                        // σαν παράμετρο τον πίνακα
                                        ms.sort(test);

                                        // Παίρνουμε το χρόνο τερματισμού
                                        long stop = System.currentTimeMillis();

                                    // Κάνουμε τον υπολογισμό για να βρούμε το χρόνο
                                    // που χρειάστηκε η ταξινόμηση
                                        double elapsed = (stop - start) / 1000.0;

                                // Εμφανίζουμε το χρόνο
                                quickSortTime.setForeground(Color.red);
                                quickSortTime.setText("The time to sort the table is "
                                                + String.valueOf(elapsed) + " sec");

                                        // Εμφανίζουμε τα 100 πρώτα
                                        // στοιχεία του πίνακα για έλεγχο
                                        for (int i = 0; i < 100; i++)
                                        {
                                                System.out.print(test[i] + " ");
                                        }
                                        System.out.println();

                                    }
                                });

                        }
                        return quickSort;
                }

        } // @jve:decl-index=0:visual-constraint="291,9"
```

**QuickSortThread.java**

```
1.  package gr.teikal;
2.
3.  import java.util.concurrent.atomic.AtomicInteger;
4.
5.  public class QuickSortThread
6.  {
7.
8.          static Random random = new Random();
9.          private final AtomicInteger cpusAvailable;
10.         private final Object sync = new Object();
11.
12.         // Αριθμός των επεξεργαστών
13.         private final int nrOfCpus;
14.
15.         /**
16.          * Δημιουργεί ένα προκαθορισμένο στιγμιότυπο της quicksort με τον αριθμό των
17.          * διαθέσιμων επεξεργαστών του σιτέματος.
18.          * {@link Runtime#availableProcessors()}.
19.          */
20.         public QuickSortThread()
21.         {
22.                 this(Runtime.getRuntime().availableProcessors());
23.         }
24.
25.         /**
26.          * Δημιουργεί ένα στιγμιότυπο της quicksort ανάλογα με των αριθμό των
27.          * επεξεργαστών που δόθηκε. Τυπικά αυτό πρέπει να είναι
28.          * {@link Runtime#availableProcessors()}.
29.          *
30.          * @throws IllegalArgumentException
31.          *              Εαν η παραμετρος ειναι <= 0.
32.          */
33.         public QuickSortThread(final int nrOfCpus)
34.         {
35.                 if (nrOfCpus < 0)
36.                 {
37.                         throw new IllegalArgumentException("CPUs must be > 0");
38.                 }
39.                 this.nrOfCpus = nrOfCpus;
40.                 cpusAvailable = new AtomicInteger(this.nrOfCpus);
41.         }
42.
43.         /**
44.          * Ταξινομεί τον πίνακα που δόθηκε χρησιμοποιώντας μια παράλληλη
45.          * πολύ-νηματική quicksort. Για κάθε CPU ένα ξεχωριστό νήμα δημιουργείται.
```

```
46.          * Αν υπάρχει μονό μια CPU τότε χρησιμοποιείτε η σειριακή quicksort (κανένα
47.          * νήμα).
48.          *
49.          * @param items
50.          *            Ο πίνακας που θα ταξινομηθεί.
51.          * @throws IllegalArgumentException
52.          *             Αν οι παράμετροι είναι null.
53.          * @throws RuntimeException
54.          *             Αν κάποιο νήμα διακόπηκε.
55.          *
56.          */
57.         public void sort(final int[] items)
58.         {
59.
60.                 if (nrOfCpus <= 1)
61.                 {
62.                         QuickSort.sort(items);
63.                         return;
64.                 }
65.                 Thread thread = new Thread("WaitingThread")
66.                 {
67.
68.                         @Override
69.                         public void run()
70.                         {
71.                                 synchronized (sync)
72.                                 {
73.                                         try
74.                                         {
75.                                                 sync.wait();
76.                                         } catch (InterruptedException e)
77.                                         {
78.                                                 throw new RuntimeException(e);
79.                                         }
80.                                 }
81.                         }
82.                 };
83.                 thread.start();
84.                 quicksort(items, 0, items.length - 1);
85.                 try
86.                 {
87.                         thread.join();
88.                 } catch (InterruptedException e)
89.                 {
90.                         throw new RuntimeException(e);
91.                 }
92.         }
93.
94.         @SuppressWarnings(
95.         { "unchecked", "rawtypes" })
96.         private void quicksort(final int[] items, final int first,
97.                         final int last)
```

```
98.     {
99.
100.                    if (first < last)
101.                    {
102.                            int p = partition(items, first, last);
103.
104.                            if (cpusAvailable.intValue() > 0)
105.
106.                            {
107.                                    cpusAvailable.decrementAndGet();
108.                                    final int ii = p;
109.                                    new Thread("Quicksort")
110.                                    {
111.
112.                                            @Override
113.                                            public void run()
114.                                            {
115.                                                    quicksort(items, first, ii - 1);
116.                                                    returnCpuHandle();
117.                                            }
118.                                    }.start();
119.                            } else
120.                            {
121.                                    quicksort(items, first, p - 1);
122.                            }
123.                            if (cpusAvailable.intValue() > 0)
124.                            {
125.                                    cpusAvailable.decrementAndGet();
126.                                    final int ii = p;
127.                                    new Thread("Quicksort")
128.                                    {
129.
130.                                            @Override
131.                                            public void run()
132.                                            {
133.                                                    quicksort(items, ii + 1, last);
134.                                                    returnCpuHandle();
135.                                            }
136.                                    }.start();
137.                            } else
138.                            {
139.                                    quicksort(items, p + 1, last);
140.                            }
141.                    }
142.            }
143.
144.            private void returnCpuHandle()
145.            {
146.                    if (cpusAvailable.incrementAndGet() >= nrOfCpus)
147.                    {
148.                            synchronized (sync)
149.                            {
```

```
150.                                    sync.notify();
151.                            }
152.                    }
153.            }
154.
155.            public static int partition(int[] list, int first, int last)
156.            {
157.                    int p = first + random.nextInt(last - first + 1);
158.                    swap(list, first, p);
159.                    p = first;
160.                    for (int n = p + 1; n <= last; n++)
161.                    {
162.                            if (list[n] < list[p])
163.                            {
164.                                    swap(list, n, p + 1);
165.                                    swap(list, p, p + 1);
166.                                    p++;
167.                            }
168.                    }
169.                    return p;
170.            }
171.
172.            private static void swap(int[] list, int index1, int index2)
173.            {
174.                    int temp = list[index1];
175.                    list[index1] = list[index2];
176.                    list[index2] = temp;
177.            }
178.    }
```

**QuickSortTask.java**

```
1. package gr.teikal;
2.
3. import java.io.Serializable;
4. import java.util.ArrayList;
5. import java.util.Collection;
6. import java.util.List;
7.
8. import org.gridgain.grid.GridException;
9. import org.gridgain.grid.GridJob;
10. import org.gridgain.grid.GridJobAdapter;
11. import org.gridgain.grid.GridJobResult;
12. import org.gridgain.grid.GridTaskSplitAdapter;
13.
14. public class QuickSortTask extends
15.                 GridTaskSplitAdapter<int[], Serializable>
16. {
17.         // Λίστα με τους υπό-πινάκες
18.         ArrayList<int[]> array = new ArrayList<int[]>();
19.
20.         // Παίρνουμε το χρόνο εκκίνησης της split
21.         long start = System.currentTimeMillis();
22.
23.         /**
24.          *Σπάει αναδρομικά τον πίνακα σε κομμάτια και παίρνει κάθε κομμάτι στους
25.          * κόμβους ώστε να γίνει παράλληλα η εκτέλεση
26.          */
27.         public Collection<? extends GridJob> split(int gridSize, int[] A)
28.                         throws GridException
29.         {
30.                 // Το άθροισμα των χρονών που
31.                 // χρειάστηκε ώστε να ταξινομηθούν
32.                 // όλοι οι υπό-πινάκες
33.                 double sum = 0.0;
34.
35.                 // Ο διαιρέτης των χρονών του αθροίσματος sum
36.                 int flag = 0;
```

```
37.
38.             // Λίστα με τις εργασίες του πλέγματος
39.             List<GridJob> jobs = new ArrayList<GridJob>();
40.
41.             // Προσθέτει τον πίνακα στη λίστα
42.             // όταν φτάσει στο επιθυμητό μέγεθος
43.             if (A.length <= GridSortGui.NUM / gridSize)
44.
45.             {
46.                     // jobnum++;
47.                     array.add(A);
48.
49.             } else
50.
51.             {
52.                     int mid = A.length / 2;
53.                     int k = mid;
54.                     int[] B = new int[mid];
55.                     int[] C = new int[(A.length - mid)];
56.                     System.arraycopy(A, 0, B, 0, mid);
57.                     for (int i = 0; i < (A.length - mid); i++)
58.                     {
59.                             C[i] = A[k];
60.                             k++;
61.                     }
62.                     split(gridSize, B);
63.                     split(gridSize, C);
64.             }
65.             // Παίρνουμε το χρόνο τερματισμού της split
66.             long stop = System.currentTimeMillis();
67.
68.             // Κάνουμε τον υπολογισμό για να βρούμε το χρόνο
69.             // που χρειάστηκε ο ταξινόμησης του πίνακα
70.             double elapsed = (stop - start) / 1000.0;
71.             GridSortGui.time = 0.0;
72.             GridSortGui.time += elapsed;
73.
74.             for (final int[] arra : array)
75.             {
76.                     // Παίρνουμε το χρόνο εκκίνησης της ταξινόμησης
77.                     long start = System.currentTimeMillis();
78.                     // Δημιουργεί εργασίες από τη λίστα των εργασιών
79.                     jobs.add(new GridJobAdapter<int[]>(arra)
80.                     {
81.
82.                             public Serializable execute()
83.                             {
84.                             System.out.println("Sorting... Be patient");
85.                                 QuickSort.sort(arra);
86.
87.                                 return arra;
88.                             }
```

```
89.                                     });
90.                         // Παίρνουμε το χρόνο τερματισμού της ταξινόμησης
91.                         stop = System.currentTimeMillis();
92.
93.                         // Κάνουμε τον υπολογισμό για να βρούμε το χρόνο
94.                         // που χρειάστηκε η ταξινόμηση του υπό-πίνακα
95.                         elapsed = (stop - start) / 1000.0;
96.                         sum += elapsed;
97.                         flag++;
98.                 }
99.                     // Βρίσκουμε το μέσο ορό και το
100.                    // προσθέτουμε στη μεταβλητή time
101.                    GridSortGui.time += (sum / flag);
102.
103.                    return jobs;
104.            }
105.
106.        /**
107.         * Παίρνει τα αποτελέσματα από τον κάθε κόμβο και τα συνθέτει σ'ένα πίνακα.
108.         */
109.            public Serializable reduce(List<GridJobResult> results)
110.                                throws GridException
111.            {
112.
113.                    // Παίρνουμε το χρόνο εκκίνησης
114.                    long start = System.currentTimeMillis();
115.
116.                    int[] sorted ={};
117.
118.                    ArrayList<int[]> store = new ArrayList<int[]>();
119.                   // Λαμβάνει τα δεδομένα από τις επιτυχής εργασίες του πλέγματος
120.                    for (GridJobResult res : results)
121.                    {
122.                            store.add((int[]) res.getData());
123.                    }
124.                    // Ενώνει τους υπό-πινάκες ξεχωριστά
125.                    while (store.size() > 2)
126.                    {
127.                            collapse(store);
128.                    }
129.
130.                    if (store.size() <= 2)
131.                    {
132.                            if (store.size() == 2)
133.                            {
134.                    sorted = new int[(store.get(0).length +
135.                                            store.get(1).length)];
136.                                    merge(store.get(0), store.get(1), sorted);
137.                            }
138.                            if (store.size() == 1)
139.                            {
140.                                    sorted = new int[store.get(0).length];
```

```
141.                                    sorted = store.get(0);
142.                            }
143.                    }
144.                    // Παίρνουμε το χρόνο τερματισμού
145.                    long stop = System.currentTimeMillis();
146.
147.                    // Κάνουμε τον υπολογισμό για να βρούμε το χρόνο
148.                    // που χρειάστηκε η αρχικοποίηση του πίνακα
149.                    double elapsed = (stop - start) / 1000.0;
150.                    GridSortGui.time += elapsed;
151.                    return sorted;
152.            }
153.
154.            /**
155.             * Βοηθητική μέθοδος της reduce που ενώνει όλες τις εργασίες μέχρι να
156.             * μείνουν μονό δυο. Η reduce κάνει την τελευταία σύνθεση
157.             */
158.            private void collapse(ArrayList<int[]> store)
159.            {
160.                    int size = store.size();
161.                    if (size == 2)
162.                    {
163.                            return;
164.                    } else
165.                    {
166.                          int[] temp = new int[(store.get(size - 1).length + store
167.                                          .get(size - 2).length)];
168.                            merge(store.get(size - 1), store.get(size - 2), temp);
169.                            store.remove(size - 1);
170.
171.                            return;
172.                    }
173.            }
174.
175.            /**
176.             * Ενώνει δυο υπό-πινάκες σ'ένα πίνακα.
177.             *
178.             * @param left
179.             * @param right
180.             * @param whole
181.             */
182.            private static void merge(int[] left, int[] right, int[] whole)
183.            {
184.                    int leftIndex = 0;
185.                    int rightIndex = 0;
186.                    int wholeIndex = 0;
187.
188.                     // Εφόσον ούτε ο αριστερός ούτε ο δεξιός
189.                    //πίνακας δεν χρησιμοποιούνται,
190.                    //συνεχίζει να παίρνει το μικρότερο
191.                    // από τα left[leftIndex] ή από right[rightIndex]
192.                    //και τα πρόσθετη στο whole[wholeIndex]
```

```
193.                    while (leftIndex < left.length && rightIndex < right.length)
194.                    {
195.                            if (left[leftIndex] < right[rightIndex])
196.                            {
197.                                    whole[wholeIndex] = left[leftIndex];
198.                                    leftIndex++;
199.                            } else
200.                            {
201.                                    whole[wholeIndex] = right[rightIndex];
202.                                    rightIndex++;
203.                            }
204.                            wholeIndex++;
205.                    }
206.
207.                    int[] rest;
208.                    int restIndex;
209.                    if (leftIndex >= left.length)
210.                    {
211.                            // Ο αριστερός πίνακας χρησιμοποιείτε...
212.                            rest = right;
213.                            restIndex = rightIndex;
214.                    } else
215.                    {
216.                            // Ο δεξιός πίνακας χρησιμοποιείτε...
217.                            rest = left;
218.                            restIndex = leftIndex;
219.                    }
220.                    // Αντιγραφή το υπόλοιπο των πινάκων (αριστερό ή δεξιό)
221.                    //που δεν χρησιμοποιήθηκε
222.                    for (int i = restIndex; i < rest.length; i++)
223.                    {
224.                            whole[wholeIndex] = rest[i];
225.                            wholeIndex++;
226.                    }
227.            }
228.    }
```

```
1. package gr.teikal;
2.
3. import java.util.Random;
4.
5. public class QuickSort
6. {
7.         static Random random = new Random();
8.
9.         /**
```

```
10.         * Η μέθοδος sort καλεί τη μέθοδο
11.         * recursiveQuickSort με τις
12.         * απαραίτητες παραμέτρους
13.         * @param a
14.         */
15.        public static void sort(int[] a)
16.        {
17.
18.                recursiveQuickSort(a, 0, a.length - 1);
19.        }
20.
21.        /**
22.         * Η μέθοδος recursiveQuickSort καλεί
23.         * τη μέθοδο partition για να σπάσει υπολογίσει
24.         * που θα σπάσει ο πίνακας
25.         * @param list
26.         * @param first
27.         * @param last
28.         */
29.        public static void recursiveQuickSort(int[] list, int first, int last)
30.        {
31.                if (first < last)
32.                {
33.                        int p = partition(list, first, last);
34.                        // printList(list);
35.                        recursiveQuickSort(list, first, p - 1);
36.                        recursiveQuickSort(list, p + 1, last);
37.                }
38.        }
39.
40.        /**
41.         * Η μέθοδος partition καθορίζει
42.         * το στοιχειό διαχωρισμού
43.         * @param list
44.         * @param first
45.         * @param last
46.         * @return
47.         */
48.        public static int partition(int[] list, int first, int last)
49.        {
50.                int p = first + random.nextInt(last - first + 1);
51.                swap(list, first, p);
52.                p = first;
53.                for (int n = p + 1; n <= last; n++)
54.                {
55.                        if (list[n] < list[p])
56.                        {
57.                                swap(list, n, p + 1);
58.                                swap(list, p, p + 1);
59.                                p++;
60.                        }
61.                }
```

```
62.                 return p;
63.         }
64.
65.         /**
66.          * Η μέθοδος swap αντιμεταθέτει τα στοιχεία
67.          * του πίνακα
68.          * @param list
69.          * @param index1
70.          * @param index2
71.          */
72.         private static void swap(int[] list, int index1, int index2)
73.         {
74.                 int temp = list[index1];
75.                 list[index1] = list[index2];
76.                 list[index2] = temp;
77.         }
78.
79. }
```