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

Efficient Big Data Storage and Retrieval in Multimedia Cloud Computing Systems

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Τέντες

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π

Αθήνα, Σεπτέμβριος 2014

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

Efficient Big Data Storage and Retrieval in Multimedia Cloud Computing Systems

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Τέντες

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 19η Σεπτεμβρίου 2014.

....

.....

Γεώργιος Τέντες
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π

Ευχαριστίες

Η διπλωματική αυτή εκπονήθηκε στο Εργαστήριο Distributed Knowledge and Media Systems Group του Τομέα Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου, υπό την επίβλεψη της Καθηγήτριας Θεοδώρας Βαρβαρίγου.

Θα ήθελα να ευχαριστήσω θερμά την κ. Θεοδώρα Βαρβαρίγου για την υποστήριξη και καθοδήγηση που μου προσέφερε, καθώς και για την ευκαιρία που μου έδωσε να ασχοληθώ με την ερευνητική περιοχή του Cloud Computing.

Επίσης, θα ήθελα να ευχαριστήσω ιδιαίτερα τον υποψήφιο διδάκτορα Βρεττό Μουλό για την πολύτιμη βοήθειά του καθόλη τη διάρκεια της εργασίας, καθώς και τους ερευνητές: Μάρκο Ζάμπογλου, Γιώργο Βαφειάδη και Αθανασία Ευαγγελινού που με τις στοχευμένες συμβουλές τους συνέβαλλαν στην εκπόνηση αυτής της διπλωματικής εργασίας.

Περίληψη

Η μεγαλύτερη πρόκληση των σύγχρονων υπολογιστικών συστημάτων είναι αναμφισβήτητα η αποδοτική αποθήκευση και ανάκτηση πολύ μεγάλου όγκου δεδομένων. Η ανάγκη αυτή έκανε την εμφάνισή της τα τελευταία χρόνια λόγω της έκρηξης δεδομένων που παρατηρείται στο διαδίκτυο και αποκτά ολοένα και μεγαλύτερη σημασία λόγω του πολύ μεγάλου εύρους πληροφοριών που μπορούμε να αντλήσουμε.

Στην παρούσα εργασία μελετάμε τρόπους αποδοτικής αποθήκευσης και αναζήτησης πολυμεσικών δεδομένων σε συστήματα υπολογιστικού νέφους. Λόγω της πληθώρας πληροφοριών που περιέχεται στα πολυμεσικά δεδομένα, το μέγεθος τους μας αποτρέπει να κάνουμε χρήση παραδοσιακών τεχνικών αποθήκευσης όπως η χρήση των σχεσιακών συστημάτων βάσεων δεδομένων, και επεξεργασίας, όπως το xml parsing, και έτσι αναζητούμε λύσεις με χρήση τεχνικών Big Data.

Η λύση που προτείνουμε, αφορά σε πρώτο στάδιο τη χρήση μιας μη σχεσιακής (NoSQL) βάσης δεδομένων για την αποθήκευση, και ακολούθως τη χρήση του μοντέλου Map Reduce για την εξαγωγή χρήσιμης πληροφορίας από τα πολυμεσικά δεδομένα. Η βάση δεδομένων που χρησιμοποιούμε είναι η MongoDB λόγω της πολύ καλής κλιμακωσιμότητας που προσφέρει, γεγονός υψίστης σημασίας στα συστήματα υπολογιστικού νέφους. Επιπλέον, με τη χρήση του μοντέλου Map Reduce μπορούμε να επιτύχουμε κατανεμημένη επεξεργασία των δεδομένων, εκμεταλλευόμενοι έτσι μεγάλο αριθμό επεξεργαστών για να επιτύχουμε καλύτερους χρόνους εκτέλεσης των ερωτημάτων.

Λέξεις Κλειδιά

Cloud Computing, NoSQL βάσεις δεδομένων, Multimedia, MPEG-7, X3D, MongoDB, Μη σχεσιακές βάσεις δεδομένων, ενδιάμεσο λογισμικό, μεταδεδομένα, υπολογιστικό νέφος

Abstract

Efficient Big Data storage and retrieval is undoubtedly the biggest challenge faced by modern computing systems. In the last few years, this necessity has become more obvious due to the huge data explosion that is currently taking place on the internet and it has become a critical issue because of the wide variety of information we can retrieve from all this data.

In this thesis, we study various ways of efficient storage and searching of multimedia data in cloud computing systems. Due to the plethora of information included in multimedia data, their total size does not allow us to make use of tradition storing techniques such as the use of relation database systems, or searching techniques such as xml parsing, therefore we try applying some of the more modern Big Data techniques.

The approach presented in this thesis, regards, first of all, the use of a non relational (NoSQL) database for storage, and afterwards, the use of the Map Reduce programming model for extracting useful information out of multimedia data. The database we chose is MongoDB because of the high scalability potential, which plays a vital role in cloud computing systems. Moreover, using Map Reduce, we can achieve distributed data processing in a very efficient way, allowing us to execute queries in a very short time.

Keywords

Cloud Computing, NoSQL databases, Multimedia, MPEG-7, X3D, MongoDB, Non Relational Databases, Middleware, Metadata

Περιεχόμενα

Κατάλογος Σχημάτων

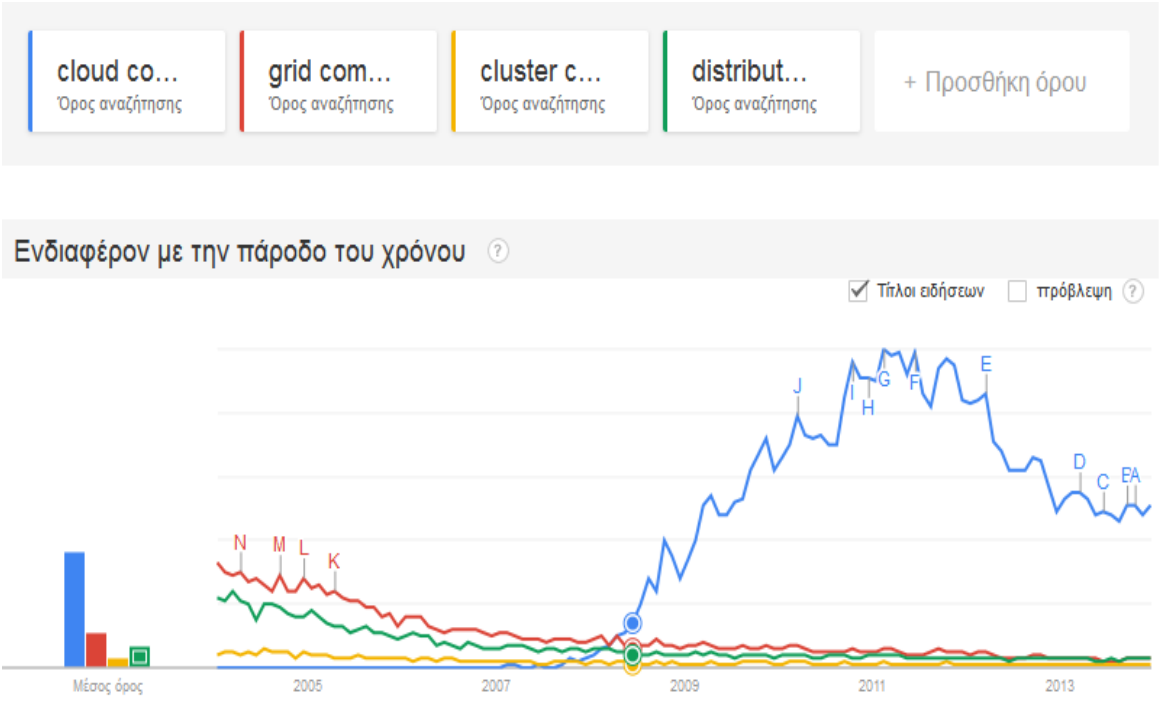
ΚΕΦΑΛΑΙΟ 1

Cloud Computing

1.1 Ορισμός

Το cloud computing (υπολογιστικό νέφος) είναι ένας από τους πιο δημοφιλείς όρους στον τομέα της πληροφορικής τα τελευταία χρόνια (όπως φαίνεται και στην εικόνα 1), και αναμένεται να συνεχίσει να απασχολεί ερευνητές και επιχειρήσεις για πολλά χρόνια ακόμα. Η εικόνα που επικρατεί στο ευρύ κοινό σχετικά με το “σύννεφο” είναι πως πολλοί υπολογιστές από διαφορετικά σημεία της γης συνδέονται μέσω διαδικτύου και παρέχουν από κοινού τις υπηρεσίες τους στους πελάτες. Η εικόνα αυτή δεν απέχει και πολύ από την πραγματικότητα. Αναζητήσας ένα πιο πλήρη ορισμό λοιπόν, βλέπουμε πως μέχρι σήμερα έχουν προταθεί πολλοί διαφορετικοί ορισμοί για το cloud computing. Αυτό γιατί από τη φύση του είναι λίγο δύσκολο να οριστεί επακριβώς, μιας και δεν μπορεί να χαρακτηριστεί ως μία συγκεκριμένη νέα τεχνολογία ή εφαρμογή, αλλά μάλλον ως ένα νέο μοντέλο εργασιών και υπηρεσιών που συνδυάζει υπάρχουσες τεχνολογίες. Ωστόσο, ένας ορισμός που αναδεικνύει με σαφήνεια τα κυριότερα χαρακτηριστικά του cloud computing είναι αυτός που χρησιμοποιείται από το NIST (The National Institute of Standards and Technology), σύμφωνα με τον οποίο “Το cloud computing είναι ένα μοντέλο που επιτρέπει ευέλικτη, on-demand δικτυακή πρόσβαση σε ένα κοινόχρηστο σύνολο παραμετροποιήσιμων υπολογιστικών πόρων (π.χ. δίκτυα, servers, αποθηκευτικοί χώροι, εφαρμογές και υπηρεσίες), το οποίο μπορεί να τροφοδοτηθεί γρήγορα και να διατεθεί με ελάχιστη προσπάθεια διαχείρισης ή αλληλεπίδραση με τον πάροχο της υπηρεσίας.".

Γίνεται σαφές επομένως πως στο cloud, η πληροφορική προσφέρεται ως μια ολοκληρωμένη υπηρεσία. Οι χρήστες έχουν τη δυνατότητα να χρησιμοποιήσουν μέσω του διαδικτύου υπολογιστικούς πόρους χωρίς να εμπλέκονται καθόλου με την τεχνολογική υποδομή που τους υποστηρίζει, το οποίο αναλαμβάνεται εξ ολοκλήρου από τους παρόχους. Εισάγεται έτσι ένα νέο επίπεδο αφαίρεσης, που δεν υπήρχε παλιότερα και δίνει τη δυνατότητα στο χρήστη να απεμπλακεί πλήρως από το φυσικό σύστημα που θέλει να χρησιμοποιήσει, παρέχοντας του ένα εικονικό σύστημα που μπορεί να το παραμετροποιήσει όπως και όποτε εκείνος επιθυμεί.



Σχήμα 1: Αριθμός αναζητήσεων με την πάροδο του χρόνου για τους όρους: cloud computing (μπλε), grid computing (κόκκινο), cluster computing (πορτοκαλί), distributed computing (πράσινο).

1.2 Ιστορική Αναδρομή

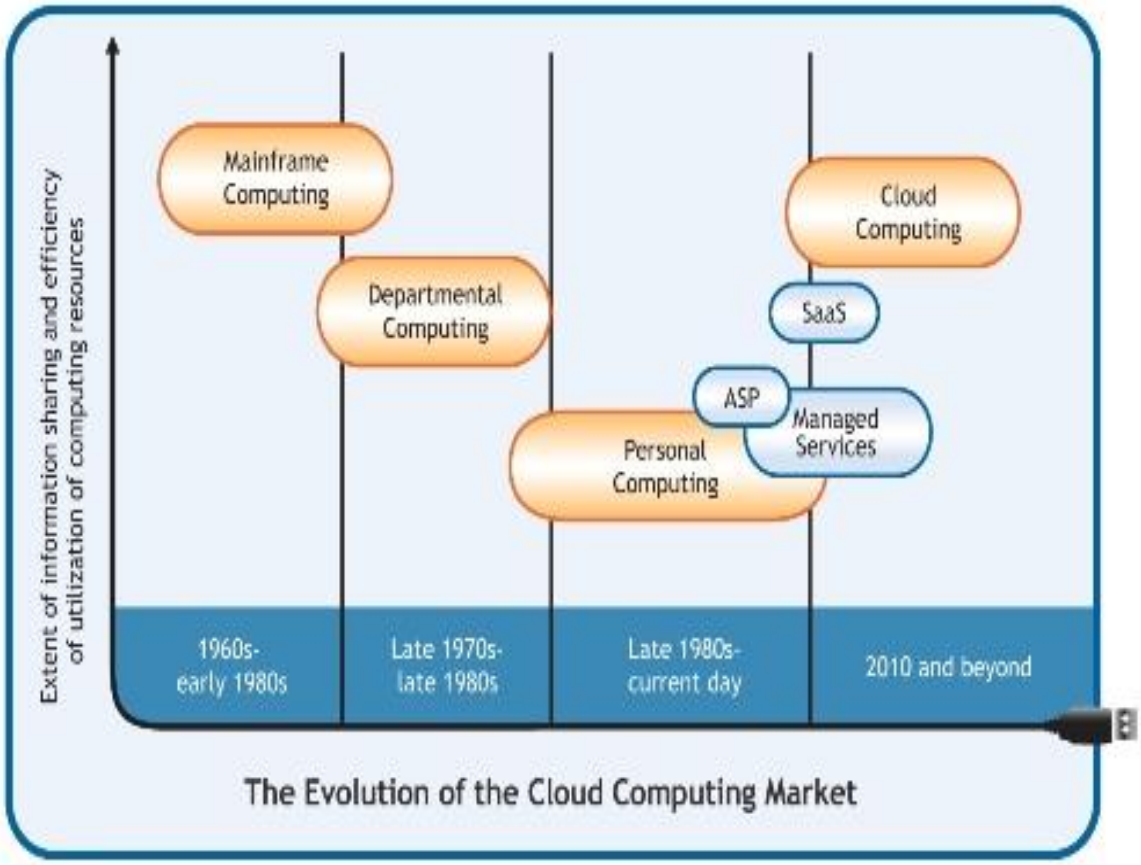
Το cloud computing, στην πιο πρωτόγονη μορφή του, εμφανίστηκε στη δεκαετία του 1950, όπου ερευνητές και επιχειρήσεις μπορούσαν να συνδεθούν μέσω τερματικού, σ' έναν mainframe υπολογιστή ώστε να τρέξουν τις εφαρμογές τους στο δικό του, πιο ισχυρό hardware. Οι χρήστες μοιράζονταν τους πόρους που διέθεταν οι συγκεκριμένοι υπολογιστές, αποκτώντας έτσι τη δυνατότητα να αναπτύξουν πιο απαιτητικές εφαρμογές, χωρίς να επιβαρύνονται το επιπλέον κόστος για το υλικό.

Αργότερα, στη δεκαετία του 1960 γεννήθηκε η ιδέα ενός παγκόσμιου δικτύου υπολογιστών (J.C.R Licklider – Intergalactic Computer Network) , πρόγονου του διαδικτύου και του υπολογιστικού νέφους. Στην ίδια δεκαετία, αποτυπώθηκε για πρώτη φορά η άποψη πως η πληροφορική θα μπορούσε κάποια στιγμή να χρησιμοποιηθεί ως δημόσιο αγαθό (J. McCarthy - “Computing may someday be organized as a public utility). Φυσικά, με την τότε τεχνολογία, και οι δύο αυτές απόψεις φάνταζαν ουτοπικές.

Με τη διαρκή πρόοδο στους τομείς του hardware και των δικτύων όμως, οι

εταιρείες τηλεπικοινωνιών ξεκίνησαν την παροχή υπηρεσιών εικονικών δικτύων (VPNs) προς το κοινό, ενώ η εξάπλωση του διαδικτύου, και η εξέλιξη του virtualization, είχαν ως αποτέλεσμα και την παροχή εικονικών servers.

Το πρώτο ίσως βήμα προς τη διαμόρφωση του cloud computing όπως το ξέρουμε σήμερα, έγινε το 1999 με την ίδρυση της Salesforce, την πρώτη εταιρία που παρείχε επιχειρησιακές εφαρμογές μέσα από ένα απλό website. Έκτοτε, η εξέλιξη ήταν ραγδαία. Το 2002 είχαμε την ανάπτυξη του Amazon Web Services και το 2006 το Amazon Elastic Compute Cloud (EC2), που ήταν η πρώτη φορά που υπηρεσίες υποδομής (IaaS) έγιναν διαθέσιμες στο ευρύ κοινό. Το 2009, με την καθιέρωση του Web 2.0 είχαμε την ανάπτυξη των πρώτων υπηρεσιών πλατφόρμας (PaaS) όπως το Google App Engine και το Microsoft Azure Services Platform.



Σχήμα 2: Evolution of the Cloud Computing Market

1.3 Σύντομη Περιγραφή

Αφού δώσαμε έναν σαφή ορισμό και εξετάσαμε την ιστορική πορεία του cloud computing, θα επικεντρωθούμε τώρα στα κυριότερα χαρακτηριστικά που το προσδιορίζουν και το διαφοροποιούν από άλλα μοντέλα:

On-demand self-service: ο χρήστης μπορεί να επιλέξει κατά βούληση την ποσότητα υπολογιστικών πόρων που θα χρησιμοποιήσει, και να ρυθμίζει την κατανάλωση αυτή ανάλογα με τις ανάγκες του, εντελώς ανεξάρτητα, χωρίς να απαιτείται δηλαδή άμεση, ανθρώπινη αλληλεπίδραση με τον πάροχο. Για την μεγαλύτερη διευκόλυνση των χρηστών, ειδικά για τους λιγότερο εξειδικευμένους, συνήθως παρέχονται ορισμένες επιλογές με προκαθορισμένες ρυθμίσεις παραμέτρων που ταιριάζουν στις πιο συνηθισμένες χρήσεις του cloud. Έτσι, ακόμα και κάποιος με μηδαμινές τεχνικές γνώσεις μπορεί να κάνει χρήση μιας cloud εφαρμογής.

Resource Pooling: Το σύνολο των πόρων -εικονικών και φυσικών- που μπορεί να διαθέσει ο πάροχος διατίθενται στους χρήστες μέσω ενός πολυπελατειακού μοντέλου, δηλαδή εκχωρούνται δυναμικά και αναδιατάσσονται ανάλογα με τη ζήτηση του κάθε πελάτη. Αυτό σημαίνει πως δεν υπάρχει μόνιμη σύνδεση κάποιου χρήστη με κάποιο συγκεκριμένο πόρο και έτσι δεν μπορούν να γνωρίζουν την ακριβή τοποθεσία τους ή να έχουν κάποιον έλεγχο πάνω σε αυτούς, αφού είναι διαρκώς μεταβαλλόμενοι.

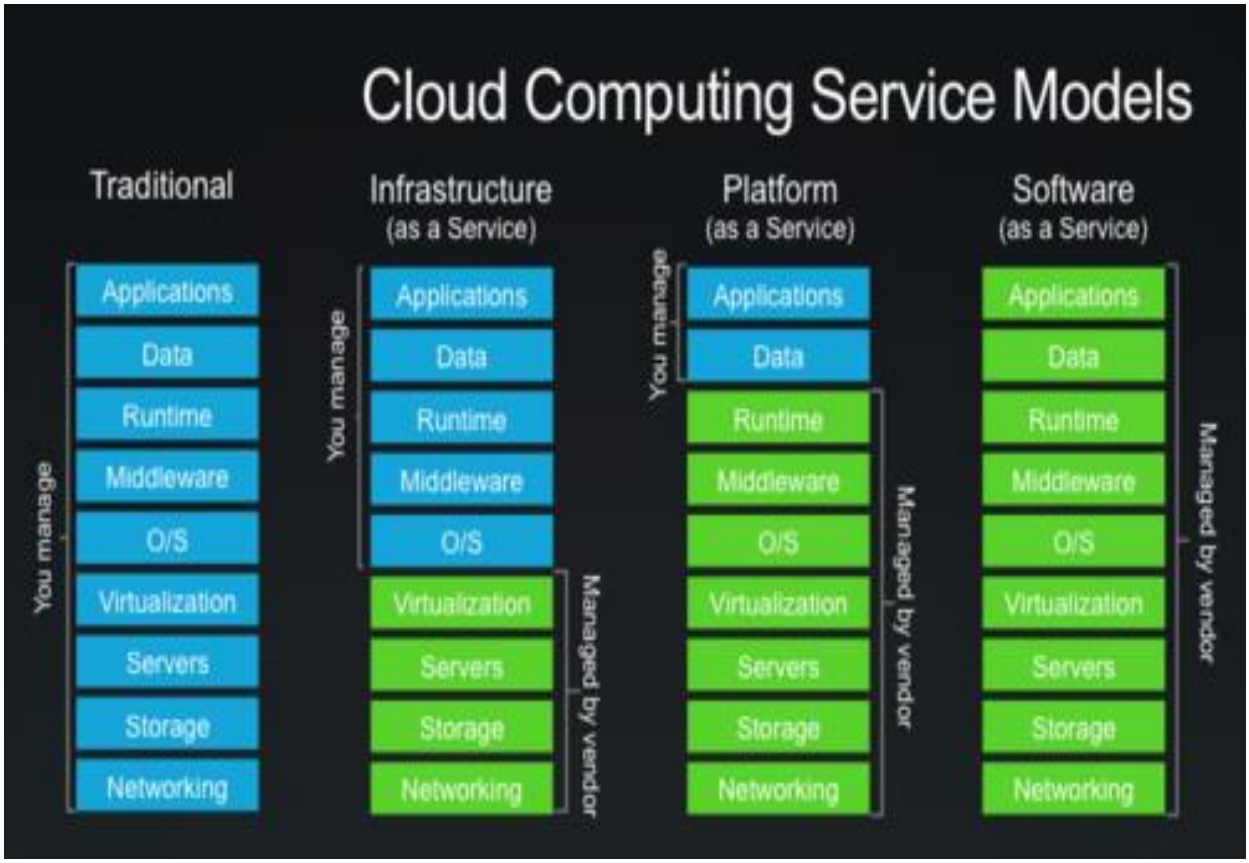
Scalability and Flexibility: Ίσως τα σημαντικότερα χαρακτηριστικά για την εξάπλωση του cloud computing. Με την άμεση (πολλές φορές και αυτόματη) δέσμευση ή αποδέσμευση πόρων, οι χρήστες είναι σε θέση να εξυπηρετούν οποιαδήποτε ανάγκη προκύψει, χωρίς να την έχουν προβλέψει. Οι cloud υπηρεσίες χαρακτηρίζονται από το μεγάλο βαθμό κλιμάκωσης που έχουν, είτε όσον αφορά τη γεωγραφική κατανομή τους, αφού μπορούν να χρησιμοποιούν πόρους κυριολεκτικά από οποιοδήποτε μέρος της γης, είτε σε επίπεδο hardware, αφού διευκολύνεται πάρα πολύ η δυναμική προσάρτηση νέων κόμβων στο σύστημα, και οι εφαρμογές σχεδιάζονται με γνώμονα την καλύτερα αξιοποίηση αυτής της δυνατότητας.

Multitenancy: μια εφαρμογή μπορεί να χρησιμοποιείται από πολλούς διαφορετικούς χρήστες με διαμοιρασμό τον πόρων, αυξάνοντας έτσι την αποδοτικότητα της. Δεν απαιτείται ένα διαφορετικό αντίγραφο της εφαρμογής για κάθε χρήστη, και συνεπώς επιτυγχάνεται μεγαλύτερη εξοικονόμηση φυσικών και εικονικών πόρων.

Utility-based pricing: η πολιτική χρέωσης που εφαρμόζουν οι περισσότεροι cloud πάροχοι είναι pay-per-use. Δεν υπάρχει δηλαδή κάποια fixed χρέωση, αλλά κάθε χρήστης κοστολογείται ανάλογα με τους πόρους που δεσμεύει. Με αυτό τον τρόπο δίνεται η δυνατότητα σε νέους χρήστες να δημιουργήσουν εφαρμογές χωρίς ιδιαίτερες απαιτήσεις σε αρχικό κεφάλαιο και χωρίς την ανάγκη για απόσβεση της αρχικής επένδυσης. Δίνεται χώρος δηλαδή για νέες καινοτόμες ιδέες ακόμα και με μεγάλο βαθμό ρίσκου, που υπό άλλες συνθήκες, το απαιτούμενο αρχικό κεφάλαιο και η αβεβαιότητα επιτυχίας θα τις καθιστούσε αποτρεπτικές.

Geo-distribution and ubiquitous network access: Οι cloud υπηρεσίες είναι διαθέσιμες μέσω διαδικτύου, επομένως οι χρήστες μπορούν να συνδεθούν από οποιοδήποτε μέρος και από οποιαδήποτε συσκευή με την προϋπόθεση ότι είναι συνδεδεμένοι στο Internet. Ακόμα, οι μεγάλοι cloud πάροχοι διαθέτουν data centers σε διάφορα σημεία του κόσμου, αυξάνοντας έτσι τις επιδόσεις των δικτύων τους. Μπορεί να γίνει δηλαδή αντιστοίχιση των χρηστών με τα data centers που βρίσκονται γεωγραφικά κοντά τους, μειώνοντας το latency.

Οι προσφερόμενες υπηρεσίες μπορούν να χωριστούν σε 3 βασικά επίπεδα: IaaS (Infrastracture as a Service – Υποδομή ως Υπηρεσία), PaaS (Platform as a Service – Πλατφόρμα ως Υπηρεσία) και SaaS (Software as a Service - Λογισμικό ως Υπηρεσία).



Σχήμα 3: Cloud Service Models

Στο χαμηλότερο επίπεδο (IaaS), ο πάροχος προσφέρει υπολογιστική υποδομή δηλαδή μνήμη, επεξεργαστές, λειτουργικά συστήματα, δίκτυα κλπ είτε αυτούσια, είτε συχνότερα μέσα από εικονικές μηχανές (VMs). Μέσω της εικονικοποίησης, ο πάροχος δίνει στο χρήστη την αίσθηση της ύπαρξης απεριόριστων πόρων. Έτσι, οι ίδιοι φυσικοί πόροι, μοιράζονται στους χρήστες σε διαφορετικές χρονικές στιγμές ανάλογα με τις απαιτήσεις του καθενός (statistical multiplexing). Οι χρήστες επωφελούνται από το γεγονός ότι τους δίνεται η δυνατότητα να δουλέψουν σε μεγάλες υπολογιστικές μονάδες, χωρίς να επιβαρύνονται το κόστος για την αγορά και συντήρηση τους, παρά μόνο το κόστος ανάλογα με τη χρήση τους (pay as you go). Ακόμα, δεν έχουν καμία δέσμευση ως προς το πώς θα χρησιμοποιήσουν τη δοθείσα υποδομή. Μπορούν να εγκαταστήσουν και να τρέξουν οποιαδήποτε εφαρμογή της αρεσκείας τους, ή να χρησιμοποιήσουν το διαθέσιμο hardware για την υποστήριξη νέων, δικών τους εφαρμογών. Από την άλλη, οι πάροχοι αν και αναλαμβάνουν την αγορά του φυσικού εξοπλισμού και είναι υπεύθυνοι για τη συντήρησή του, έχουν τη δυνατότητα να εξυπηρετούν μεγάλο αριθμό χρηστών, για συγκεκριμένο χρονικό διάστημα τον καθένα, εξασφαλίζοντας έτσι μεγαλύτερα κέρδη απ'ότι αν έκαναν μια 1-1αντιστοιχία πόρων – χρηστών.

Ο κυρίαρχος πάροχος IaaS είναι η Amazon με το EC2 (Elastic Cloud) του πακέτου

Amazon Web Services. Το EC2 δίνει στους χρήστες τη δυνατότητα να νοικιάσουν υπολογιστικούς πόρους, να δημιουργήσουν εικονικές μηχανές και να αναπτύξουν τις δικές τους υπηρεσίες οι οποίες θα τρέχουν στο cloud. Η Amazon δίνει τη δυνατότητα στους χρήστες να παραμετροποιήσουν πλήρως τις εικονικές μηχανές που θα χρησιμοποιήσουν, αφού μπορούν να κάνουν επιλογές σχετικά με τις βιβλιοθήκες, εφαρμογές και τα δεδομένα που θα υπάρχουν στο virtual machine, αλλά και σχετικά με τις ρυθμίσεις δικτύου, ασφαλείας και λειτουργίας του κάθε vm. Τα κυριότερα χαρακτηριστικά του Amazon EC2 έχουν να κάνουν με:

Αξιοπιστία

Η Amazon αναλαμβάνει την ευθύνη για τη διαθεσιμότητα των υπηρεσιών της σε ποσοστό 99.95% και για την ανθεκτικότητα των δεδομένων σε ποσοστό 99.999999999% ! Αυτό σημαίνει πως αν ένας χρήστης που έχει αποθηκευμένα 10.000 αντικείμενα, η Amazon του εγγυάται πως η μέγιστη απώλεια που μπορεί να έχει είναι ένα αντικείμενο κάθε 10.000.000 χρόνια! Αυτό εξασφαλίζεται με τη διατήρηση πολλαπλών αντιγράφων των αντικειμένων σε διαφορετικές τοποθεσίες και συσκευές, καθώς και με τον τακτικό έλεγχο της ακεραιότητας αυτών των αντιγράφων.

Ασφάλεια

Παρέχονται διάφοροι μηχανισμοί ασφαλείας και ελέγχου πρόσβασης στους πόρους. Αυτοί έχουν να κάνουν κυρίως με τη δημιουργία και ρύθμιση firewalls για τον έλεγχο της κίνησης μεταξύ χρηστών και vm, αλλά και τη δημιουργία ιδιωτικού νέφους (Amazon Virtual Cloud – AWP).

Ευελιξία

Παρέχεται πληθώρα επιλογών στους χρήστες σχετικά με την επιλογή του vm, του λειτουργικού και των πακέτων λογισμικού που θα χρησιμοποιήσουν.
Ανάλογα με τις ανάγκες τους μπορούν να ρυθμίσουν τη CPU, τη μνήμη RAM, τον αποθηκευτικό χώρο, το λειτουργικό σύστημα κ.α.

Ελαστικότητα

Όπως τονίσαμε παραπάνω υπάρχει η δυνατότητα στους χρήστες να διαμορφώσουν τα μηχανήματά τους ανάλογα με τις ανάγκες τους. Πέραν αυτού όμως, η διαμόρφωση αυτή μπορεί να γίνεται και δυναμικά, να μεταβάλλεται δηλαδή κάποιο στοιχείο ανάλογα και με τη μεταβολή των αναγκών του χρήστη. Το στοιχείο αυτό είναι πάρα πολύ σημαντικό κυρίως για όσους αναπτύσσουν εμπορικές εφαρμογές, καθώς είναι πολύ δύσκολο να προβλεφθεί εξ αρχής το μέγεθος της ζήτησης.

Έλεγχος

Οι χρήστες έχουν τον απόλυτο έλεγχο των μηχανών τους. Μέσω web service APIs μπορούν να τα εκκινήσουν και να τα τερματίσουν όποτε θέλουν, αλλά και να τα ελέγχουν μέσω root access.

Συμβατότητα με Amazon Web Services

Το EC2 παρέχει την υπολογιστική υποδομή, αλλά για την ανάπτυξη μεγάλων εφαρμογών δεν αρκεί μόνο αυτό. Απαιτούνται ακόμη χώροι αποθήκευσης των δεδομένων αλλά και βάσεις δεδομένων για τη διαχείρισή τους. Την πρόσθετη αυτή λειτουργικότητα την παρέχουν το Amazon Simple Storage Service S3 (αποθηκευτικό χώρο), το Amazon Relational Database Service (σχεσιακή βάση δεδομένων) και το Amazon SimpleDB (NoSQL βάση).

Πέρα από αυτά, το Amazon EC2 παρέχει ορισμένες επιπλέον ευκολίες στους χρήστες:

- Αρχικά, τους δίνεται η δυνατότητα να δεσμεύσουν τα μηχανήματα τους με 3 διαφορετικούς τρόπους, ώστε να προσαρμόζεται η χρέωση καλύτερα στον καθένα. Έτσι, μπορούν να επιλέξουν on demand instances για περιπτώσεις όπου δεν μπορούν να προβλέψουν εκ των προτέρων τη χρήση τους ή αναμένουν μεγάλες και συχνές μεταβολές στις ανάγκες τους. Η χρέωση τους τότε γίνεται ανάλογα με τη χρήση των πόρων ανά ώρα. Ο δεύτερος τρόπος είναι μέσω των reserved instances, όπου οι χρήστες επιλέγουν από πριν τον τύπο instance που θα χρησιμοποιήσουν (Light, Medium ή Heavy) αντίστοιχα με τη χρήση που αναμένουν να κάνουν και έχουν έτσι μια σημαντική έκπτωση στην ωριαία χρέωσή τους. Τέλος, μέσω των spot Instances οι χρήστες μπορούν να ορίσουν οι ίδια μια τιμή και όταν αυτή είναι μεγαλύτερη από την Spot Price, να προσφέρεται μια εικονική μηχανή στο χρήστη, αλλιώς να αποδεσμεύεται. Η μέθοδος αυτή ενδείκνυται κυρίως για fault tolerant εφαρμογές.

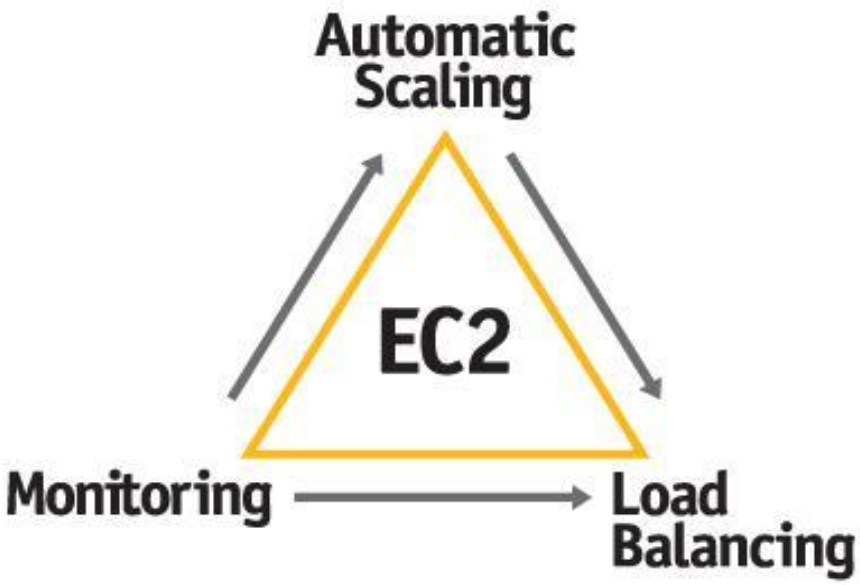
- Η υποδομή του Amazon συνίσταται από περιοχές (Regions) και Availability Zones (ζώνες διαθεσιμότητας). Τα Regions βρίσκονται σε διάφορα μέρη του πλανήτη ενώ έχει μία ή περισσότερες availability zones το καθένα. Μεταξύ των ζωνών της ίδιας περιοχής επιτυγχάνεται ταχύτερη δικτυακή μεταφορά δεδομένων. Έτσι οι χρήστες μπορούν να επωφεληθούν από αυτό, επιλέγοντας να αποθηκευτούν τα δεδομένα τους στην ίδια περιοχή ώστε να εξασφαλίσουν χαμηλό latency ή σε διαφορετικές, ώστε να εξασφαλίσουν ακόμα μεγαλύτερη διαθεσιμότητα.

- Η Amazon παρέχει την υπηρεσία ελέγχου Cloud Watch που δίνει στους χρήστες τη δυνατότητα να ελέγχουν τις εφαρμογές και τους πόρους που χρησιμοποιούν. Οι χρήστες επιλέγουν τα μηχανήματα που θέλουν να παρακολουθήσουν, και λαμβάνουν έτσι πληροφορίες και μετρικές μπορούν να χρησιμοποιήσουν είτε για τη συλλογή

στατιστικών αποτελεσμάτων είτε για την αυτόματη κλιμάκωση των πόρων τους. Οι μετρικές αυτές έχουν να κάνουν με τη χρήση της CPU, τα read/writes στο δίσκο, την κίνηση του δικτύου κ.α.

- Μέσω του Elastic Load Balancing επιτυγχάνεται μεγάλη ανοχή σε σφάλματα. Εδώ ο πάροχος αναλαμβάνει να κατανείμει τα requests από τις εφαρμογές σε instances με χαμηλό φόρτο εργασίας, ώστε να αποφορτίζονται αυτά που δεν είναι σε θέση να ικανοποιήσουν τα requests και συνεπώς να μη δημιουργούνται σφάλματα.

- Μέσω του AWS Marketplace δίνεται στους χρήστες η δυνατότητα να εκμεταλλευτούν εμπορικά τα μηχανήματά τους. Συγκεκριμένα, μπορούν να πωλούν και να αγοράζουν είτε σε άλλους χρήστες είτε στην Amazon, τα reserved instances που έχουν δημιουργήσει οι ίδιοι.

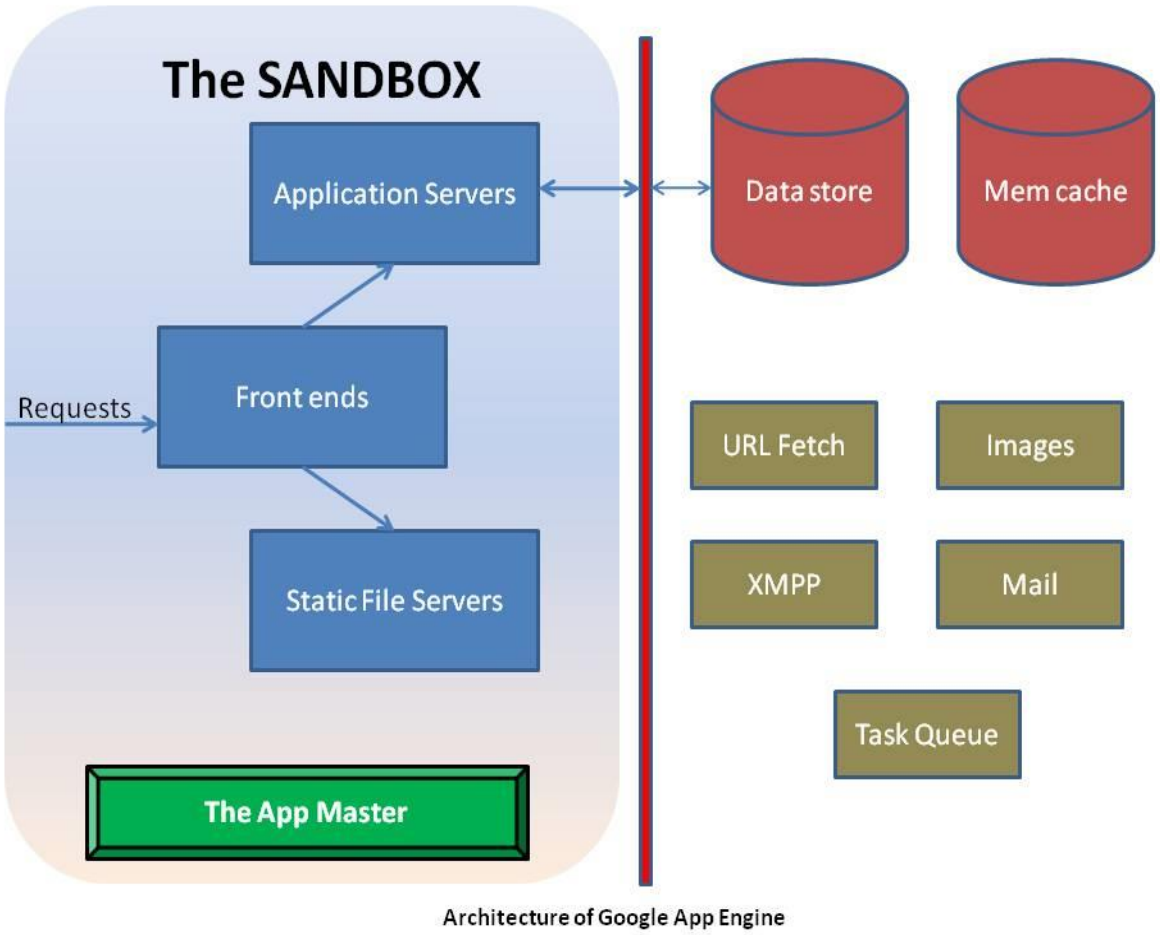


Σχήμα 4: Amazon EC2

Στο επόμενο επίπεδο (PaaS) προσφέρεται η πλατφόρμα, πάνω στην οποία ο πελάτης μπορεί να αναπτύξει και να τρέξει εφαρμογές. Συνήθως μια τέτοια υπηρεσία περιλαμβάνει εγκατεστημένο λειτουργικό σύστημα, προγραμματιστικό περιβάλλον και βιβλιοθήκες, προκαθορισμένη βάση δεδομένων και web server. Οι προγραμματιστές που τρέχουν τις εφαρμογές τους στο cloud επωφελούνται από την αυτόματη κλιμάκωση και την εξισορρόπηση φορτίου, καθώς και από άλλες υπηρεσίες

που τους προσφέρονται από τον πάροχο και διευκολύνουν την ανάπτυξη και διαχείριση των εφαρμογών, όπως υπηρεσίες ηλ. Ταχυδρομείου, διεπαφές χρηστών κλπ.

Ένα παράδειγμα τέτοιας υπηρεσίας είναι η Google App Engine. Μέσω του πακέτου Google App Engine παρέχονται και οι τρεις τύποι παροχής cloud υπηρεσιών, ωστόσο εμείς θα εστιάσουμε στο PaaS. Το App Engine μπορεί να φιλοξενήσει οποιουδήποτε είδους διαδικτυακές εφαρμογές, ωστόσο είναι σχεδιασμένο κυρίως για την ανάπτυξη δυναμικών, πραγματικού χρόνου (real time) εφαρμογών, που χρειάζεται να εξυπηρετούν ταυτόχρονα πολλούς χρήστες. Το κυριότερο χαρακτηριστικό της είναι η αυτόματη κλιμάκωση για διαδικτυακές εφαρμογές. Έτσι, οι πόροι που διατίθενται για κάθε εφαρμογή μεταβάλλονται αυτόματα ανάλογα με τον αριθμό των χρηστών που τη χρησιμοποιούν, ώστε να μπορεί διαχειριστεί είτε μια αύξηση της ζήτησης, είτε να μη δεσμεύει άσκοπα πόρους σε περίπτωση μείωσής της. Όσον αφορά το μοντέλο χρέωσης, διαφέρει ως προς το κλασσικό μοντέλο που χρησιμοποιούν οι εξυπηρετητές (servers) για τη φιλοξενία ιστοσελίδων, αφού ακολουθείται το τυπικό cloud μοντέλο pay-as-you-go. Αυτό σημαίνει πως δεν υπάρχουν πάγιες χρεώσεις, αλλά χρεώνονται οι πόροι όπως η χρήση της CPU, η αποθήκευση δεδομένων, το εύρος ζώνης κ.α., ανάλογα με το βαθμό χρησιμοποίησής τους. Επιπλέον, παρέχεται η δυνατότητα σε μικρές εφαρμογές να φιλοξενούνται χωρίς κάποιο κόστος, αφού διατίθεται 1GB αποθηκευτικού χώρου και κίνησης δωρεάν. Η αρχιτεκτονική του App Engine έχει ως εξής:
Ένας client στέλνει ένα http request μέσω του web browser. Το αίτημα αυτό το παίρνει από το frontend ένας εξισορροπητής φόρτου (load balance) κι ένα σύστημα για τη βέλτιστη δρομολόγηση των αιτημάτων και έτσι δρομολογείται σε έναν από τους frontend servers. Τώρα, αν το αίτημα αυτό αφορά ένα στατικό αρχείο, ο frontend server το μεταβιβάζει στον server στατικών αρχείων, επιστρέφεται το αρχείο και έτσι ολοκληρώνεται το αίτημα. Αν όμως, το αίτημα δε μπορεί να ικανοποιηθεί από κάποιο στατικό αρχείο, δηλαδή αφορά κώδικα που πρέπει να τρέξει η εφαρμογή, τότε αποστέλλεται σε κάποιον από τους servers εφαρμογών (app servers) ο οποίος ξεκινά ένα στιγμιότυπο ενός της εφαρμογής ή χρησιμοποιεί κάποιο υπάρχον, ώστε να διεκπεραιώσει το request.



Σχήμα 5: Google App Engine Architecture

Στο ανώτερο επίπεδο (SaaS), ο πελάτης μπορεί να κάνει χρήση έτοιμων εφαρμογών που υπάρχουν στο cloud μέσω του φυλλομετρητή ιστού. Ο πάροχος είναι υπεύθυνος για την υποδομή πάνω στην οποία θα τρέξει η εφαρμογή αλλά και για τη συντήρηση της. Ο πελάτης δε χρειάζεται επιβαρύνει το σύστημά του με επιπλέον πόρους για την εφαρμογή αυτή, γεγονός πολύ σημαντικό όταν μιλάμε για απαιτητικές εφαρμογές που μπορεί να μειώσουν αισθητά την απόδοση του συστήματος. Ακόμα, αποβάλλεται από τον πελάτη η ευθύνη για τη συντήρηση της εφαρμογής (δε χρειάζεται να κατεβάζει και να εγκαθιστά ανανεωμένες εκδόσεις) αφού όλα γίνονται στο cloud. Ο πιο διαδεδομένος πάροχος SaaS υπηρεσιών είναι και πάλι η Google μέσω των έτοιμων εφαρμογών προς χρήση που διαθέτει (Google Apps). Σε αυτές περιλαμβάνονται το Gmail, τα Google Calendar, Drive, Docs, Sheets, Slides κ.α. Η μόνη προϋπόθεση για να κάνει κάποιος χρήση των εφαρμογών αυτών είναι η πρόσβαση στο διαδίκτυο και για κάποιες ένα Google Account.

Ο χρήστης έχει τη δυνατότητα να ρυθμίσει και να παραμετροποιήσει τις εφαρμογές ώστε να εξυπηρετούν καλύτερα τις ανάγκες του. Η Google αναλαμβάνει

να τρέξει την εφαρμογή, να αποθηκεύσει τα δεδομένα και τις προσωπικές ρυθμίσεις του κάθε χρήστη καθώς και να συντηρεί και να αναβαθμίζει το λογισμικό των εφαρμογών αυτών. Είναι φανερός λοιπόν ο τρόπος με τον οποίο η ευθύνη για το τρέξιμο και τη συντήρηση μιας εφαρμογής στο cloud, μετατοπίζεται πλήρως στον πάροχο, αφήνοντας το χρήστη ελεύθερο να επικεντρωθεί στο κύριο μέρος της δουλειάς του.

Σχήμα 6: Google Apps

Αφού είδαμε τις υπηρεσίες που συναντάμε στο cloud, ας εξετάσουμε τώρα τα μοντέλα ανάπτυξης κάτω από τα οποία αυτές προσφέρονται.

Public Cloud: Εδώ οι πάροχοι κάνουν διαθέσιμες τις υπηρεσίες τους στο ευρύ κοινό με τη χρεωστική πολιτική pay-as-you-go. Οι χρήστες δηλαδή δε χρεώνονται από πριν για δεδομένη ποσότητα υπολογιστικών πόρων, αλλά χρεώνονται ανάλογα με το βαθμό χρησιμοποίησης τους. Έτσι, οι πάροχοι απαλλάσσονται από το αρχικό κόστος της επένδυσης σε υπολογιστικό εξοπλισμό, ενώ οι πελάτες δε χρειάζεται να ανησυχούν για την κλιμάκωση της εφαρμογής τους, αφού έχουν στη διάθεση τους εικονικά άπειρους πόρους (available on demand).

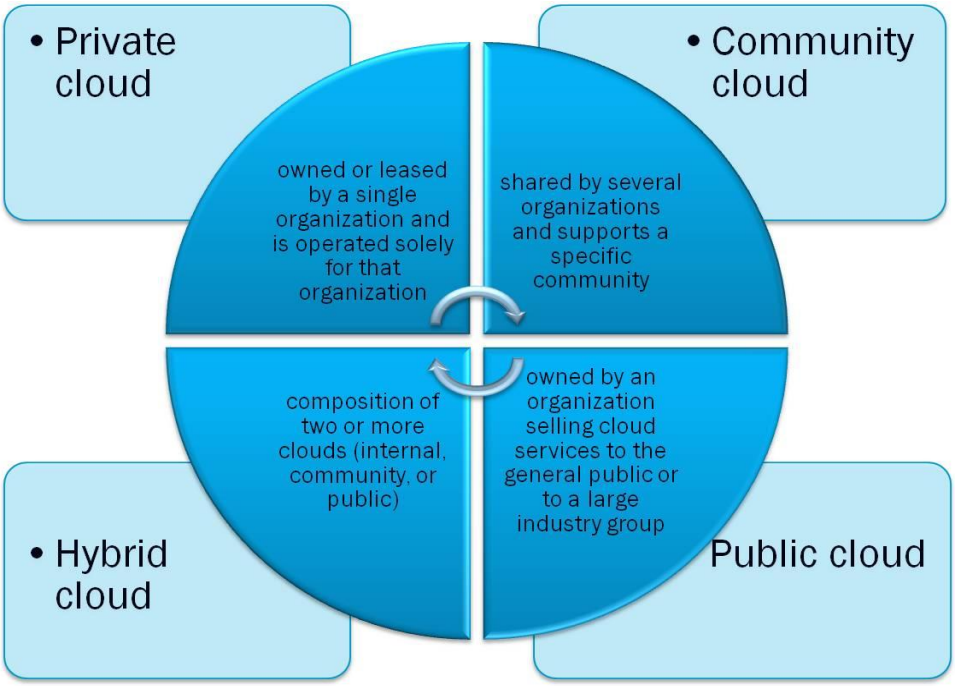
Private Cloud: Τα ιδιωτικά νέφη σχεδιάζονται για αποκλειστική χρήση από έναν

οργανισμό ή επιχείρηση σχεδιασμός και η συντήρησή τους μπορεί να γίνεται και από εξωτερικούς παράγοντες, ωστόσο η χρήση τους περιορίζεται μέσα στα πλαίσια και της ανάγκες του ίδιου του οργανισμού.

Hybrid Cloud: Εδώ έχουμε τη σύνθεση δύο ή περισσοτέρων μοντέλων cloud (public ή private), με σκοπό να ξεπεραστούν οι περιορισμοί που τίθενται από τα συγκεκριμένα μοντέλα. Τα επιμέρους στοιχεία παραμένουν μοναδικές οντότητες, ωστόσο διευκολύνεται η μεταξύ τους επικοινωνία, προσφέροντας έτσι μεγαλύτερη ευελιξία.

Virtual Private Cloud: Αποτελούν μια εναλλακτική λύση για την αντιμετώπιση των περιορισμών που θέτουν τα public και private clouds. Τα VPC τρέχουν πάνω σε δημόσια νέφη αλλά παρέχουν εικονικοποίηση του δικτύου που βρίσκεται από κάτω τους, επιτρέποντας έτσι στους παρόχους να διαμορφώνουν την τοπολογία κατά βούληση.

Η επιλογή του κατάλληλου μοντέλου ανάπτυξης εξαρτάται σε μεγάλο βαθμό από το σενάριο χρήσης του, ωστόσο προβλέπεται πως τα hybrid clouds θα είναι τα δημοφιλέστερα μεταξύ των περισσοτέρων οργανισμών.



Σχήμα 7: Cloud Deployment Models

1.4 Πλεονεκτήματα και μειονεκτήματα του Cloud Computing

Για πολλούς το cloud computing αποτελεί μια επανάσταση στον τομέα της πληροφορικής που θα φέρει μεγάλες αλλαγές στον τρόπο που αναπτύσσεται το λογισμικό, ενώ για άλλους δεν αποτελεί τίποτα παραπάνω από μια απλή αναδιατύπωση όρων και τεχνολογιών που ήδη χρησιμοποιούμε. Η αλήθεια είναι πως το υπολογιστικό νέφος συγκεντρώνει πολλά πλεονεκτήματα που μπορούν να το καταστήσουν ορόσημο στην εξέλιξη της πληροφορικής, αλλά παράλληλα για να συμβεί αυτό θα πρέπει να αντιμετωπιστούν ορισμένοι κινδύνους και προκλήσεις που εμφανίζει. Ας δούμε μερικά από τα θετικά στοιχεία του cloud:

- ***Ευελιξία***. Οι χρήστες μπορούν να έχουν πρόσβαση στις cloud υπηρεσίες από οποιοδήποτε σημείο της γης, αρκεί να είναι συνδεδεμένοι στο διαδίκτυο. Αυτό το στοιχείο είναι πολύ σημαντικό αφού διαχωρίζει ουσιαστικά τα δεδομένα και την πληροφορία από το φυσικό μέσο. Έτσι πχ, ένας απλός χρήστης μπορεί να ανεβάσει τα αρχεία του στο cloud και να έχει πρόσβαση σε αυτά οπουδήποτε και αν βρίσκεται τη στιγμή που αν τα κρατούσε στο σκληρό του δίσκο, θα είχε πρόσβαση σε αυτά μόνο μέσω του τοπικού του υπολογιστή.

- ***Εικονικά άπειροι διαθέσιμοι πόροι και pay as you go χρέωση.*** Πλέον αποβάλλεται από τον χρήστη η ανάγκη να προβλέψει εξ αρχής την ποσότητα των υπολογιστικών πόρων που θα χρειαστεί. Με αυτό τον τρόπο, αφενός γλυτώνει πολλά χρήματα σε περίπτωση λανθασμένης εκτίμησης, αφετέρου έχει τη δυνατότητα να μεταβάλλει δυναμικά το υλικό που χρησιμοποιεί και κοστολογείται βάση αυτού, ανάλογα με τις ανάγκες του κάθε χρονική στιγμή. Έτσι, ένας προγραμματιστής ιστοσελίδων, μπορεί να ξεκινήσει τη δημιουργία ενός site με μηδαμινό αρχικό κόστος και να προσθέτει servers και αποθηκευτικές μονάδες δυναμικά, όσο αυξάνει η επισκεψιμότητα του. Παρατηρούμε πως με αυτό τον τρόπο ενθαρρύνεται και η επιχειρηματικότητα, αφού το αρχικό κεφάλαιο που απαιτείται για την ανάπτυξη μιας web υπηρεσίες μειώνεται κατακόρυφα.

- ***Μεγαλύτερη αξιοπιστία και διαθεσιμότητα.*** Χρησιμοποιώντας το παράδειγμα μιας web υπηρεσίας που είδαμε και πριν, ο προγραμματιστής επιλέγοντας να την αναπτύξει στο cloud απαλλάσσεται από το βάρος της συντήρησης του υλικού και την ευθύνη για οποιοδήποτε hardware failure. Η ευθύνη αυτή πλέον βαραίνει τον πάροχο, και ο χρήστης προστατεύεται μέσω των SLAs. Ακόμα, ο προγραμματιστής δε χρειάζεται να ανησυχεί για τη διάθεση updates και patches στους χρήστες, αφού όλοι

“βλέπουν” την τελευταία έκδοση της εφαρμογής.

Πέρα από τις εμφανείς διευκολύνσεις και ευελιξίες που παρέχει η τεχνολογία του cloud computing, δεν θα πρέπει να παραγνωρίσουμε ορισμένους από τους κινδύνους και τις ερευνητικές προκλήσεις που το συνοδεύουν.

- ***Ασφάλεια και ιδιωτικότητα.*** *Ο κυριότερος λόγος που ορισμένοι* αντιμετωπίζουν με δυσπιστία τις cloud εφαρμογές έχει να κάνει με την ασφάλεια και την ιδιωτικότητα των δεδομένων τους. Η ασφάλεια έχει να κάνει κυρίως με το αν υπάρχει κίνδυνος να χαθούν τα δεδομένα, ενώ η ιδιωτικότητα έχει να κάνει με το αν κάποιος τρίτος (πέραν του πελάτη και του παρόχου) μπορεί να αποκτήσει πρόσβαση σε αυτά αλλά και σε τι βαθμό έχει πρόσβαση ο ίδιος ο IaaS πάροχος.

- ***Εξάρτηση από τον πάροχο.*** Εδώ εμφανίζεται το μειονέκτημα του κατά πόσο είναι εύκολο για κάποιον πελάτη να μεταφέρει τα δεδομένα του από έναν cloud πάροχο σε κάποιον άλλο (data migration και vendor lock-in). Στις περισσότερες περιπτώσεις αυτό δεν είναι καθόλου εύκολο, ειδικά όταν μιλάμε για τεράστιους όγκους δεδομένων, και έτσι ο πελάτης δεσμεύεται ως ένα βαθμό με τον πάροχο.

1.5 Εξέλιξη και σύγκριση με το Grid Computing

Αναμφίβολα το cloud computing εμφανίζει πολλές ομοιότητες με το Grid και θα μπορούσε να θεωρηθεί εξέλιξή του. Και στα δύο μοντέλα, στόχος είναι η δυνατότητα πρόσβασης σε περισσότερη υπολογιστική ισχύ, με μικρότερο κόστος και περισσότερη αξιοπιστία και ευελιξία. Η πληροφορική παρέχεται πλέον ως υπηρεσία και η διαχείριση των υπολογιστικών μονάδων γίνεται από τρίτους. Παρόλα αυτά, υπάρχουν ορισμένα στοιχεία που τα διαφοροποιούν.

Το Grid Computing θα μπορούσε να χαρακτηριστεί ως μία προσπάθεια για αποκεντροποιημένο συντονισμό και διαχείριση ετερογενών υπολογιστικών πόρων που είναι κατανεμημένοι σε διαφορετικούς εικονικούς οργανισμούς. Για την επίτευξη υψηλής αποδοτικότητας χρησιμοποιούνται τυποποιημένα, γενικού σκοπού πρωτόκολλα και διεπαφές. Στο cloud αυτό δεν ισχύει, αφού η διαχείριση των πόρων γίνεται συνήθως από μια κεντρική μονάδα.

Τα Grids απευθύνονται κυρίως σε ερευνητικές ομάδες. Κάθε εργαστήριο που λαμβάνει μέρος σε κάποιο project, κάνει τους πόρους του διαθέσιμους σε όλα τα άλλα μέλη του έργου, αλλά και μπορεί να κάνει χρήση των πόρων των υπολοίπων. Στο cloud από την άλλη, έχουμε ένα ξεκάθαρο επιχειρησιακό μοντέλο, όπου παρέχονται υπηρεσίες πληροφορικής στο ευρύ κοινό, και ο κάθε χρήστης κοστολογείται με βάση την κατανάλωσή του. Ακριβώς όπως στο ηλεκτρικό ρεύμα ή στο νερό δηλαδή.

Σε πιο τεχνικό επίπεδο, παρατηρούμε σημαντικές διαφορές ως προς τη διαχείριση του hardware. Το virtualization αποτελεί ένα από τα θεμέλια του cloud computing. Οι διαθέσιμοι πόροι εικονικοποιούνται και συντονίζονται από την κεντρική διαχείριση του συστήματος. Στο grid όμως, αν και έχουν γίνει προσπάθειες προς αυτή την κατεύθυνση (Nimbus), κατά κανόνα δε γίνεται χρήση virtualization, και συνεπώς κάθε οργανισμός είναι αποκλειστικός υπεύθυνος για τους πόρους που παρέχει. Παρακάτω φαίνονται συγκριτικά οι κυριότερες διαφορές μεταξύ των δύο μοντέλων:

Parameter	**Grid computing**	**Cloud computing**
Goal	Collaborative sharing of resources	Use of service (eliminates the detail)
Computational focuses	Computationally intensive operations	Standard and high-level instances
Workflow management	In one physical node	In EC2 instance (Amazon EC2+S3)
Level of abstraction	Low (more details)	High (eliminate details)
Degree of scalability	Normal	High
Multitask	Yes	Yes
Transparency	Low	High
Time to run	Not real-time	Real-time services
Requests type	Few but large allocation	Lots of small allocation

Allocation unit	Job or task (small)	All shapes and sizes (wide & narrow)
Virtualization	Not a commodity	Vital
Portal accessible	Via a DNS system	Only using IP (no DNS registered)
Transmission	Suffered from internet delays	Was significantly fast
Security	Low (grid certificate service)	High (Virtualization)
Infrastructure	Low level command	High level services (SaaS)
Operating System	Any standard OS	A hypervisor (VM) on which multiple OSs run
Ownership	Multiple	Single
Interconnection network	Mostly internet with latency and low bandwidth	Dedicated, high-end with low latency and high bandwidth
Discovery	Centralized indexing and decentralized info services	Membership services
Service negotiation	SLA based	SLA based
User management	Decentralized and also Virtual Organization (VO)-based	Centralized or can be delegated to third party
Resource management	Distributed	Centralized/Distributed
Allocation/Scheduling	Decentralized	Both centralized/decentralized
Interoperability	Open grid forum standards	Web Services (SOAP and REST)

Failure management	Limited (often failed tasks/applications are restarted)	Strong (VMs can be easily migrated from one node to other)
Pricing of services	Dominated by public good or privately assigned	Utility pricing, discounted for larger customers
User friendly	Low	High
Type of service	CPU, network, memory, bandwidth, device, storage, ...	IaaS, PaaS, SaaS, Everything as a Service
Data intensive storage	Suited for that	Not suited for that
Example of real world	SETI, BOINC, Folding@home, GIMPS	Amazon Web Service (AWS), Google Apps
Response Time	Can't be serviced at a time and need to be scheduled	Real-time
Critical object	Computer resource	Service
Number of users	Few	More
Resource	Limited (because hardware are limited	Unlimited
Configuration	Difficult as users haven't administrator privilege	Very easy to configure
Future	Cloud computing	Next generation of internet

Πηγή: Cloud Computing Vs. Grid Computing S.M.Hashemi, A.K.Bardsiri

1.6 Εφαρμογές

Τα πλεονεκτήματα που παρέχει το cloud computing ευνοούν την ανάπτυξη εφαρμογών που απαιτούν μεγάλη επεκτασιμότητα (scalability), απομακρυσμένη πρόσβαση σε δεδομένα, ή μεγάλη υπολογιστική ισχύ. Ορισμένες από αυτές τις εφαρμογές είναι:

Επιχειρησιακές εφαρμογές (business applications): Μεγάλα συστήματα (ERP, CRM κλπ) που χρησιμοποιούνται για το στρατηγικό σχεδιασμό των επιχειρήσεων, είναι συνήθως πολύ απαιτητικά σε υπολογιστική ισχύ. Η λύση του cloud λοιπόν φαντάζει μονόδρομος, από τη στιγμή που το μέγεθος των δεδομένων προς επεξεργασία για τις μεγάλες επιχειρήσεις αυξάνεται εκθετικά.

Εφαρμογές για κινητές συσκευές: Τα τελευταία χρόνια οι κινητές συσκευές αποκτούν όλο και μεγαλύτερο μερίδιο της αγοράς για τις διαδικτυακές υπηρεσίες. Οι υπηρεσίες αυτές για να ανταποκριθούν κατάλληλα, θα πρέπει να εξασφαλίσουν υψηλή διαθεσιμότητα αλλά και τη δυνατότητα για άμεση επεξεργασία μεγάλου πλήθους δεδομένων. Τα δύο αυτά χαρακτηριστικά υποδεικνύουν το cloud ως την καλύτερη λύση.

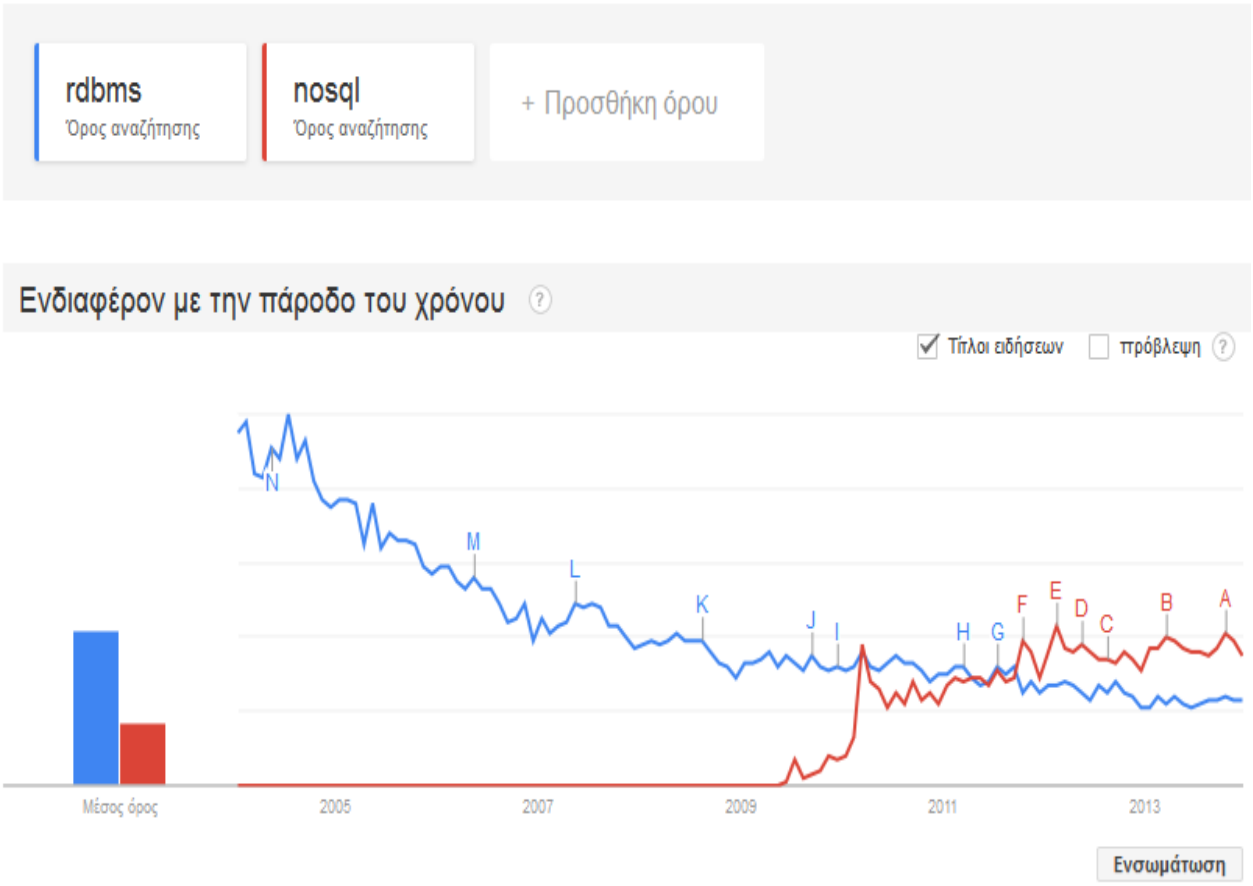
Εφαρμογές με υψηλό βαθμό παραλληλοποίησης: Η σημερινή έκρηξη δεδομένων έχει ως συνέπεια την ανάγκη για γρήγορη επεξεργασία τεράστιου όγκου πληροφορίας. Αυτό με τις συμβατικές τεχνολογίες είναι αδύνατον να επιτευχθεί. Στο cloud ωστόσο, με την ύπαρξη εικονικά απεριόριστου hardware (on demand), μπορούμε να μοιράσουμε τα δεδομένα μας σε εκατοντάδες διαφορετικούς υπολογιστές για να επιτύχουμε γρηγορότερη επεξεργασία τους. Μάλιστα, οι πάροχοι διαθέτουν έτοιμα περιβάλλοντα (MapReduce, Hadoop) για τη διευκόλυνση αυτών των εφαρμογών.

ΚΕΦΑΛΑΙΟ 2

NoSQL Databases

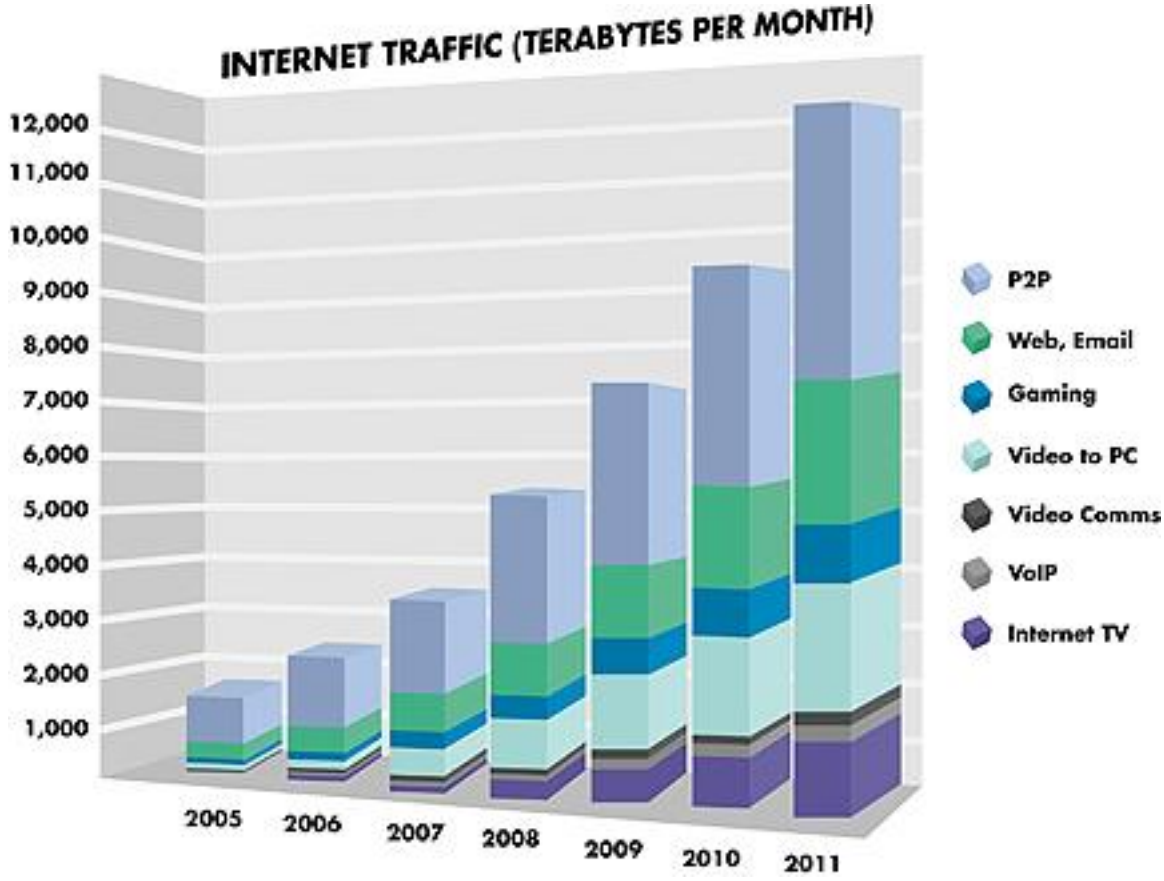
2.1 Ορισμός

Οι σχεσιακές βάσεις δεδομένων αποτελούσαν για πολλά χρόνια την κυρίαρχη επιλογή για δικτυακές και επιχειρησιακές εφαρμογές. Λόγω της σχεδόν καθολικής επικράτησης τους μάλιστα, δημιουργήθηκαν πολλά εργαλεία που διευκολύνουν και εμπλουτίζουν τη χρήση τους. Το σχεσιακό σχήμα που υλοποιείται με τη χρήση δισδιάστατων πινάκων και κλειδιών, καλύπτει τις ανάγκες για ένα πολύ μεγάλο μέρος εφαρμογών μέχρι τώρα. Με την έλευση του cloud computing ωστόσο και την ανάγκη για αποδοτικό, κατανεμημένο χειρισμό όλο και μεγαλύτερου και πιο ποικιλόμορφου όγκου δεδομένων, πολλοί οργανισμοί έστρεψαν το ενδιαφέρον τους σε εναλλακτικούς τρόπους διαχείρισης και αποθήκευσης δεδομένων. Έτσι, δημιουργήθηκαν πολλά συστήματα βάσεων δεδομένων που δεν υπακούν στο σχεσιακό μοντέλο, και πλέον κατέχουν μεγάλο μερίδιο της αγοράς ιδίως σε διαδικτυακές εφαρμογές. Τα συστήματα αυτά έγιναν γνωστά ως “NoSQL”. Αν και ο όρος παραπέμπει σε “Όχι- SQL”, θα ήταν ορθότερο να τον αποδώσουμε ως “Όχι μόνο SQL” ή “Μη – σχεσιακό”.



Σχήμα 8: NoSQL vs RDBMS

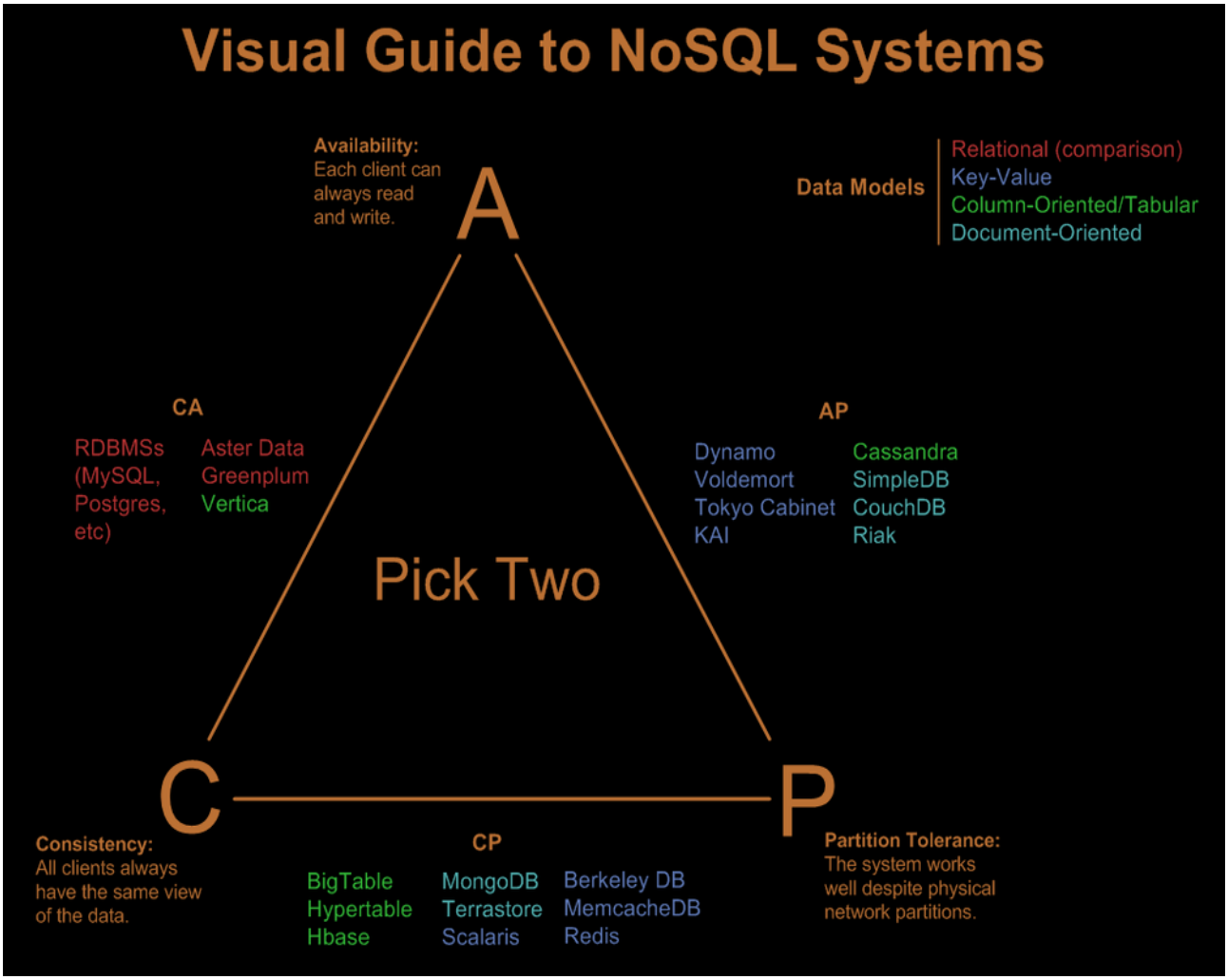
Στην παραπάνω εικόνα φαίνεται η αύξηση της δημοτικότητας των NoSQL βάσεων συγκριτικά με τα παραδοσιακά RDBMS. Το διάστημα 2009-2010 παρατηρούμε μια κατακόρυφη αύξηση του ενδιαφέροντος για τις μη σχεσιακές βάσεις, που συνεχίζει να αυξάνει (με χαμηλότερο ρυθμό) μέχρι και σήμερα. Αυτό συμβαδίζει με την εξίσου απότομη αύξηση του όγκου των δεδομένων που διακινούνται στο διαδίκτυο, όπως φαίνεται και παρακάτω:



Σχήμα 9: Internet traffic

Αν και τα συστήματα αυτά μπορούν να χωριστούν σε πολλές κατηγορίες, έχουν έναν κοινό άξονα 4 σημείων: είναι μη σχεσιακά, κατανεμημένα, ανοικτού κώδικα και οριζόντια κλιμακώσιμα. Ακόμη υπακούν στο θεώρημα CAP, σύμφωνα με το οποίο ένα κατανεμημένο υπολογιστικό σύστημα δε μπορεί να παρέχει ταυτόχρονα συνέπεια (consistency), διαθεσιμότητα (availability) και ανοχή τμημάτων (partition tolerance). Με τον όρο συνέπεια εννοούμε πως όλοι οι κόμβοι του συστήματος "βλέπουν" τα ίδια δεδομένα στην ίδια κατάσταση, κάθε χρονική στιγμή. Με τον όρο διαθεσιμότητα εννοούμε πως κάθε ενέργεια του πελάτη θα λάβει πάντα μια απάντηση (επιτυχής / ανεπιτυχής). Ενώ με τον όρο ανοχή τμημάτων εννοούμε πως το σύστημα θα συνεχίσει να λειτουργεί παρά μια βλάβη σε ένα τμήμα του. Τα περισσότερα RDBMS παρέχουν συνέπεια και διαθεσιμότητα, ενώ στα NoSQL συναντάμε και συνδυασμούς CP ή AP αλλά σε καμία περίπτωση και των τριών (CAP). Αυτό πρακτικά σημαίνει πως πολλές φορές στα NoSQL συστήματα είτε επιλέγουμε ένα πιο χαλαρό μοντέλο

συνέπειας, είτε θυσιάζουμε τη διαθεσιμότητα του συστήματος ώστε να επιτύχουμε μεγαλύτερη ανοχή τμημάτων.



Σχήμα 10: CAP Theorem

2.2 Σύγκριση με τις SQL βάσεις

Τα σχεσιακά συστήματα βάσεων δεδομένων παρέχουν πολύ υψηλή απόδοση για αναζητήσεις βασισμένες στις σχέσεις μεταξύ των οντοτήτων, καθώς και για δεδομένα με αυστηρή δομή και ομοιομορφία. Μάλιστα η εύκολη στην κατανόηση μορφή τους

και η πληθώρα εφαρμογών που διευκολύνουν τη διαχείριση τους, τα κάνει ιδιαίτερα προσιτά σε αρχάριους – μη εξειδικευμένους χρήστες. Ωστόσο, τα δεδομένα μας πολλές φορές δεν έχουν αυστηρά καθορισμένη μορφή ή πιθανώς να μας ενδιαφέρει περισσότερο η οριζόντια κλιμάκωση όπως συμβαίνει στα κατανεμημένα συστήματα. Σε τέτοιες περιπτώσεις ενδείκνυται η χρήση NoSQL συστημάτων. Μία τέτοια βάση δεδομένων μπορεί να βρίσκεται διαμοιρασμένη σε μεγάλο αριθμό από επεξεργαστές, ελαφρύνοντας έτσι το φόρτο του καθενός ξεχωριστά, και δίνοντας τη δυνατότητα για διαχείριση τεράστιου όγκου δεδομένων. Ορισμένες NoSQL βάσεις μάλιστα, παρέχουν αυτόματο διαμοιρασμό (auto-sharding) των δεδομένων στους κόμβους του συστήματος, ευνοώντας ακόμα περισσότερο τη χρήση κατανεμημένων συστημάτων και μειώνοντας το φόρτο εργασίας του διαχειριστή. Κάτι τέτοιο είναι μεν εφικτό και στα RDBMS με χρήση clusters αλλά με πολύ μεγαλύτερο κόστος διαχείρισης, και με μικρότερη αποτελεσματικότητα.

Στα σχεσιακά συστήματα τα δεδομένα αποθηκεύονται σε πίνακες με αυστηρά καθορισμένη δομή. Οι στήλες – attributes ενός πίνακα καθορίζονται κατά τη δημιουργία του και δε μπορούν να μεταβάλλονται δυναμικά, ανάλογα με το στοιχείο που θέλουμε να εισάγουμε. Στις NoSQL όμως τα δεδομένα ανήκουν σε έγγραφα (εξ' ου και ο όρος docubases που χρησιμοποιούν πολλοί) και κάθε βάση είναι μια συλλογή από αυτά τα έγγραφα. Έτσι, κάθε έγγραφο μπορεί να έχει εντελώς ανεξάρτητη δομή από τα υπόλοιπα. Στην πραγματικότητα βέβαια, είναι προτιμότερο να υπάρχει μια στοιχειώδης δομή , ώστε να διευκολύνονται τα queries, αλλά ακόμα και έτσι αποφεύγεται η δέσμευση extra χώρου για null values που αποτελεί σημαντικό παράγοντα στις μέρες μας.

Ένα άλλο χαρακτηριστικό των RDBMS που ανατρέπεται από τις NoSQL είναι το μοντέλο ACID (Atomicity – Consistency – Isolation - Durability). Σε μια σχεσιακή βάση, κάθε συναλλαγή (transaction) είτε θα ολοκληρωθεί επιτυχώς είτε θα αποτύχει πλήρως αφήνοντας τη βάση ανεπηρέαστη. Δεν υπάρχει μερική επιτυχία (Atomicity). Ακόμα, μόνο έγκυρα δεδομένα γράφονται στη βάση, φέρνοντας την από μια έγκυρη κατάσταση (να πληρούνται δηλαδή όλα τα constraints) στην επόμενη (consistency), ενώ συναλλαγές που εκτελούνται ταυτόχρονα δεν επηρεάζουν η μία την άλλη, επιφέροντας έτσι ίδιο αποτέλεσμα με τη σειριακή εκτέλεση τους (isolation). Τέλος, αφού ολοκληρωθεί με επιτυχία μια συναλλαγή, παραμένουν οι αλλαγές που έχει κάνει στη βάση δεδομένων, ακόμα και αν το σύστημα έχει πρόβλημα (durability). Στις NoSQL βάσεις η συνέπεια και η απομόνωση θυσιάζονται για χάρη της διαθεσιμότητας, της ανοχής σε σφάλματα και της απόδοσης του συστήματος (availability, graceful degradation and performance). Δημιουργείται έτσι το μοντέλο BASE(Basically Available, Soft State, Eventually Consistent) στο οποίο στηρίζονται οι NoSQL βάσεις. Σύμφωνα με αυτό, ένα σύστημα δουλεύει διαρκώς (basically available), χωρίς αναγκαστικά να είναι διαρκώς συνεπές (soft state), αλλά θα είναι εν τέλει σε κάποια γνωστή κατάσταση (eventually consistent). Οι συναλλαγές επομένως

γίνονται πιο γρήγορα, αφού δεν υπόκεινται στους περιορισμούς του ACID, αλλά με μικρότερη αξιοπιστία και συνέπεια.

Σχήμα 11: ACID vs BASE

NoSQL vs. SQL Summary

	SQL Databases	NoSQL Databases
Types	One type (SQL database) with minor variations	Many different types including key-value stores, document databases, wide-column stores, and graph databases
Development History	Developed in 1970s to deal with first wave of data storage applications	Developed in 2000s to deal with limitations of SQL databases, particularly concerning scale,

	SQL Databases	NoSQL Databases
		replication and unstructured data storage
Examples	MySQL, Postgres, Oracle Database	MongoDB, Cassandra, HBase, Neo4j
Data Storage Model	Individual records (e.g., "employees") are stored as rows in tables, with each column storing a specific piece of data about that record (e.g., "manager," "date hired," etc.), much like a spreadsheet. Separate data types are stored in separate tables, and then joined together when more complex queries are executed. For example, "offices" might be stored in one table, and "employees" in another. When a user wants to find the work address of an employee, the database engine joins the "employee" and "office" tables together to get all the information necessary.	Varies based on database type. For example, key-value stores function similarly to SQL databases, but have only two columns ("key" and "value"), with more complex information sometimes stored within the "value" columns. Document databases do away with the table-and-row model altogether, storing all relevant data together in single "document" in JSON, XML, or another format, which can nest values hierarchically.
Schemas	Structure and data types are fixed in advance. To store information about a new data item, the entire database must be altered, during which time the database must be taken offline.	Typically dynamic. Records can add new information on the fly, and unlike SQL table rows, dissimilar data can be stored together as necessary. For some databases (e.g., wide-column stores), it is somewhat more challenging to add new fields dynamically.
Scaling	Vertically, meaning a single server must be made increasingly powerful in order to deal with increased demand. It is possible to spread SQL databases over many servers, but significant additional engineering is generally required.	Horizontally, meaning that to add capacity, a database administrator can simply add more commodity servers or cloud instances. The database automatically spreads data across servers as necessary
Development Model	Mix of open-source (e.g., Postgres, MySQL) and closed source (e.g., Oracle Database)	Open-source
Supports Transactions	Yes, updates can be configured to complete entirely or not at all	In certain circumstances and at certain levels (e.g., document level vs. database level)

	SQL Databases	NoSQL Databases
Data Manipulation	Specific language using Select, Insert, and Update statements, e.g. SELECT fields FROM table WHERE…	Through object-oriented APIs
Consistency	Can be configured for strong consistency	Depends on product. Some provide strong consistency (e.g., MongoDB) whereas others offer eventual consistency (e.g., Cassandra)

Σχήμα 12: SQL vs NoSQL Summary

2.3 Σύντομη ανάλυση διαφορετικών τύπων NoSQL βάσεων

Μεταξύ των διαφόρων συστημάτων διαχείρισης μη σχεσιακών βάσεων δεδομένων παρατηρούνται διαφορές ως προς τον τρόπο αποθήκευσης και οργάνωσης των δεδομένων τους. Χωρίζονται λοιπόν βάση του μοντέλου δεδομένων τους σε 4 κύριες κατηγορίες: Key-Value Stores, Column Family Stores, Document Databases, Graph Databases.

Στις ***Key-Value Stores*** κυριαρχεί η ιδέα της ύπαρξης ενός hash table όπου ο χρήστης αποθηκεύει δεδομένα-τιμές(values) τα οποία δεικτοδοτούνται από κάποιο μοναδικό κλειδί (key). Δεν είναι απαραίτητη η ύπαρξη ενός αυστηρού σχήματος για τα δεδομένα που αποθηκεύονται, ενώ για την επίτευξη μεγαλύτερης κλιμακωσιμότητας και απόδοσης του συστήματος γίνεται χρήση μηχανισμών caching και δεν υποστηρίζονται συνενώσεις (joins) και συναθροιστικές (aggregate) λειτουργίες. Τα κυριότερα τέτοια συστήματα είναι τα Amazon Dynamo, Riak, Project Voldemort, Redis κ.α.

Στις ***Column Family Stores*** υπερισχύει η ιδέα της αποθήκευσης και επεξεργασίας δεδομένων κατά στήλη και όχι κατά γραμμή. Κύριος στόχος τους είναι να μπορούν να

διαχειριστούν πολύ μεγάλο όγκο δεδομένων που βρίσκονται κατανεμημένα σε διάφορους εξυπηρετητές. Και εδώ γίνεται χρήση κλειδιών (rows), που δείχνουν όμως σε μία ή περισσότερες Column Families. Τυπικά παραδείγματα τέτοιων συστημάτων είναι τα BigTable, Hbase, Accumulo κ.α.

Οι ***Document Databases*** μοιάζουν με τις Key-/Value Stores αλλά υποστηρίζουν πιο πολύπλοκα δεδομένα. Τα δεδομένα αυτά ("έγγραφα") δεν έχουν αυστηρή δομή και έτσι οι χρήστες έχουν τη δυνατότητα να προσθέτουν και να αφαιρούν πεδία κατά βούληση. Όπως και στις Key-/Value Stores, κάθε έγγραφο δεικτοδοτείται από ένα κλειδί (key) με το οποίο μπορεί ο χρήστης να το ανακτήσει, και επιπλέον, στις περισσότερες document databases υπάρχει διαθέσιμο API ή query language ώστε να διευκολύνεται η ανάκτηση εγγράφων με βάση το περιεχόμενό τους. Οι πιο δημοφιλείς document databases είναι οι: MongoDB, CouchDB, Cassandra κ.α.

Στις ***Graph Databases*** γίνεται χρήση κόμβων (nodes) και ακμών (edges) για την αναπαράσταση και αποθήκευση των δεδομένων. Οι γράφοι δηλαδή αντικαθιστούν το κλασσικό μοντέλο των πινάκων. Το μοντέλο γραφημάτων που ακολουθείται ευνοεί την οριζόντια κλιμάκωση αφού μπορεί να χρησιμοποιηθεί παράλληλα σε πολλούς εξυπηρετητές.

2.4 Παραδείγματα και πλεονεκτήματα μερικών βάσεων NoSQL

Στην ενότητα αυτή θα εξετάσουμε μία αντιπροσωπευτική NoSQL βάση από καθέναν από τους 4 παραπάνω τύπους ώστε να γίνει φανερή η διαφορά τους από τα παραδοσιακά RDBMS, αλλά και μεταξύ τους. Θα δούμε σε μεγαλύτερο βάθος τη MongoDB, αφού αυτή είναι η βάση που χρησιμοποιούμε στην εργασία μας.

2.4.1 MongoDB

Η MongoDB (από το "humongous" - τεράστιος) είναι η πιο διαδεδομένη Document Database και ίσως και η πιο διαδεδομένη NoSQL Database γενικότερα. Αναπτύχθηκε από την εταιρεία 10gen στα πρότυπα του ανοιχτού κώδικα και δόθηκε για πρώτη φορά σε κυκλοφορία το 2009. Είναι υλοποιημένη στη γλώσσα προγραμματισμού C++, και κύριος στόχος της είναι η επίτευξη υψηλής ταχύτητας και κλιμακωσιμότητας. Χρησιμοποιείται στο back-end πολλών μεγάλων υπηρεσιών όπως Craigslist, eBay, Foursquare, SourceForge κ.ά., ακριβώς λόγω της πολύ υψηλής δυνατότητας για κλιμάκωση που προσφέρει. Ο προκαθορισμένος τρόπος για τη διαχείριση MongoDB βάσεων είναι μέσω του διαδραστικού Javascript shell που παρέχει. Επιπλέον, σε πολλές γλώσσες προγραμματισμού (C, C#, C++, Haskell, Java, JavaScript, Perl, PHP, Python, και Ruby) έχουν αναπτυχθεί βιβλιοθήκες και drivers για την υποστήριξή της.

Μοντέλο δεδομένων

Στην MongoDB ακολουθείται η νοοτροπία data as documents.Τα έγγραφα που αποθηκεύονται είναι στη μορφή BSON αντικειμένων, δηλαδή δυαδικά κωδικοποιημένα JSON αντικείμενα. Τα BSON είναι σχεδιασμένα έτσι ώστε να γίνεται εύκολα η διάσχιση και η ανάλυσή τους. Οι χρήστες εισάγουν τα δεδομένα τους σε μορφή JSON, τα οποία στη συνέχεια μετατρέπονται σε BSON ώστε να αποθηκευτούν στη βάση, και αντίστοιχα όταν παίρνουν κάποιο BSON αντικείμενο από τη βάση, το βλέπουν σε JSON μορφή. Δηλαδή, οι χρήστες δε βλέπουν καθόλου τη BSON μορφή, χρησιμοποιείται μόνο εσωτερικά στη βάση. Ένα JSON document είναι μηδέν ή περισσότερα ζεύγη κλειδιού – τιμής (key-value pairs). Σαν κλειδί αποθηκεύεται το όνομα του πεδίου ως συμβολοσειρά και σαν τιμή, μια τιμή ενός από τους υποστηριζόμενους JSON data types (Number, String, Boolean, Array, Object, Whitespace, null, ή JSON value) είτε κάποιος άλλος όπως ημερομηνία. Το μέγιστο μέγεθος ενός εγγράφου είναι 16MB, ενώ για μεγαλύτερα αρχεία χρησιμοποιείται το GridFS. Τα αντικείμενα αυτά είναι schemaless, που σημαίνει πως δεν υπάρχει κάποια ομαδοποίηση μεταξύ εγγράφων που περιέχουν ίδια κλειδιά, κάτι που συναντάμε στο σχεσιακό μοντέλο, όπου όλες οι εγγραφές ενός πίνακα έχουν την ίδια δομή. Αντιθέτως, έγγραφα με παρόμοια δομή που περιλαμβάνουν δεδομένα σχετικά με το ίδιο αντικείμενο, αλλά με διαφορετικά ζεύγη κλειδιού – τιμής, αποθηκεύονται μαζί, σε μια "συλλογή".

Το πλεονέκτημα των NoSQL που έχει να κάνει με τη μη ύπαρξη αυστηρά καθορισμένου σχήματος των δεδομένων, φαίνεται και στη MongoDB αφού υπάρχει απόλυτη ελευθερία ως προς τα πεδία, τη δομή και τους τύπους δεδομένων κάθε εγγράφου μιας συλλογής. Από την άλλη όμως, ένα πολύ βασικό στοιχείο είναι πως δεν υπάρχει δυνατότητα για joins μεταξύ συλλογών όπως στις σχεσιακές βάσεις

δεδομένων. Έτσι, ο σχεδιαστής καλείται να διαμορφώσει το κάθε έγγραφο ξεχωριστά, με τέτοιο τρόπο ώστε να ελαχιστοποιεί τον απαιτούμενο αποθηκευτικό χώρο αλλά και να διευκολύνει τις αναζητήσεις του, μοντελοποιώντας τα δεδομένα με γνώμονα τον τρόπο χρήσης τους και όχι τον τρόπο αποθήκευσης. Παρακάτω φαίνεται ένα παράδειγμα JSON αρχείου.

```
{
  "id": "1",
  "name" : { "first" : "Georgios", "last" : "Tentes" },
  "education" : {
         "school" : "National Technical University of Athens",
         "department" : "Electrical and Computer Engineering",
         "graduated"  : true
              }
  "thesis" :   {
         "title" : "...",
         "date" : ...,
         "supervisor" : "Prof. Theodora Varvarigou"
         "year" : 2014,
         "tags" : [ "cloud computing", "NoSQL", "Big Data", "
            Multimedia" ]
       }
}
```

Σχήμα 13: Json

Indexing

Για την υποστήριξη γρήγορων ερωτημάτων, ο σχεδιαστής μπορεί να δημιουργήσει ένα ευρετήριο για ένα πεδίο ενός εγγράφου που θα δεχθεί ερωτήσεις. Η MongoDB επιτρέπει τη δημιουργία ευρετηρίων σε οποιοδήποτε στοιχείο ενός εγγράφου, ακριβώς όπως και τα RDBMS επιτρέπουν ευρετηριοποίηση σε οποιαδήποτε στήλη ενός πίνακα. Τα ευρετήρια υλοποιούνται ως B-Δένδρα και παρότι η διατήρηση πολλών ευρετηρίων δημιουργεί καθυστερήσεις στις ενημερώσεις, εάν εφαρμοστούν σωστά, μπορούν να αυξήσουν κατακόρυφα την απόδοση των queries. Μια καλή πρακτική για γρήγορη εκτέλεση των ερωτημάτων είναι να διατηρούνται τα ευρετήρια στη RAM. Σε κάθε ερώτημα χρησιμοποιείται ένα μόνο ευρετήριο, η επιλογή του οποίου γίνεται από το βελτιστοποιητή ερωτημάτων (query optimizer). Τα ευρετήρια

δημιουργούνται μέσα από το shell της Mongo με χρήση της συνάρτησης ensureIndex(). Μπορούμε να δημιουργήσουμε ευρετήρια σε απλά κλειδιά, ενσωματωμένα κλειδιά ή ακόμα και ολόκληρα ενσωματωμένα αντικείμενα.

Αρχιτεκτονική

Η MongoDB έχει ως κύριο στόχο την υποστήριξη εφαρμογών που τρέχουν σε κατανεμημένα συστήματα – clusters υπολογιστών. Έτσι, παίζει πολύ μεγάλο ρόλο η οριζόντια κλιμακωσιμότητα και για αυτό πραγματοποιείται αυτόματη κατανομή του φόρτου εργασίας (auto sharding) μεταξύ των κόμβων του cluster. Οι κόμβοι αυτοί διακρίνονται σε 3 κατηγορίες: shard nodes, configuration nodes και routers.

Shard Nodes

Αποτελούνται από εξυπηρετητές που τρέχουν mongod διεργασίες και αποθηκεύουν δεδομένα. Για να εξασφαλιστεί η διαθεσιμότητα η αυτόματη ανακατεύθυνση ενός μη διαθέσιμου κόμβου, κάθε shard συνήθως αποτελείται από πολλούς εξυπηρετητές που δημιουργούν ένα replica set. (αντίγραφο των ίδιων δεδομένων).

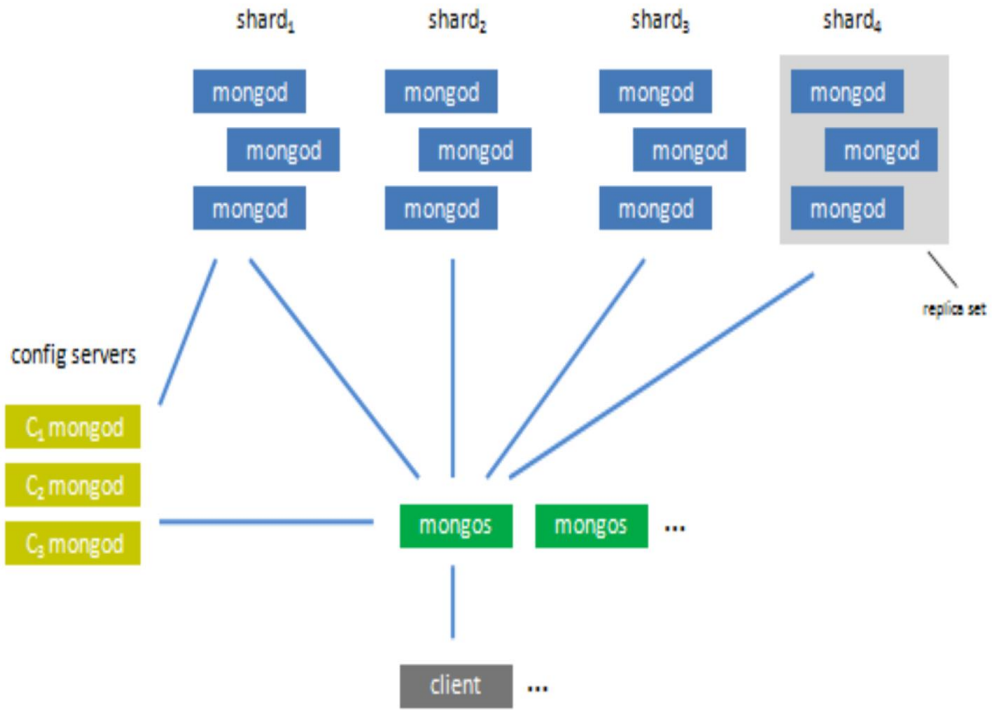
Configuration Servers

Αποθηκεύουν τα metadata (μετα-πληροφορίες) του cluster, που περιλαμβάνουν βασικές πληροφορίες για κάθε shard node και τα "chunks" που περιέχει ο καθένας. Τα chunks είναι συνεχόμενα δεδομένα από συλλογές ταξινομημένες με βάση το sharding key και που αποθηκεύονται στα shards. Κάθε configuration server κρατάει τα metadata των chunks και έτσι μπορεί να διαπιστώσει σε ποιο shard βρίσκεται κάθε ζητούμενο document. Η συνέπεια των δεδομένων στους configuration servers εξασφαλίζεται μέσω ενός two - phase commit πρωτοκόλλου και ενός ειδικού replication σχήματος (όχι του Master - Slave ή των replication sets όπως στα shards).

Routing Services

Είναι διεργασίες mongos που εκτελούν αιτήματα ανάγνωσης και εγγραφής εκ μέρους των εφαρμογών πελάτη και τρέχουν στην πλευρά του εξυπηρετητή. Είναι υπεύθυνες για την εύρεση των shards που θα διαβαστούν ή θα εγγραφούν δια μέσου των configuration servers, για τη σύνδεση με αυτά τα shards, την εκτέλεση του αντίστοιχου αιτήματος και την επιστροφή του αποτελέσματος στην εφαρμογή του πελάτη, συγχωνεύοντας το ενδεχομένως με τα αποτελέσματα του ίδιου αιτήματος από άλλα shards. Έτσι, επιτυγχάνεται η προβολή στους πελάτες ενός κατανεμημένου MongoDB συστήματος ως ένας απλός εξυπηρετητής, καθώς οι πελάτες δε χρειάζεται να γνωρίζουν το παραμικρό σχετικά με το πώς ή το πού είναι κατανεμημένα τα

ζητούμενα έγγραφα. Οι mongos διεργασίες δε διατηρούν καμία κατάσταση και δε συντονίζονται μεταξύ τους. Αντιθέτως είναι αρκετά "ελαφριές" ώστε να μπορούν να υποστηριχθεί μεγάλος αριθμός από requests.

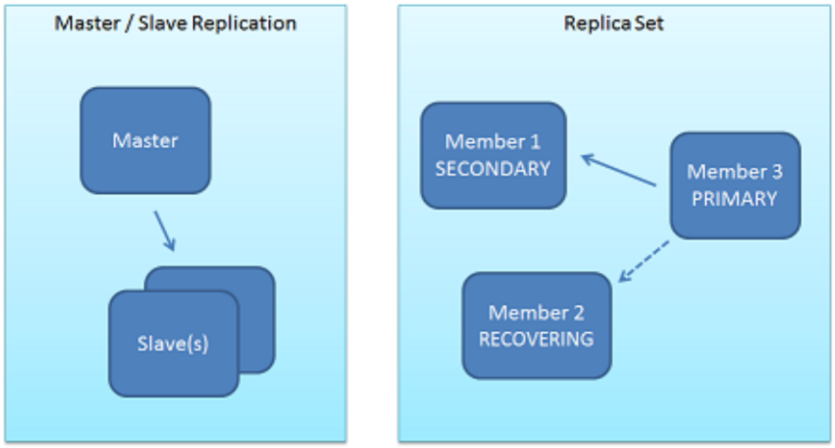


Σχήμα 14: MongoDB Architecture

Replication

Η MongoDB παρέχει έμφυτη υποστήριξη για τη διατήρηση αντιγράφων (replication), που σημαίνει πως η βάση κλωνοποιείται και συγχρονίζεται σε τουλάχιστον 2 υπολογιστές. Με τη φύλαξη αντιγράφων διασφαλίζεται αυτόματη εναλλακτική σύνδεση σε περίπτωση σφάλματος και έτσι εξασφαλίζεται η διαθεσιμότητα του συστήματος. Ο απλούστερος μηχανισμός φύλαξης αντιγράφων είναι αυτός του Single Master/ Single Slave, δηλαδή χρήση ενός πρωτεύοντος και ενός δευτερεύοντος εξυπηρετητών. Στη MongoDB ωστόσο, συνήθως χρησιμοποιούμε replica sets κάθε ένα από τα οποία είναι ένας shard node που αποτελείται από έναν πρωτεύον και κάποιους δευτερεύοντες κόμβους. Ένα replica set μπορεί να αποτελείται από έως και 7 servers ενώ δίνεται και στο χρήστη η

δυνατότητα να μην γίνει replication κάποιων αρχείων του, εισάγοντάς τα στην ειδική βάση local.



Σχήμα 15: MongoDB Replication Approaches

Sharding

Από την έκδοση 1.6 και έπειτα, η MongoDB υποστηρίζει την οριζόντια κλιμάκωση μέσω μιας αυτόματης αρχιτεκτονικής καταμερισμού της βάσης δεδομένων ώστε να κατανέμει τα δεδομένα μεταξύ των διαθέσιμων κόμβων, με αυτόματο καταμερισμό φόρτου εργασίας και όγκου δεδομένων και ανακατεύθυνση σε περίπτωση σφάλματος (failover). Σύμφωνα με το documentation της MongoDB, ως sharding ορίζεται ο διαχωρισμός των δεδομένων μεταξύ πολλαπλών υπολογιστών με τέτοιο τρόπο ώστε να διατηρείται η σχετική τους σειρά. Το sharding γίνεται με βάση τη συλλογή (collection) και όχι συνολικά τη βάση δεδομένων. Ένα MongoDB σύστημα, μπορεί να διακρίνει αυτόματα ποιες από τις συλλογές μεγαλώνουν με μεγαλύτερο ρυθμό από τις υπόλοιπες, ώστε να τις καταμερίσει σε περισσότερους κόμβους, ενώ οι άλλες μπορούν να παραμείνουν σε έναν μόνο κόμβο. Ακόμα, μπορεί να εντοπίσει τις ανισορροπίες μεταξύ του φορτίου των shards, και να ανακατανείμει τα δεδομένα ώστε αυτές να εξομαλυνθούν (auto-sharding).

Queries

Τα ερωτήματα στη MongoDB υλοποιούνται ως query objects, δηλαδή αντικείμενατης μορφής BSON, που περιέχουν τα κριτήρια του ερωτήματος. Έτσι, δημιουργούνται είτε μέσω του interactive shell, είτε μέσω μιας γλώσσας προγραμματισμού, τα αρχεία στη μορφή JSON με βάση τις παραμέτρους που εισάγει

ο χρήστης, και αυτά στέλνονται σαν BSON αντικείμενα μέσω του driver της εφαρμογής, ώστε να εκτελεστεί το query. Ένα ερώτημα εκτελείται για μια συγκεκριμένη συλλογή. Εάν ο χρήστης θέλει να συμπεριλάβει αποτελέσματα από περισσότερες συλλογές, θα πρέπει να εφαρμόσει το ερώτημα σε όλες τις συλλογές που πιθανόν να περιέχουν τα αντίστοιχα έγγραφα. Το μοντέλο ερωτημάτων της MongoDB παρέχει αρκετά μεγάλη λειτουργικότητα αφού υποστηρίζει μεταξύ άλλων τα ακόλουθα χαρακτηριστικά:

- Υποβολή ερωτημάτων σε documents και τα embedded subdocuments
- Υποβολή γεωχωρικών ερωτημάτων
- Χρήση τελεστών σύγκρισης (>, ≥, < , ≤ , =)
- Χρήση λογικών τελεστών (and, nor, not, or)
- Χρήση τελεστών ικανοποίησης συνθήκης (in, exists, equals, mod, type, all κ.α.)
- Χρήση συναρτήσεων ομαδοποίησης (count, sum κλπ)

Φαίνεται λοιπόν πως αντίθετα με πολλές άλλες NoSQL βάσεις δεδομένων, η MongoDB υποστηρίζει εξειδικευμένα ερωτήματα. Για την αποδοτική εκτέλεση αυτών των ερωτημάτων, δημιουργούνται πλάνα ερωτημάτων από ένα συστατικό της MongoDB που ονομάζεται query optimizer (βελτιστοποιητής ερωτημάτων). Σε αντίθεση με αντίστοιχα συστατικά των RDBMS, δε βασίζεται σε στατιστικά στοιχεία ούτε υπολογίζει το κόστος πολλαπλών δυνατών πλάνων ερωτημάτων. Αντίθετα, εκτελεί τα διάφορα πλάνα ερωτημάτων παράλληλα, και τα διακόπτει όλα, όταν ένα από αυτά επιστρέψει το ζητούμενο αποτέλεσμα, συμπεραίνοντας έτσι ποιο είναι το αποδοτικότερο query plan για το αντίστοιχο ερώτημα. Η προσέγγιση αυτή λειτουργεί αποδοτικά στα μη σχεσιακά συστήματα, αφού λόγω της μη ύπαρξης joins, ελαχιστοποιείται ο χώρος των δυνατών πλάνων ερωτημάτων.

Σχετικά με τις συναλλαγές (transactions), η MongoDB παρέχει ατομικότητα μόνο σε λειτουργίες delete και update, με χρήση της αντίστοιχης παραμέτρου($atomic).
Το κλείδωμα συναλλαγών και οι πολύπλοκες συναλλαγές δεν επιτρέπονται για τους εξής λόγους:

- Απόδοση. Αφού σε μια διαμερισμένη βάση τα κατανεμημένα κλειδώματα θα ήταν πολύ “βαριά¨” και αργά, ενώ αντίθετα η MongoDB έχει σχεδιαστεί με άξονα την ταχύτητα και την “ελαφρότητα”
- Αποφυγή αδιεξόδων
- Διατήρηση των απλών και προβλέψιμων λειτουργιών στη βάση
- Στόχευση στην καλή λειτουργία της βάσης για προβλήματα πραγματικού χρόνου.

Συνεπώς το κλείδωμα μεγάλου όγκου δεδομένων μπορεί να οδηγούσε σε μεγάλη καθυστέρηση της εκτέλεσης ελαφρύτερων ερωτημάτων.

Τέλος, η MongoDB παρέχει και μια ελαφρώς παραλλαγμένη υλοποίηση του

MapReduce (το οποίο αναλύεται σε βάθος παρακάτω). Στην υλοποίηση αυτή, η συνάρτηση map δέχεται ως είσοδο ένα document, το επεξεργάζεται και παράγει ζεύγη κλειδιού - τιμής. Τα ζεύγη αυτά ομαδοποιούνται με βάση το κλειδί, και για κάθε ένα προκύπτει ένας πίνακας από τιμές. Αυτά τα ζεύγη κλειδιού – πίνακα τιμών προωθούνται στη συνάρτηση reduce η οποία παράγει και επιστρέφει για κάθε κλειδί μία μοναδική τιμή, ίδιου τύπου με την τιμή που κάνει emit η map function. Αυτός ο περιορισμός υπάρχει επειδή οι συναρτήσεις reduce μπορεί να κληθούν επαναληπτικά. Προαιρετικά, μπορεί να γίνει χρήση της συνάρτησης finalize. Η συνάρτηση αυτή εκτελείται μία φορά για κάθε key, μετά τη φάση του reduce και χρησιμοποιείται για την ομαδοποίηση των αποτελεσμάτων της reduce function, όπως πχ για τον υπολογισμό ενός μέσου όρου.

2.4.2 HBase

Η Hbase είναι μια column-oriented βάση δεδομένων που αναπτύχθηκε ως μια ανοιχτού κώδικα έκδοση της BigTable. Τα column-oriented συστήματα αποθήκευσης τυγχάνουν αυξανόμενης προσοχής τα τελευταία χρόνια τόσο σε ερευνητικό όσο και σε βιομηχανικό επίπεδο, λόγω της υποστήριξης που προσφέρουν για μεγάλη κλιμακωσιμότητα των δεδομένων, αποδοτικής πρόσβασης στα δεδομένα και υψηλής ανοχής σφαλμάτων. Τα δεδομένα οργανώνονται εσωτερικά στη βάση ως εμφωλευμένα ζεύγη κλειδιού-τιμής και παρουσιάζονται εξωτερικά στο χρήστη ως αραιοί πίνακες. Κάθε γραμμή σε αυτούς τους πίνακες αντιστοιχεί σε ένα σύνολο εμφωλευμένων ζευγών κλειδιού-τιμής που δεικτοδοτούνται σε ανώτερο επίπεδο από το ίδιο κλειδί (το κλειδί της γραμμής, “row key”). Σε δεύτερο επίπεδο το κλειδί είναι το “column family” και σε τρίτο το “column qualifier”). Κάθε στήλη σε μια γραμμή αντιστοιχεί στην τιμή (αποθηκευμένη ως πίνακας από bytes) που δεικτοδοτείται από το συνδυασμό του column family και του column qualifier.

Στην Hbase τα δεδομένα αποθηκεύονται σε αραιούς πίνακες, δηλαδή πίνακες με γραμμές που έχουν μεταβλητό αριθμό στηλών. Κάθε γραμμή έχει ένα μοναδικό row key με βάση το οποίο ταξινομούνται αυτόματα. Οι στήλες ομαδοποιούνται σε οικογένειες στηλών. Τα δεδομένα της ίδιας οικογένειας στηλών αποθηκεύονται σε φυσικό επίπεδο, κοντά στο δίσκο ώστε να επιτυγχάνεται βελτίωση της απόδοσης των queries. Για την υποστήριξη πολλαπλών εκδόσεων των δεδομένων χρησιμοποιούνται timestamps. Αυτές ορίζονται άμεσα από το χρήστη όταν εισάγει τα δεδομένα, είτε αυτόματα από το σύστημα, και σε συνδυασμό με το row key και τη στήλη, προσδιορίζουν μοναδικά τα δεδομένα. Ένα παράδειγμα πίνακα της Hbase φαίνεται παρακάτω:

Row Key	**Time Stamp**	**ColumnFamily `contents`**	**ColumnFamily `anchor`**
"com.cnn.www"	t9		`anchor:cnnsi.com` = "CNN"
"com.cnn.www"	t8		`anchor:my.look.ca` = "CNN.com"
"com.cnn.www"	t6	`contents:html` = "<html>..."	
"com.cnn.www"	t5	`contents:html` = "<html>..."	
"com.cnn.www"	t3	`contents:html` = "<html>..."	

Βλέπουμε λοιπόν πως μπορούμε να έχουμε δύο γραμμές με το ίδιο row key, ακόμα και στην ίδια column family, αλλά όχι και με ίδιο timestamp, αφού όπως είπαμε ο συνδυασμός αυτών των τριών προσδιορίζει μοναδικά τα δεδομένα.

Η οργάνωση αυτή βοηθάει στην αποδοτική ανάκτηση και ενημέρωση των δεδομένων σε πραγματικό χρόνο. Εξαιτίας της μορφής των πινάκων μπορούμε να συμπιέσουμε τα δεδομένα κερδίζοντας πολύτιμο αποθηκευτικό χώρο.

Στις column oriented βάσεις η επεκτασιμότητα επιτυγχάνεται με το range-partitioning που γίνεται στα δεδομένα με βάση το row key, σε τμήματα ίσου μεγέθους που δε μοιράζονται καθόλου δεδομένα. Αυτά τα τμήματα (partitions) που προκύπτουν, αποστέλλονται σε διάφορους κατανεμημένους εξυπηρετητές. Όσο αυξάνεται ο όγκος των δεδομένων, τόσο περισσότερα partitions δημιουργούνται. Θεωρητικά λοιπόν, η ικανότητα αποθήκευσης των δεδομένων κλιμακώνεται τόσο όσο αυξάνονται οι διαθέσιμοι εξυπηρετητές.

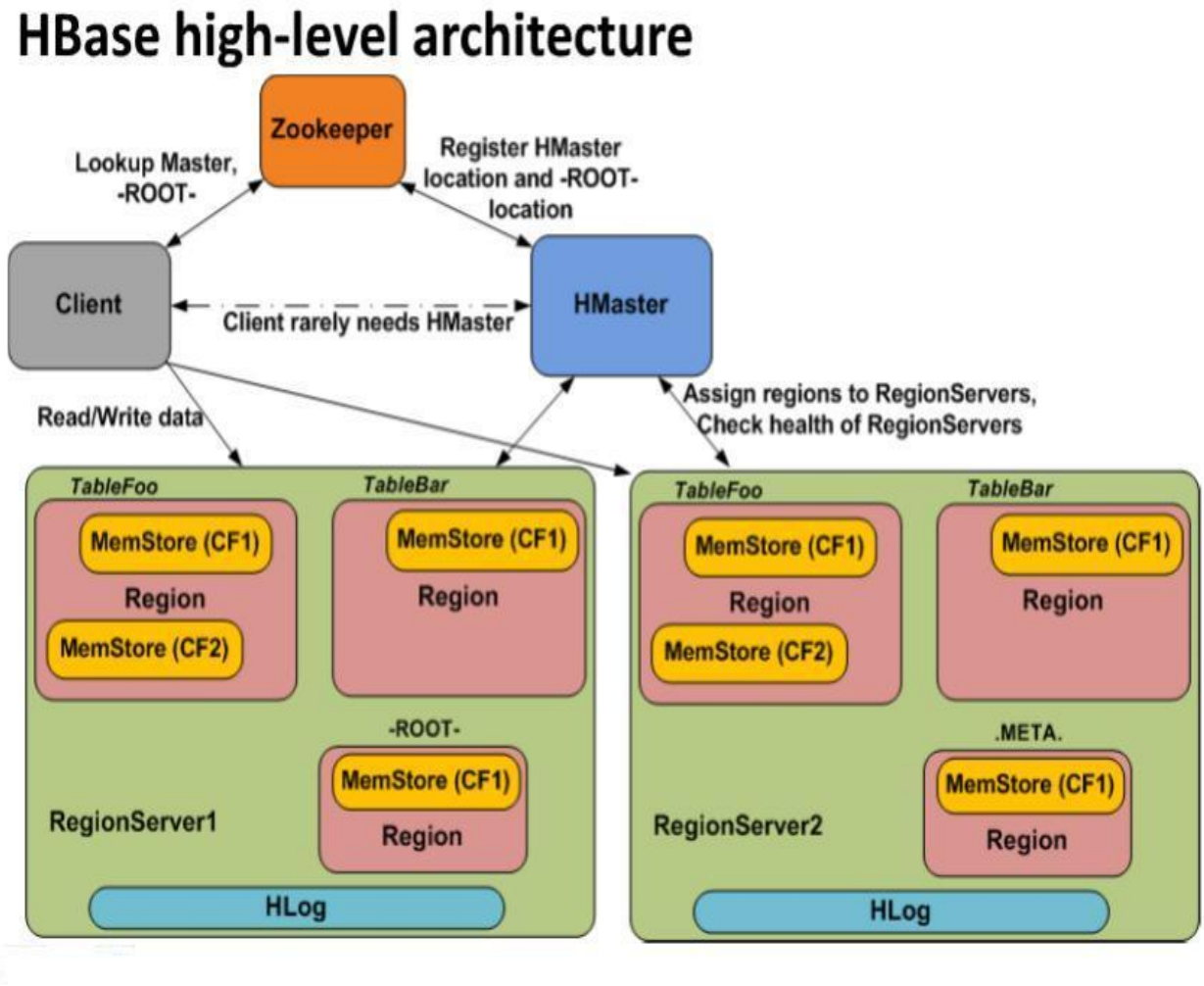
Η HBase χρησιμοποιεί μια master-slave τοπολογία. Τα partitions λοιπόν που εδώ ονομάζονται “regions” αποθηκεύονται στους slaves, οι οποίοι ονομάζονται “region servers”. Χρησιμοποιούνται πολλαπλοί masters ώστε να αποφεύγεται το πρόβλημα του single point of failure. Όταν ένας region server τίθεται εκτός λειτουργίας, τα δεδομένα του ανακτώνται από το file system (HDFS) και αποστέλλονται σε άλλον εξυπηρετητή. Μια πιθανή συμφόρηση που μπορεί να δημιουργηθεί όμως λόγω αυτής της αρχιτεκτονικής είναι όταν έχουμε πολλαπλά αιτήματα για δεδομένα στο ίδιο εύρος row key. Έτσι, όλα αυτά τα αιτήματα πάνε στον ίδιο εξυπηρετητή και πιθανόν να έχουμε υπερφόρτωση.

Επί του παρόντος, η Hbase υποστηρίζει απλά queries με βάση το row key και τα timestamps, δεν υποστηρίζει όμως πολύπλοκα ερωτήματα, join λειτουργίες, συναλλαγές και δευτερεύοντες δείκτες. Μπορούμε επιπλέον να ανακτήσουμε και να

διατρέξουμε ένα σύνολο από στήλες, σε ένα δεδομένο εύρος γραμμών.

Αν και το query model φαίνεται αρκετά απλοϊκό και πιθανόν ανεπαρκές, αν οι πίνακες δομηθούν σωστά, επιτρέπουν την επίλυση αρκετών σύνθετων προβλημάτων, ενώ για ακόμα πιο πολύπλοκες λειτουργίες μπορούμε να χρησιμοποιήσουμε εργασίες map – reduce.

Παρακάτω φαίνεται πιο αναλυτικά η αρχιτεκτονική της Hbase:



Σχήμα 16: Hbase Architecture

2.4.3 Amazon's Dynamo DB

Η Dynamo DB αναπτύχθηκε από την Amazon βασισμένη στις αρχές του Dynamo (προγόνου του NoSQL) και προσφέρεται για χρήση στο πακέτο του Amazon Web Services. Ως προς την τεχνολογική υποδομή στην οποία στηρίζεται υποθέτει μεταξύ άλλων πως απαρτίζεται από δεκάδες χιλιάδες εξυπηρετητές που βρίσκονται σε πολλά datacenters ανά τον κόσμο, χρησιμοποιείται χαμηλού κόστους υλικό και πως το σφάλμα κάποιου στοιχείου είναι η συνηθισμένη κατάσταση λειτουργίας. Στην Amazon η Dynamo χρησιμοποιείται για να διαχειρίζεται την κατάσταση υπηρεσιών με πολύ υψηλές απαιτήσεις σε αξιοπιστία και χρειάζονται αυστηρό έλεγχό ως προς το συμβιβασμό μεταξύ διαθεσιμότητας, συνέπειας, χαμηλού κόστους και απόδοσης.

Η Dynamo χρησιμοποιεί consistent hashing σε συνδυασμό με replication ως σχήμα διαμερισμού των δεδομένων. Επιπλέον, για την αντιμετώπιση των δύο κυριότερων μειονεκτημάτων του consistent hashing (άνισα κατανομή δεδομένων και αδιαφορία ως προς τις δυνατότητες του κάθε κόμβου), γίνεται χρήση εικονικών κόμβων. Τα αντικείμενα που αποθηκεύονται σε τμήματα μεταξύ των κόμβων έχουν συγκεκριμένη έκδοση το καθένα (multi-version storage). Για τη διατήρηση της συνέπειας κατά τη διάρκεια ενημερώσεων, η Dynamo χρησιμοποιεί μια τεχνική απαρτίας και ένα πρωτόκολλο για αποκεντρωμένο συγχρονισμό των αντιγράφων. Για τον εντοπισμό αστοχιών μεταξύ των κόμβων, χρησιμοποιείται ένα gossip-based πρωτόκολλο που διευκολύνει κατά πολύ την εισαγωγή και απομάκρυνση εξυπηρετητών με ελάχιστη ανθρώπινη ευθύνη.

Μία από τις κεντρικές ιδέες πάνω στις οποίες βασίζεται η Dynamo είναι πως οι περισσότερες υπηρεσίες στην Amazon μόνο αποθηκεύουν και ανακτούν δεδομένα με βάση το πρωτεύον κλειδί και συνεπώς είναι αχρείαστα τα πολύπλοκα ερωτήματα και η διαχειριστική λειτουργικότητα που προσφέρεται από τα RDBMS. Η Dynamo χρησιμοποιεί ένα απλό key-value interface, αποθηκεύοντας τις τιμές σαν BLOBs (Binary Large Objects). Οι λειτουργίες περιορίζονται σε ένα ζεύγος κλειδιού-τιμής κάθε φορά.

Η Dynamo στοχεύει κυρίως στην επίτευξη υψηλής διαθεσιμότητας και επεκτασιμότητας και για αυτό το λόγο θυσιάζει την αυστηρή συνέπεια και εν μέρει την απομόνωση. Αυτό φαίνεται από την πολιτική ακολουθείται στα writes, τα οποία επιτρέπονται ακόμα και όταν έχουμε αποσυνδεδεμένα αντίγραφα. Έτσι, η Dynamo σχεδιάζεται ως “always writable” και η ενημέρωση των αντιγράφων γίνεται στα reads.

Όλοι οι κόμβοι στην Dynamo είναι ισότιμοι. Δεν υπάρχουν κόμβοι με ειδικές ευθύνες και λειτουργίες όπως πχ master, slave κλπ. Ο σχεδιασμός αυτός ευνοεί τις

αποκεντρωμένες peer-to-peer τεχνικές έναντι του κεντρικού ελέγχου.

Το γεγονός ότι η Dynamo λειτουργεί εντός του δικτύου της Amazon, το περιβάλλον και όλοι οι κόμβοι θεωρούνται μη εχθρικοί και επομένως δεν επιφορτίζεται με την υλοποίηση μηχανισμών ασφαλείας (εξουσιοδότησης, αυθεντικότητας κλπ.).

Το interface της Dynamo παρέχει μόνο δύο λειτουργίες:
get (key) : που επιστρέφει μια λίστα με αντικείμενα και ένα context
put (key, context, object) : χωρίς τιμή επιστροφής. Το get μπορεί να επιστρέψει περισσότερα από ένα αντικείμενα αν υπάρχουν συγκρουόμενες εκδόσεις αντικειμένων με το ίδιο κλειδί. Στο context περιέχονται metadata όπως η έκδοση του αντικειμένου κλπ.

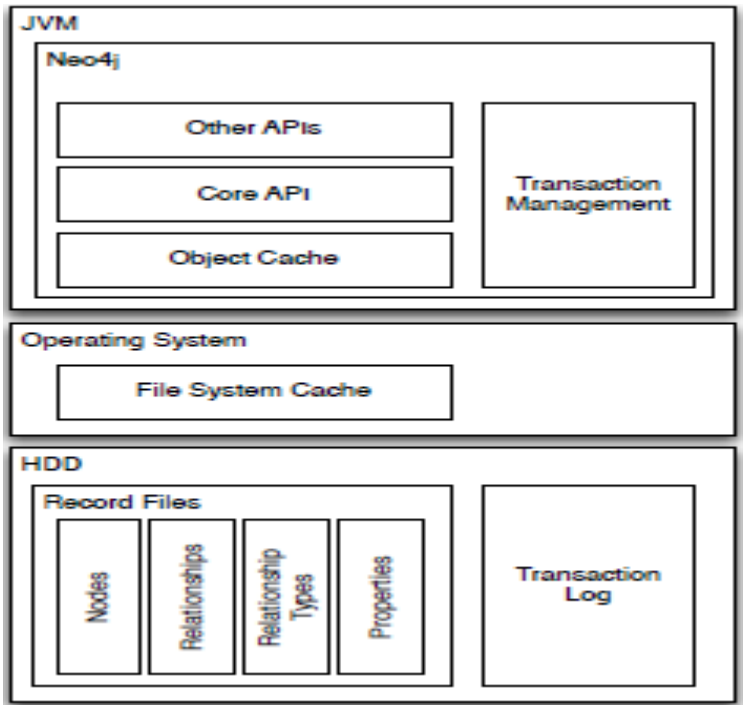
Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

Σχήμα 17: Amazon's Dynamo techniques

2.4.4 Neo4j

Η Neo4j είναι η πιο διαδεδομένη graph database. Είναι ανοιχτού κώδικα, γραμμένη σε java και η πρώτη έκδοσή της δόθηκε στην κυκλοφορία το Φεβρουάριο του 2010 από τη Neo Technology. Χρησιμοποιεί property graphs, δηλαδή ένα σύνολο από κόμβος και κατευθυνόμενες ακμές, όπου κάθε κόμβος και κάθε ακμή μπορεί να έχουν έναν οποιοδήποτε αριθμό από ιδιότητες που χρησιμεύουν ώστε να αποθηκεύονται τα δεδομένα. Οι ιδιότητες αυτές έχουν ένα μοναδικό αναγνωριστικό κλειδί και μια τιμή. Όπως οι περισσότερες NoSQL βάσεις, είναι schema-free που σημαίνει πως οι κόμβοι, οι συσχετίσεις και οι ιδιότητες μπορούν να δημιουργηθούν με οποιοδήποτε τρόπο, με μοναδική προϋπόθεση κάθε συσχέτιση να έχει έναν αρχικό και έναν τελικό κόμβο.

Όσον αφορά την αρχιτεκτονική της βάσης, η Neo4j τρέχει πάνω σε JVM (Java Virtual Machine). Πέρα από τα Java APIs, υπάρχουν διάφορα άλλα APIs για να διευκολύνουν τη διαχείριση και την αποθήκευση των graph data. Χρησιμοποιείται μια object cache εντός του JVM για γρήγορη εύρεση των objects, σε επόμενο επίπεδο μια file system cache και τελικά τα δεδομένα διατηρούνται στο σκληρό δίσκο. Η αρχιτεκτονική αυτή φαίνεται καλύτερα στην παρακάτω εικόνα:



Σχήμα 18: Neo4j Architecture

Οι κόμβοι αποθηκεύονται ως εγγραφές των 9 byte, με τέτοιο τρόπο ώστε να διευκολύνεται η γρήγορη εύρεση τους με βάση το ID. Οι συσχετίσεις αποθηκεύονται ως εγγραφές των 33 με δείκτες προς τον αρχικό και τον τελικό κόμβο τους. Οι ιδιότητες είναι ζεύγη κλειδιού – τιμής των 41 bytes, όπου το κλειδί είναι ένα java string και η τιμή οποιοσδήποτε java primitive type, πίνακας ή string.

Όσον αφορά τις δοσοληψίες, η Neo4j ακολουθεί ένα πολύ αυστηρό μοντέλο, αφού απαιτεί κάθε ενεργεία που πιθανόν να μεταβάλλει τα δεδομένα, να τρέξει ως μια δοσοληψία. Έτσι, καταφέρνει να εξασφαλίσει την ακεραιότητα των δεδομένων και τις ACID ιδιότητες.

ΚΕΦΑΛΑΙΟ 3

Big Data

Ο όρος Big Data τείνει να γίνει ένας από τους πιο δημοφιλείς τα τελευταία χρόνια στις τεχνολογικές επιστήμες. Αρχικά διαμορφώθηκε από μεγάλους οργανισμούς που έπρεπε να χειριστούν δεδομένα με μεγάλο αυξητικό ρυθμό, όπως διαδικτυακά δεδομένα, δεδομένα από επιστημονικές ή επιχειρηματικές προσομοιώσεις κ.ά. Πολλοί από αυτούς τους οργανισμούς στηρίζονται στον αποδοτικό χειρισμό αυτού του μεγάλου όγκου δεδομένων και έτσι οδηγήθηκαν στην ανάπτυξη εργαλείων και εφαρμογών προς αυτή την κατεύθυνση, όπως πχ. η Google με το GFS και το MapReduce.

Κατά μία έννοια, τα big data βρίσκονται παντού και εμπλέκουν τους πάντες. Οι απλοί χρήστες του διαδικτύου και των κοινωνικών δικτύων παράγουν καθημερινά τεράστιο όγκο δεδομένων με τις δραστηριότητές τους (tweets, blog posts, upload φωτογραφιών και βίντεο κλπ). Οι επιστήμονες δημιουργούν λεπτομερείς μετρήσεις αναζητώντας όσο το δυνατόν μεγαλύτερη ακρίβεια για τα πειράματά τους. Οι επιχειρήσεις αποθηκεύουν δεδομένα που έχουν να κάνουν με το ιστορικό των πωλήσεων τους, τα στοιχεία των πελατών τους και τις προτιμήσεις τους ώστε να μπορέσουν να αναλύσουν καλύτερα τις ανάγκες τους. Βλέπουμε λοιπόν πώς προέκυψε η ανάγκη για την ανάπτυξη νέων τεχνολογιών που να μας επιτρέπουν να διαχειριστούμε αυτά τα δεδομένα, κάτι που με τις υπάρχουσες μεθόδους θα ήταν αδύνατο. Έτσι, προκύπτει και ένας αρκετά αντιπροσωπευτικός ορισμός για τα big data που δόθηκε από το McKinsley Global Institute το Μάιο του 2011: “Big Data refers to data sets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze.”

3.1 Volume Velocity Variety

Τα Big Data χαρακτηρίζονται πολύ εύστοχα από το αποκαλούμενο μοντέλο 3V Model, όπου οι όροι Volume, Velocity και Variety έχουν κυρίαρχο ρόλο. Με το

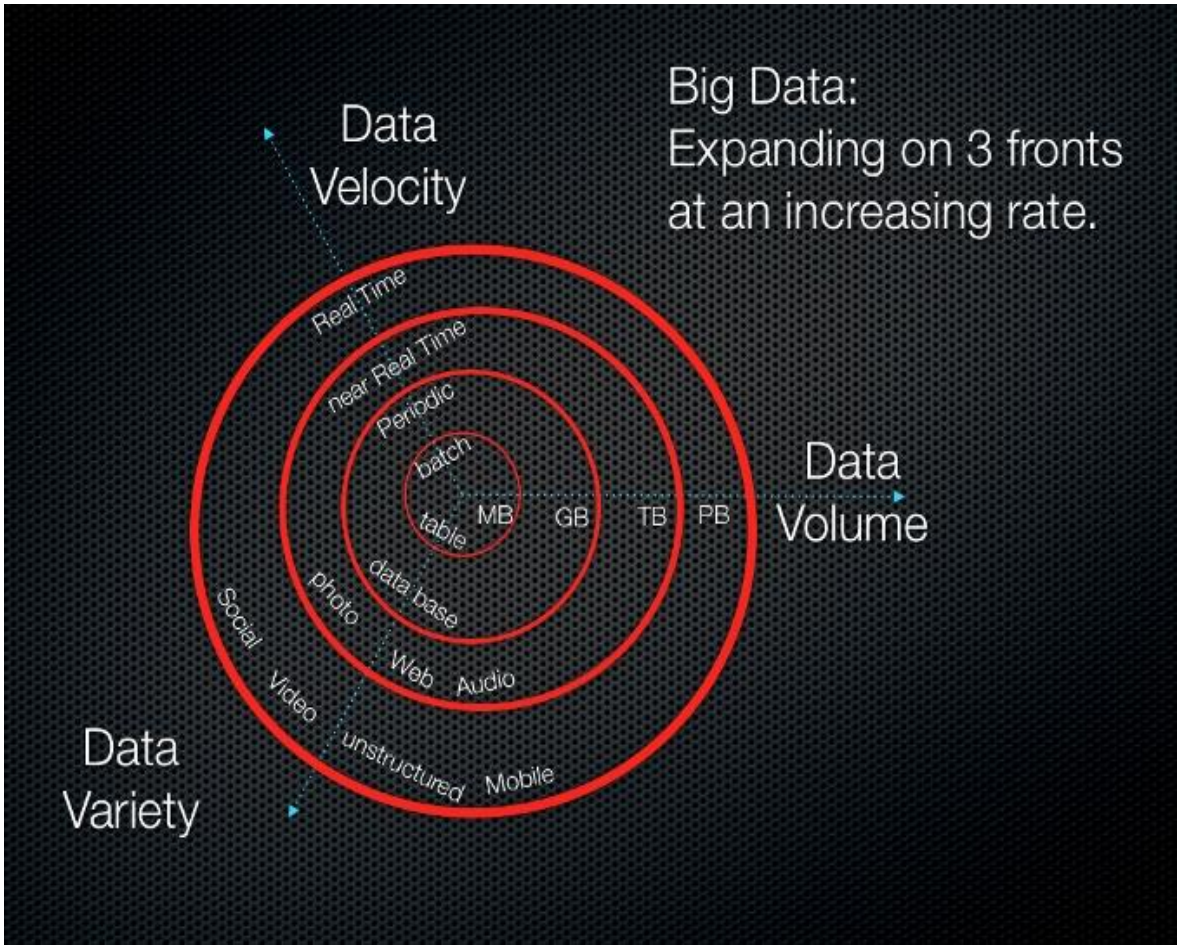
Variety (ποικιλία) εννοούμε το μεγάλο εύρος πιθανών διαφορετικών τύπων δεδομένων που καλούμαστε να χειριστούμε, το Velocity (ταχύτητα) αναφέρεται στον ρυθμό που τα δεδομένα παράγονται και επεξεργάζονται, και ο όρος Volume (όγκος) αναφέρεται στην ποσότητα-μέγεθος των διαθέσιμων δεδομένων.

Σχετικά με το Volume, τα τελευταία χρόνια παρατηρούμε μια εκθετική αύξηση στον όγκο των δεδομένων που καλούμαστε να αποθηκεύσουμε, αφού αυτά έπαψαν να είναι κατά κύριο λόγω δεδομένα σε μορφή κειμένου. Πλέον, έχουμε τεράστιες ποσότητες δεδομένων σε μορφή video, ήχου και εικόνων τόσο σε επιστημονικές εφαρμογές όσο και στα ευρέως πλέον διαδεδομένα κοινωνικά δίκτυα. Είναι πλέον πολύ συνηθισμένο μια επιχείρηση να διαθέτει Terabytes ή ακόμα και Petabytes αποθηκευτικού χώρου. Έτσι, καθώς αυξάνονται τα δεδομένα, χρειάζεται να επανεξεταστούν οι εφαρμογές και οι αρχιτεκτονικές που τα υποστηρίζουν καθώς οι συμβατικές μέθοδοι δεν επαρκούν.

Το Velocity αναφέρεται στον ταχύτατο ρυθμό με τον οποίο εισέρχονται νέα δεδομένα αλλά τα υπάρχοντα ανανεώνονται. Επιπλέον, έχει να κάνει με τον απαιτούμενο χρόνο για την επεξεργασία και ανάλυση των δεδομένων καθώς αυτά εισέρχονται στο σύστημα. Για το πρώτο σκέλος, συνήθως ο φόρτος εργασίας είναι δοσοληψίες (OLTP) και το πρόβλημα είναι πώς το σύστημα θα δεχθεί, θα φιλτράρει, θα διαχειριστεί και θα αποθηκεύσει τα δεδομένα που έρχονται συνεχώς και με ταχύτατο ρυθμό. Τα παραδοσιακά RDBMS εδώ δεν καλύπτουν τις ανάγκες μας, αφού αναγκαστικά επεξεργάζονται πολύ μεγάλο overhead στα δεδομένα για λόγους κλειδώματος, logging, και latching σε πολυνηματικές εφαρμογές. Το δεύτερο σκέλος, αφορά στο χρόνο που απαιτείται ώστε να εξάγουμε πληροφορία από τα εισερχόμενα δεδομένα (stream analysis / mining). Μάλιστα, δεν επαρκεί μόνο να είμαστε σε θέση να αναλύουμε τα δεδομένα και να εξάγουμε πληροφορία σε πραγματικό χρόνο, αλλά επιπλέον είναι απαραίτητο να εκτελούμε και όλες τις λειτουργίες που ενεργοποιούνται από αυτά, σε πραγματικό χρόνο, ώστε να μην καθυστερεί η όλη διαδικασία. Για παράδειγμα μια εφαρμογή που παρακολουθεί τις τιμές των εταιρειών στο χρηματιστήριο, δεν αρκεί να μπορεί να τις καταγράφει, αλλά πιθανώς να θέλουμε και να αγοράζει ή να πουλάει μια μετοχή όταν αυτή ξεπερνά κάποια τιμή.

Όσον αφορά το Variety, υπάρχει πλέον η ανάγκη να αποθηκεύσουμε, να συνδυάσουμε και να επεξεργαστούμε δεδομένα από πολλές διαφορετικές πηγές. Αυτό έχει ως συνέπεια φυσικά να αντιμετωπίζουμε όχι μόνο διαφορετικούς τύπους δεδομένων, αλλά και διαφορετική δομή μεταξύ των ίδιων τύπων. Σε πρώτο επίπεδο, δημιουργείται έτσι η απαίτηση να ενσωματωθούν δεδομένα με αυστηρή δόμηση (structured), ημιδομημένα (semi-structured) και αδόμητα (unstructured). Σε δεύτερο επίπεδο, ακόμα και αν οι πηγές μας χρησιμοποιούν αυστηρή δόμηση των δεδομένων, πιθανόν να είναι ετερογενή, η δόμηση της μιας να μην είναι συμβατή με κάποια άλλη, να χρησιμοποιούν διαφορετική σημασιολογία κλπ. Είναι λοιπόν προφανές πως τα RDBMS που απαιτούν αυστηρή δομή στα δεδομένα τους, δεν μπορούν να

ανταποκριθούν σε αυτές τις νέες προκλήσεις.



Σχήμα 19: Volume, Velocity, Variety

3.2 Technology and Application Challenges

Αποτέλεσμα όλων αυτών ήταν να εντατικοποιηθεί η έρευνα πάνω σε τομείς που θα έδιναν τη δυνατότητα να ξεπεραστούν τα μέχρι πρότινος όρια που υπήρχαν σχετικά με το μέγεθος των δεδομένων που ήταν διαχειρίσιμα.

Οι τομείς αυτοί είναι:

- **High-speed networking:** Η ταχύτερη δικτυακή μεταφορά μεγάλου όγκου δεδομένων είναι μια από τις προκλήσεις που αντιμετωπίζουμε σήμερα. Ενδεικτικά, 1 terabyte απαιτεί περίπου μία ώρα για να μεταφερθεί εντός ενός cluster υπολογιστών, και περίπου μία μέρα για να μεταφερθεί διαμέσου μιας διαδικτυακής σύνδεσης "υψηλής ταχύτητας". Έτσι, καθώς αυξάνεται ο όγκος των δεδομένων που θέλουμε να αποθηκεύουμε και να μεταφέρουμε, θα πρέπει να αυξάνονται αναλόγως και οι ταχύτητες μετάδοσης, ώστε να μπορούν να το υποστηρίξουν.
- **Cluster computer programming:** Ο προγραμματισμός μεγάλης κλίμακας κατανεμημένων υπολογιστικών συστημάτων είναι μια από τις μακροχρόνια προκλήσεις για να μπορέσουμε να επεξεργαστούμε πολύ μεγάλα data sets σε λογικά χρονικά πλαίσια. Το λογισμικό θα πρέπει να είναι σε θέση να ισοκατανείμει τα δεδομένα και τους υπολογισμούς σε όλους τους κόμβους ενός cluster, αλλά και να μπορεί να διαχειρίζεται τις αστοχίες. Μία πολλά υποσχόμενη λύση σε αυτό το ζήτημα είναι το MapReduce framework που αναλύεται αργότερα.
- **Extending the reach of cloud computing:** Μπορεί το AWS της Amazon να έφερε την επανάσταση στο cloud computing, ωστόσο έχει ακόμα αρκετούς περιορισμούς. Αυτοί έχουν να κάνουν κυρίως με το μεγάλο χρονικό αλλά και οικονομικό κόστος για να μεταφερθούν μεγάλα datasets από και προς μια cloud εγκατάσταση. Δεν υπάρχει δηλαδή η δυνατότητα για ευκολία στην κινητικότητα των δεδομένων.

Προς αυτή την κατεύθυνση κινείται το OpenCirrus Project της Intel το οποίο είναι όμως ακόμα σε πρώιμο στάδιο.

- **Machine Learning and other data analysis techniques:** Είναι προφανές πως δεν αρκεί απλά να είμαστε σε θέση να διατηρήσουμε τεράστιους όγκους δεδομένων, αλλά το πιο σημαντικό ίσως είναι να εφαρμόσουμε κατάλληλους αλγόριθμους ώστε να τα αναλύσουμε. Αυτό είναι το αντικείμενο του machine learning, και υφίσταται η ανάγκη για ανάπτυξη αλγορίθμων που να μπορούν να κλιμακώνονται σε τόσο μεγάλα δεδομένα.
- **Widespread Deployment:** Μέχρι πρότινος, όλες οι καινοτομίες στον τομέα των big data έρχονταν αποκλειστικά από τις κορυφαίες εταιρείες του χώρου (Google, Amazon κλπ). Αν και είναι πολλοί οι οργανισμοί που έχουν στη διάθεση τους μεγάλους όγκους δεδομένων, και έχουν την ανάγκη να τα διαχειριστούν αποδοτικά, ελάχιστοι είναι αυτοί που έχουν την τεχνογνωσία να το κάνουν. Στα επόμενα χρόνια

λοιπόν, αναμένουμε να δούμε εξελίξεις στον τομέα αυτό, που να προέρχονται από ένα πιο μεγάλο εύρος οργανισμών.

- **Security and Privacy:** Με τη συλλογή όλο και περισσότερων δεδομένων, πολλά εκ των οποίων αρκετά ευαίσθητα, εγείρονται ερωτήματα σχετικά με το μέχρι ποιο σημείο θα πρέπει να μπορούμε να τα αναλύσουμε. Για παράδειγμα, οι κάμερες ασφαλείας σε πολλά σημεία μιας πόλης προσφέρουν μεγάλη βοήθεια στη διατήρηση της ασφάλειας στην καθημερινότητα, αλλά αν υπάρχει η δυνατότητα να συνδυαστούν τα δεδομένα όλων μαζί, εμφανίζεται ο κίνδυνος της κατάχρησής τους.

Από την άλλη, όπως είπαμε το cloud computing προσφέρει τη δυνατότητα να χρησιμοποιήσουμε μεγάλο αριθμό υπολογιστικών πόρων σε πολύ χαμηλό κόστος. Δίνεται όμως έτσι και η δυνατότητα σε έναν κακόβουλο χρήστη να δημιουργήσει ένα botnet ή να χρησιμοποιήσει μεγάλης κλίμακας παραλληλοποίηση για να σπάσει ένα κρυπτοσύστημα.

3.3 MapReduce

Το MapReduce είναι ίσως το σημαντικότερο εργαλείο που έχουμε στη διάθεσή μας για την ανάλυση Big Data. Είναι ένα προγραμματιστικό μοντέλο, μαζί με τη σχετική υλοποίηση για δημιουργία και επεξεργασία πολύ μεγάλων datasets. Αναπτύχθηκε στη Google από τους Jeffrey Dean και Sanjay Ghemawat. Το κίνητρο για την ανάπτυξή του βρισκόταν στο μεγάλο αριθμό υπολογισμών που γίνονταν κάθε μέρα στη Google πάνω σε τεράστιο όγκο εισερχόμενων δεδομένων. Οι υπολογισμοί αυτοί συνήθως ήταν αρκετά ξεκάθαροι εννοιολογικά, όπως πχ. να βρεθεί το query με τις περισσότερες αναζητήσεις για κάποια μέρα ή να κρατιέται αρχείο με τις ιστοσελίδες που επισκέφτηκε ένας χρήστης μέχρι στιγμής. Λόγω του πολύ μεγάλου αριθμού χρηστών όμως ο όγκος των εισερχόμενων δεδομένων επέβαλλε τη χρησιμοποίηση κατανεμημένων συστημάτων με εκατοντάδες ή και χιλιάδες υπολογιστές ώστε να καταστεί δυνατό η επεξεργασία να ολοκληρωθεί σε λογικά χρονικά πλαίσια. Από τη στιγμή όμως που η επεξεργασία των δεδομένων γίνεται κατανεμημένα, έρχονται στο προσκήνιο πολλά άλλα θέματα όπως το πώς θα γίνει διαμοιρασμός τους, πώς θα παραλληλοποιηθούν οι υπολογισμοί, πώς θα χειριστεί η ανοχή στα σφάλματα και η κατανομή του φορτίου κλπ, δυσχεραίνοντας το έργο του προγραμματιστή, που πλέον δεν μπορεί να επικεντρωθεί αποκλειστικά στον αλγόριθμο που θα εφαρμόσει πάνω στα δεδομένα, αλλά πρέπει να φροντίσει και για όλα τα υπόλοιπα.

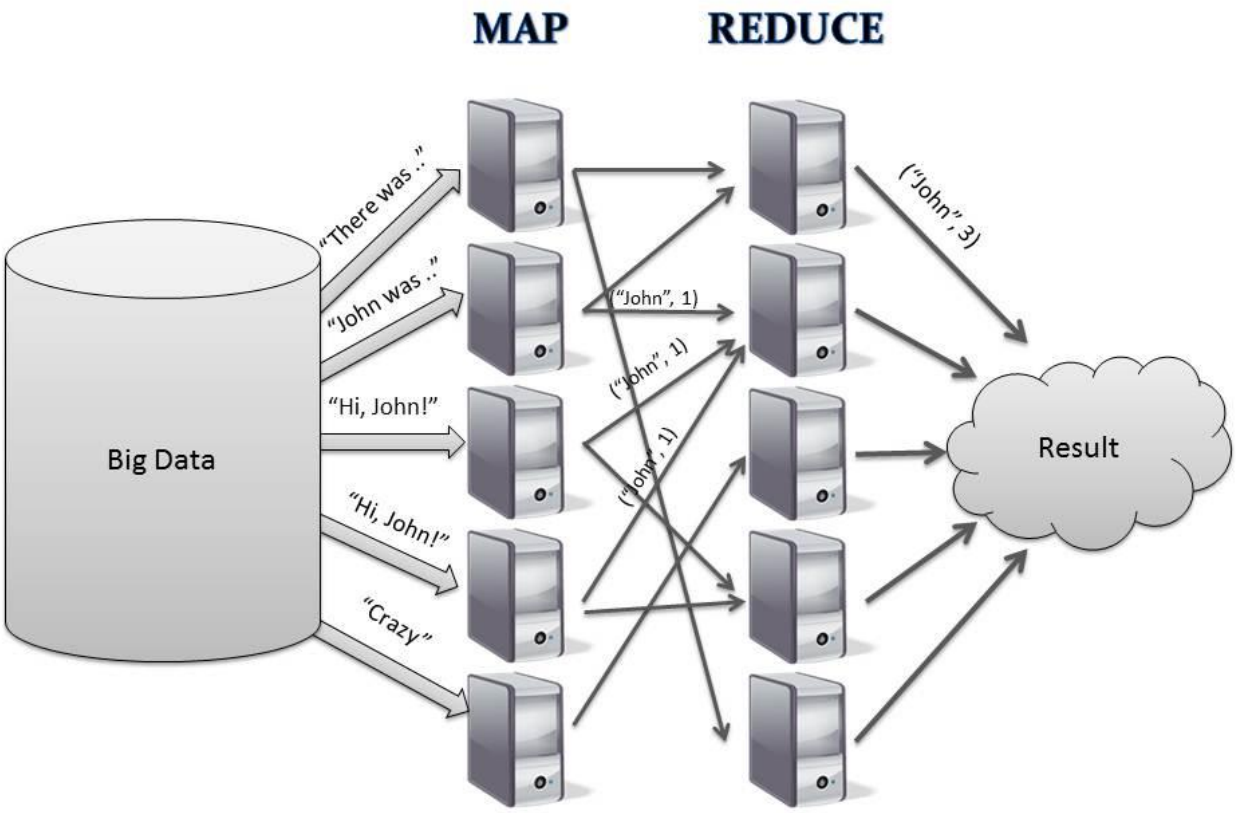
Το MapReduce λοιπόν σχεδιάστηκε για να προσφέρει ένα επίπεδο αφαίρεσης μεταξύ των πραγματικών υπολογισμών πάνω στα δεδομένα που αποτελούν ευθύνη του προγραμματιστή, και των ζητημάτων που προκύπτουν από τη χρήση

κατανεμημένων συστημάτων όπως αναφέραμε πιο πάνω. Είναι ένα απλό και ταυτοχρόνως πολύ δυνατό framework που επιτρέπει στον προγραμματιστή να εκτελεί τις απαιτούμενες εργασίες ως συναρτήσεις map και reduce. Στη συνέχεια, το framework αυτόματα διαμοιράζει τις εργασίες στο διαθέσιμο cluster, αναλαμβάνοντας έτσι τη διαχείριση όλων των ζητημάτων που θίξαμε προηγουμένως. Μερικοί από τους πιο συνήθεις και ενδιαφέροντες υπολογισμούς για τους οποίους μπορούμε να χρησιμοποιήσουμε το MapReduce είναι:

- **Distributed Grep**: Έλεγχος για το ποιες γραμμές από διάφορα κείμενα ταιριάζουν με κάποιο δοθέν pattern.
- **Count of URL Access Frequency**: Επεξεργασία αρχείων με requests web ιστοσελίδων ώστε να βρεθεί ο συνολικός αριθμός επισκέψεων για κάθε ιστοσελίδα.
- **Reverse Web-Link Graph**: Δοθείσας μιας λίστας από ζεύγη URL της μορφής (προορισμός, πηγή), επιστρέφει ζεύγη (προορισμός, Λίστα(πηγές)), δηλαδή όλα τα URL που οδηγούν σε ένα συγκεκριμένο κόμβο.
- **Term-Vector per Host**: Εδώ συνοψίζονται οι πιο σημαντικοί όροι που περιέχονται σε ένα σύνολο από έγγραφα, ως ια λίστα από ζεύγη (όρος, συχνότητα).
- **Inverted Index**: Στην ίδια βάση με προηγουμένως, εδώ επιστρέφουμε ένα ανεστραμμένο ευρετήριο των όρων, δηλαδή για κάθε λέξη ενός συνόλου από έγγραφα, εξάγεται μια λίστα από τα έγγραφα που την περιέχουν (λέξη, Λίστα(id_εγγράφου)).
- **Distributed Sort**: Εδώ η συνάρτηση map εξάγει κάποια ζεύγη της μορφής (key, record), τα οποία επιστρέφει η reduce απαράλλαχτα. Ωστόσο, λόγω του μηχανισμού partitioning επιτυγχάνεται ταξινόμηση των keys.

Για να δημιουργήσει μια MapReduce εργασία, ο προγραμματιστής πρέπει να ορίσει τις συναρτήσεις map και reduce. Η ιδέα αυτή προέρχεται από τις αντίστοιχες συναρτήσεις στις συναρτησιακές γλώσσες προγραμματισμού. Το MapReduce framework τρέχει πολλαπλά στιγμιότυπα αυτών των συναρτήσεων παράλληλα. Η συνάρτηση map επεξεργάζεται ένα ζεύγος κλειδιού – τιμής και παράγει ένα άλλο ζεύγος κλειδιού – τιμής. Έτσι συγκεντρώνονται όλα τα ενδιάμεσα ζεύγη τιμών που προέρχονται από όλα τα στιγμιότυπα των συναρτήσεων map και διαμοιράζονται στους reducers. Η reduce συνάρτηση παίρνει ένα ζεύγος από κλειδί και λίστα από τις τιμές που της αντιστοιχούν και επεξεργάζεται τις τιμές αυτές για να παράγει την τελική έξοδο.

Σχηματικά :



Σχήμα 20 : Map Reduce Execution Model

Ας δούμε ένα απλό παράδειγμα MapReduce προγράμματος για να γίνει καλύτερα κατανοητή η λειτουργία του. Πρόκειται για το πολύ απλό πρόγραμμα Wordcount που μετράει πόσες φορές εμφανίζεται κάθε λέξη μέσα σε ένα κείμενο.

```
map(key, value):
//key: document name; value: text of document
       for each word w in value:
              emit(w,1)

reduce(key, values):
//key: a word; value: an iterator over counts
       result = 0
       for each count v in values:
              result += v
       emit(result)
```

Σχήμα 21 : Map Reduce Word Count Example

Η συνάρτηση map παίρνει ως key το id του κειμένου, και ως value ένα κομμάτι του, και για κάθε λέξη που συναντά, εξάγει ένα ζεύγος της μορφής (word, 1). Τα ζεύγη αυτά διαμοιράζονται στους reducers με βάση το key (word). Η συνάρτηση reduce τώρα, παίρνει ως key κάποια από αυτές τις λέξεις και ως value μια λίστα από 1, ένα για κάθε φορά που συνάντησε τη λέξη αυτή κάποιος mapper. Έτσι, αθροίζοντας αυτές τις τιμές, εξάγει ένα ζεύγος της μορφής (word, total_count) που είναι ο συνολικός αριθμός εμφανίσεων για κάθε λέξη μέσα στο έγγραφο.

ΚΕΦΑΛΑΙΟ 4

Αντικείμενο της Διπλωματικής

4.1 Περιγραφή Dataset

X3D

Τα αρχικά δεδομένα πάνω στα οποία θα δουλέψουμε, είναι ένα σύνολο από x3d (eXtensible 3d) αρχεία. Το x3d είναι η πρότυπη μορφή αρχείου για την αναπαράσταση τρισδιάστατων γραφικών. Διαδέχθηκε τη Virtual Reality Modeling Language (VRML), επεκτείνοντάς την, επιτρέποντας την κωδικοποίηση μιας σκηνής χρησιμοποιώντας XML σύνταξη, ή με δυαδική αναπαράσταση (binary formatting) και παρέχει ενισχυμένα APIs. Το x3d μπορεί να συνδυαστεί με άλλα open source πρότυπα όπως XML, XDOM, Xpath κλπ. Περιλαμβάνει ένα πλούσιο σύνολο χαρακτηριστικών για χρήση στη μηχανική και επιστημονική απεικόνιση, CAD και αρχιτεκτονική, ιατρική απεικόνιση, εκπαίδευση και εξομοίωση, πολυμέσα, ψυχαγωγία κ.α. Το κυριότερα στοιχεία που λήθφηκαν υπόψιν κατά το σχεδιασμό του x3d είναι:

- Χρήση ανοικτού κώδικα, ώστε να μην υπάρχουν προβλήματα αδειών χρήσης.
- Επίσημη ενσωμάτωση στο πρότυπο πολυμέσων MPEG-4.
- Υποστήριξη XML ώστε να διευκολύνεται η διάθεση τρισδιάστατων δεδομένων προς Web υπηρεσίες και κατανεμημένες εφαρμογές.
- Συμβατότητα με την επόμενη γενιά αρχείων γραφικών, όπως τα Scalable Vector Graphics (SVG).
- Τα 3d αντικείμενα να είναι διαχειρίσιμα από τις κυριότερες γλώσσες προγραμματισμού (C/C++, Java κλπ.).

Το x3d ορίζει διάφορα προφίλ, για την επίτευξη διαφορετικών δυνατοτήτων:

X3D Interchange: Το πιο βασικό προφίλ για την επικοινωνία μεταξύ εφαρμογών. Υποστηρίζει γεωμετρικά σχήματα, υφές, βασικό φωτισμό και κίνηση.

X3D Interactive: Παρέχει διάφορες βασικές ιδιότητες αλληλεπίδρασης με το τρισδιάστατο περιβάλλον προσθέτοντας αισθητήρες που χρησιμοποιούνται για την περιήγηση και αλληλεπίδραση του χρήστη στο περιβάλλον. Προσθέτει ακόμα και ρολόι, και ακόμη περισσότερες επιλογές φωτισμού.

X3D Immersive: Εδώ ενεργοποιούνται πλήρως τα τρισδιάστατα γραφικά και η αλληλεπίδραση του χρήστη με αισθητήρες, παρέχοντας επιπλέον ενσωμάτωση ήχου, ανίχνευση συγκρούσεων τρισδιάστατων αντικειμένων, ομίχλη, καθώς και δυνατότητα για μικρού εύρους scripting σε Javascript.

X3D Full: Περιλαμβάνει όλα τα υπαρκτά nodes, επιπρόσθετα NURBS, H- Anim και Geospatial components.

Κάθε εφαρμογή x3d αποτελεί ένα τρισδιάστατο χώρο ορισμένο στο χρόνο (3D time - based space) που περιέχει γραφικά και ηχητικά αντικείμενα που μπορούν να φορτωθούν από δίκτυο και να μεταβληθούν δυναμικά από ένα σύνολο μηχανισμών. Πιο απλά, μια x3d εφαρμογή:

- Ορίζει και συνθέτει ένα σύνολο από τρισδιάστατα και πολυμεσικά αντικείμενα.
- Ορίζει εάν σύστημα συντεταγμένων στο χώρο, για όλα τα αντικείμενα που περιέχονται στο αρχείο ή γίνονται include.
- Καθορίζει υπερσυνδέσμους (hyperlinks) προς άλλα αρχεία και εφαρμογές.
- Μπορεί να ορίσει συμπεριφορές των αντικειμένων
- Μπορεί να συνδεθεί σε εξωτερικά τμήματα ή εφαρμογές μέσω scripting.

Το πρότυπο, εξαιτίας των δυνατοτήτων του, υποστηρίζεται από αρκετές εταιρίες ανάπτυξης, κυρίως εργαλείων για επισκόπηση X3D (browsers). Η ευελιξία και επεκτασιμότητα του προτύπου το καθιστούν ικανό να υποστηρίξει όλες τις σύγχρονες εξελίξεις στον τομέα της τρισδιάστατης απεικόνισης (3D Visualization), π.χ. Normal mapping.

MPEG - 7

Τα τελευταία χρόνια, η μεγάλη αύξηση του όγκου πολυμεσικών δεδομένων που καλούμαστε να διαχειριστούμε, συμβαδίζει με μια ανάλογη αύξηση της προσπάθειας για την ανάπτυξη τέτοιων τεχνολογιών. Ένα από τα κυριότερα σημεία που χρήζουν προτυποποίησης είναι η εξαγωγή χρήσιμης πληροφορίας από τα αρχεία πολυμέσων. Είναι πολύ σημαντικό να υπάρχει ένα κοινό πρωτόκολλο για τον ορισμό και τη χρησιμοποίηση πολυμεσικών δεδομένων, ώστε να διευκολύνεται η διαλειτουργικότητα, η κλιμακωσιμότητα και η ανεξαρτησία από την πλατφόρμα στην οποία τρέχει μια πολυμεσική εφαρμογή.

Αυτός είναι και ο κύριος στόχος του προτύπου MPEG-7. Το MPEG-7 διαφέρει

από τα προηγούμενα πρότυπα (MPEG-1, MPEG-2 και MPEG-4) ως προς το ότι δεν είναι ένα πρότυπο κωδικοποίησης αρχείων βίντεο, και δεν προτίθεται να αντικαταστήσει κάποιο από αυτά. Η αποστολή του είναι να παρέχει τις κατάλληλες περιγραφές για τη μετάδοση και αποθήκευση πολυμεσικού περιεχομένου όπως εικόνες, ήχος και βίντεο. Αυτοί οι περιγραφείς μπορούν να χρησιμοποιηθούν για να συγκρίνουν, να φιλτράρουν ή να αναζητήσουν πολυμέσα αποκλειστικά με βάση το οπτικό περιεχόμενο του αρχείου. Για το λόγο αυτό, οι περιγραφείς μπορεί να περιέχουν πληροφορίες σχετικά με:

- Τα χαρακτηριστικά της δομής του περιεχομένου (μορφή αποθήκευσης, κωδικοποίηση κλπ.)
- Low-level χαρακτηριστικά όπως χρώμα, επιφάνεια, ήχος, μουσική κ.α.
- Χωρικά, χρονικά ή και χωροχρονικά στοιχεία του περιεχομένου όπως τμηματοποίηση μιας σκηνής σε περιοχές, εντοπισμός κίνησης μιας περιοχής κ.ά.
- Εννοιολογικές πληροφορίες σχετικά με το περιβάλλον που απεικονίζεται από το περιεχόμενο, όπως τον ορισμό αντικειμένων και γεγονότων, αλληλεπίδραση μεταξύ των αντικειμένων κ.ά.
- Πληροφορίες σχετικά με τη χρήση του περιεχομένου όπως πνευματικά δικαιώματα, ιστορικό χρήσης κλπ.
- Πληροφορίες που σχετίζονται με τη δημιουργία του περιεχομένου και τη διαδικασία παραγωγής του, όπως το όνομα του δημιουργού, τον τίτλο, ημερομηνία κλπ.

Οι περιγραφείς εφαρμόζονται στα δεδομένα ανεξάρτητα από το μέσο μετάδοσης και τον τρόπο κωδικοποίησης, η αναπαράστασή τους γίνεται με αντικειμενοστραφή τρόπο και μπορούν να επεκταθούν ανάλογα με τις ανάγκες τους χρήστη. Ορίζονται 4 εργαλεία, description tools, για την περιγραφή του περιεχομένου των αρχείων, οι Descriptors, τα Description Schemas, η Description Definition Language (DDL) και τα System tools.

Descriptors (D): Ένας περιγραφές (Descriptor) αναπαριστά ένα στοιχειώδες χαρακτηριστικό και ορίζει τη σύνταξη και τη σημασιολογία μιας τέτοιας αναπαράστασης. Πιθανοί περιγραφείς είναι: ένα ιστόγραμμα χρώματος, οι time-codes για την αναπαράσταση χρονικής διάρκειας, ένα πεδίο κίνησης, το κείμενο ενός τίτλου κλπ.

Description Schemes (DS): Ένα σχήμα περιγραφής (Description Schema), ορίζει τη δομή και τη σημασιολογία των συσχετίσεων μεταξύ των συστατικών τους, που μπορεί να είναι είτε descriptors είτε description schemes. Η δομή και η σύνταξη των Description Schemes ορίζεται από την DDL (Description Definition Language), η οποία σε αρκετές περιπτώσεις δίνει τη δυνατότητα δημιουργίας νέων σχημάτων περιγραφής.

Description Definition Language (DDL) : Είναι μια γλώσσα βασισμένη στην XML και χρησιμοποιείται για να ορίζει δομικές σχέσεις μεταξύ των περιγραφέων. Επιτρέπει τη δημιουργία και την τροποποίηση των description schemes αλλά και τη δημιουργία νέων descriptors.

System Tools: τα εργαλεία αυτά υποστηρίζουν το multiplexing των περιγραφών, το συγχρονισμό των περιγραφών με το συσχετιζόμενο περιεχόμενο, τη δυαδική αναπαράσταση για αποδοτική αποθήκευση και μετάδοση κλπ.

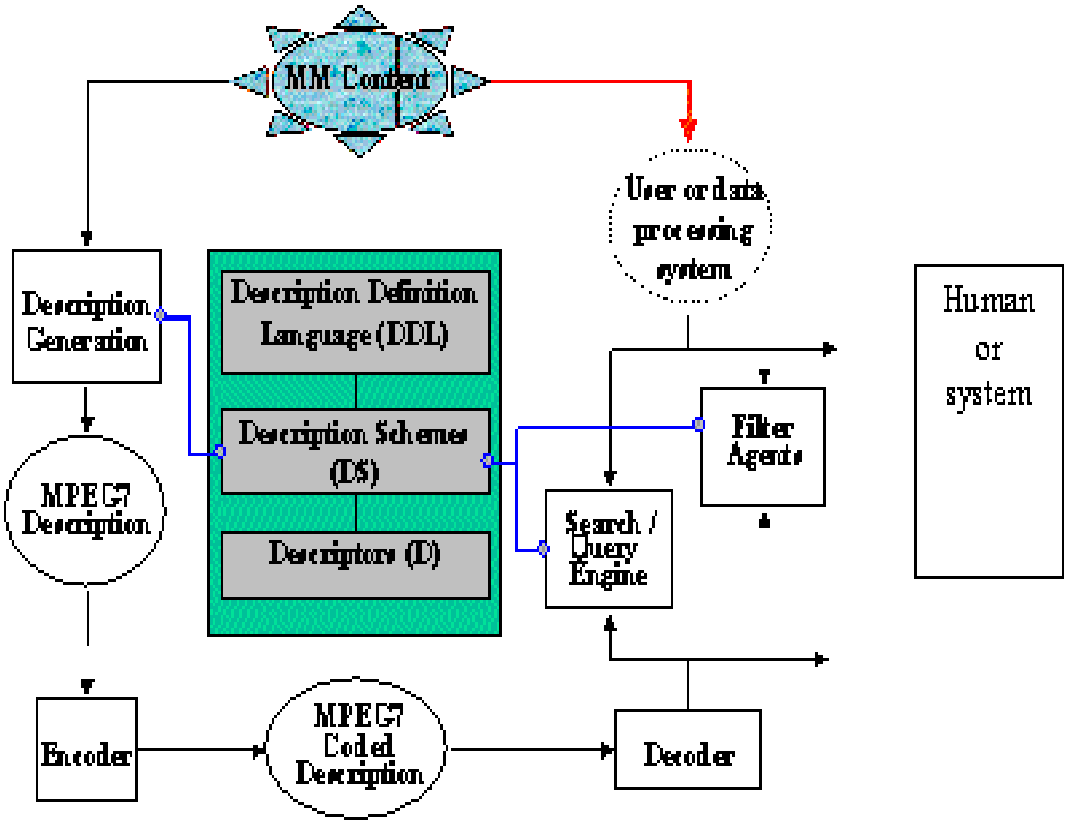
Ένα τυπικό σενάριο εφαρμογής υποδεικνύει πως οι MPEG-7 περιγραφείς εξάγονται από το περιεχόμενο του αρχείου. Στις περισσότερες περιπτώσεις, το πρότυπο MPEG-7 ορίζει μόνο εν μέρει πώς να γίνει η εξαγωγή αυτών των χαρακτηριστικών, ώστε να επιτρέπει όσο το δυνατόν μεγαλύτερη ευελιξία στις διάφορες εφαρμογές. Επιπλέον, αφήνεται ελεύθερο το πώς θα χρησιμοποιηθούν οι MPEG-7 περιγραφείς για επιπλέον επεξεργασία του περιεχομένου. Ωστόσο, ως διεθνές πρότυπο, η σύνταξη των περιγραφέων θα πρέπει να ικανοποιεί τις προδιαγραφές του MPEG-7. Αυτό εξασφαλίζει πως όλες οι εφαρμογές που είναι συμβατές με το MPEG-7 θα μπορούν ανταλλάσσουν δεδομένα. Για αυτό το λόγο, τα πρότυπα MPEG-7 οργανώνονται σε 7 επιμέρους κομμάτια, καθένα από τα οποία περιλαμβάνει τις προδιαγραφές για την επίλυση διαφορετικών πτυχών του προβλήματος. Τα μέρη αυτά είναι:

1. **Systems**: Στο κομμάτι αυτό ορίζονται λειτουργικότητες σε επίπεδο συστήματος, όπως αποδοτική μεταφορά, αποθήκευση ή συγχρονισμός των περιγραφών. Οι περιγραφές μπορεί να αποθηκεύονται ξεχωριστά ή να ενσωματώνονται στο περιεχόμενο. Επιπλέον, μπορεί να ορίζεται και η μορφοποίηση των δεδομένων.
2. **Description Definition Language (DLL):** Είναι η προτυποποιημένη σύνταξη που ορίζει τα Description Schemes και βασίζεται σε μια επέκταση του Extensible Markup Language (XML) Schema της W3C. Παρέχει όλες τις απαραίτητες τιμές και παραμέτρους για την περιγραφή αρχείων πολυμεσικού περιεχομένου.
3. **Visual**: Αυτό το κομμάτι ορίζει τους κύριους περιγραφείς και description schemes για οπτικό (visual) περιεχόμενο. Κυρίως χρησιμοποιείται χρώμα, επιφάνεια, σχήμα, κίνηση και τοπικότητα. Περιλαμβάνονται επίσης descriptors για την περιγραφή προσώπων.
4. **Audio**: Στο κομμάτι αυτό ορίζονται οι κύριοι περιγραφείς για τον ήχο. Χωρίζεται σε 4 κύριους τύπους ακουστικών σημάτων: pure music, pure speech, pure sound effects, και arbitrary soundtracks.
5. **Multimedia Description Schemes**: Εδώ ορίζονται τα εργαλεία για την περιγραφή περιεχομένου που δεν είναι ακουστικό ή οπτικό, δηλαδή generic. Εδώ περιλαμβάνεται ο μεγαλύτερος αριθμός description tools και η περιγραφή για το περιεχόμενο γίνεται

με ιεραρχική δομή.

6. **Reference Software**: Το κομμάτι αυτό παρέχει ένα ανοιχτού κώδικα λογισμικό για την επίδειξη του προτύπου MPEG-7. Είναι γνωστό ως Experimentation Model (XM).

7. **Conformance Tests**: Παρέχει οδηγίες για τη δοκιμή της εγκυρότητας των περιγραφών που έχουν υποβάλλει οι χρήστες.



Σχήμα 22: MPEG-7 Applications

4.2 Περιγραφή Προβλήματος

Όπως τονίσαμε σε προηγούμενα κεφάλαια, η αλματώδης αύξηση του όγκου των δεδομένων που καλούμαστε να διαχειριστούμε έχει οδηγήσει στην ανάγκη για εύρεση νέων, πιο αποδοτικών τεχνικών αποθήκευσης. Ειδικότερα στα Multimedia, το πρόβλημα αυτό εμφανίζεται με πολύ μεγάλη συχνότητα, καθώς το πολύ μεγάλο εύρος πληροφορίας που διατηρούν συνεπάγεται αντίστοιχες απαιτήσεις σε αποθηκευτικό χώρο και δυσχεραίνει την αναζήτηση. Για την εξαγωγή χρήσιμης και εύκολα διαχειρίσιμης πληροφορίας από x3d αρχεία είναι προτιμότερο να εξάγουμε την .mp7

περιγραφή τους. Χρησιμοποιήσαμε λοιπόν ένα software tool και εξάγαμε τα αντίστοιχα mp7 αρχεία από το αρχικό μας dataset, σύμφωνα με μια επέκταση του προτύπου αυτού που δημιουργήθηκε στα πλαίσια του έργου iPromotion. Το κυρίως πρόβλημα που κληθήκαμε να αντιμετωπίσουμε ωστόσο ήταν η εύρεση ενός αποτελεσματικού τρόπου για την αποθήκευση αυτών των δεδομένων ώστε να μπορούμε να τα διατρέχουμε και να τα αναζητούμε γρήγορα. Η χρησιμοποίηση μιας από τις ήδη γνωστές και υπάρχουσες τεχνικές δεν θα είχε κάποιο ικανοποιητικό αποτέλεσμα. Μια συγκριτική μελέτη αυτών των τεχνικών έγινε σε προηγούμενη διπλωματική εργασία στα πλαίσια του ίδιου project και έγινε φανερό πως οι παραδοσιακές τεχνικές δεν παρέχουν ικανοποιητικά αποτελέσματα. Στην εργασία αυτή εξετάστηκαν λοιπόν οι δύο δημοφιλέστερες τεχνικές (αναζήτηση σε XML αρχεία και αναζήτηση σε RDBMS).

Όσον αφορά την αναζήτηση σε XML αρχεία, αυτή γίνεται με χρήση του XQuery, μιας γλώσσας με σκοπό την υποβολή queries σε συλλογές XML αρχείων. Ωστόσο, το μεγάλο overhead που αναγκαστικά επιφέρει το xml parsing και το γεγονός ότι δεν χρησιμοποιείται κάποια optimized βάση δεδομένων αλλά όλα τα αρχεία διατρέχονται τοπικά στο file system, επιφέρουν αύξηση του χρόνου εκτέλεσης ενός query ανάλογα με το μέγεθος του dataset. Αυτό φυσικά κάνει απαγορευτική τη χρήση της μεθόδου αυτής σε περιβάλλοντα Big Data.

Όσον αφορά την αναζήτηση σε RDBMS τώρα, γνωρίζουμε ήδη πως τα τυπικά σχεσιακά συστήματα βάσεων δεδομένων δυσκολεύονται να ανταποκριθούν στις τεράστιες αποθηκευτικές απαιτήσεις των Big Data. Ακόμη, όπως τονίσαμε και σε προηγούμενα κεφάλαια, η χρήση του σχεσιακού μοντέλου απαιτεί την ύπαρξη ενός αυστηρού schema για τα δεδομένα. Αυτό αποτελεί μεγάλο μειονέκτημα όταν πρόκειται να αποθηκευτούν XML αρχεία και ιδίως MPEG-7 περιγραφές, όπου αν και υπάρχει η δόμηση σύμφωνα με τους κανόνες που ορίζει το πρότυπο, παρατηρούνται πολλές ελευθερίες στον τρόπο αναπαράστασης της πληροφορίας. Όπως αναφέρθηκε και σε προηγούμενο κεφάλαιο όμως, οι σχεσιακές βάσεις δεδομένων δεν ενδείκνυνται για την αποθήκευση δεδομένων χωρίς αυστηρό schema. Στην προκειμένη περίπτωση δηλαδή θα έπρεπε να κάνουμε χρήση πολύ μεγάλου αριθμού από null values για κάθε αρχείο ώστε να καλύψουμε όλους τους πιθανούς descriptors. Αυτό φυσικά θα οδηγούσε σε πολλαπλάσιο τελικό μέγεθος της βάσης και επομένως πολύ χαμηλότερη απόδοση σε αποθηκευτικό χώρο αλλά και ταχύτητα εκτέλεσης των queries. Ακόμα, το query model των RDBMS θα απέδιδε πολύ χειρότερα από αυτό των NoSQL βάσεων. Το κύριο σενάριο πάνω στο οποίο θα χρησιμοποιηθούν τα δεδομένα μας είναι για context based retrieval. Επομένως, θα μας ήταν περισσότερο χρήσιμο ένα μοντέλο σαν το map reduce, όπου θα μπορούσαμε να επεξεργαστούμε παράλληλα μεγάλο όγκο δεδομένων, μιας και δεν υπάρχουν πουθενά εξαρτήσεις, και στο τέλος να ομαδοποιούμε τα αποτελέσματα με βάση το context στο οποίο γίνεται η αναζήτηση. Για τους λόγους αυτούς λοιπόν επιλέξαμε να χρησιμοποιήσουμε μια

NoSQL βάση δεδομένων και να εκτελέσουμε τα ερωτήματα με map reduce.

4.3 Υλοποίηση

Αρχικά, όπως τονίσαμε χρειάζεται να εξάγουμε τους απαραίτητους περιγραφείς από τις x3d σκηνές για να είναι πιο εύκολα διαχειρίσιμη η πληροφορία που περιέχουν. Έτσι, αρχικά χρησιμοποιήσαμε ένα software tool με σκοπό την εξαγωγή των mp7 περιγραφών από τα αρχεία x3d. Για κάθε mp7 περιγραφή που εξάγουμε, ελέγχουμε αν είναι σύμφωνη με το πρότυπο που χρησιμοποιούμε, σύμφωνα με το αντίστοιχο xsl αρχείο. Αφού ολοκληρωθεί αυτή η διαδικασία για όλο το dataset, προχωράμε στην αποθήκευση των δεδομένων.

Για εξοικονόμηση χώρου και χρόνου εφαρμόζουμε την εξής τεχνική στην αποθήκευση των αρχείων στη βάση. Για κάθε x3d σκηνή δημιουργούμε ένα zip με όλα τα αρχεία που τη συνθέτουν (αρχεία ήχου, textures κλπ). Το zip αυτό το αποθηκεύουμε στη MongoDB βάση μας κατανεμημένα, στα διάφορα chunks μέσω του GridFS. Το GridFS παρέχει αφενός την απαραίτητη για εμάς λειτουργικότητα που είναι η αποθήκευση αρχείων μεγέθους μεγαλύτερου των 16MB, αλλά επιπλέον έχει το χαρακτηριστικό πως χρησιμοποιεί δύο συλλογές για την αποθήκευση κάθε αρχείου. Η μία είναι για την αποθήκευση των chunks (chunks collection) που συντελούν το αρχείο και η άλλη (files collection) για την αποθήκευση των metadata για το κάθε αρχείο. Στη δεύτερη λοιπόν αποθηκεύουμε στο πεδίο metadata, την mp7 περιγραφή του αρχείο σε μορφή json. Εναλλακτικά θα μπορούσαμε να την αποθηκεύσουμε ως ένα String το οποίο και να επεξεργαζόμαστε στη συνέχεια ώστε να βρούμε τα πεδία που μας ενδιαφέρουν. Ωστόσο, το parsing ενός json αρχείου γίνεται πολύ πιο αποδοτικά, λόγω της συγκεκριμένης δομής του (key – value pairs). Η τεχνική αυτή που ακολουθήσαμε μας επιτρέπει αφενός να εξοικονομήσουμε αρκετό χώρο, αφού δεν αναγκαζόμαστε να αποθηκεύσουμε και τα mp7 ξεχωριστά, αλλά κυρίως μας δίνει τη δυνατότητα με ένα "πέρασμα" να κάνουμε retrieve και το zip αρχείο με ολόκληρη τη σκηνή αλλά και την mp7 περιγραφή.

Για να εξετάσουμε την επίδοση της υλοποίησης μας χρειαζόμασταν ένα ερώτημα προς εκτέλεση προς τη βάση, το οποίο να είχε μεγάλες απαιτήσεις σε υπολογιστικούς πόρους αλλά παράλληλα να είχε και αρκετά μεγάλη σημασία για ένα multimedia dataset. Ένα τέτοιο ερώτημα είναι η εύρεση του ιστογράμματος χρώματος των σκηνών. Το ιστόγραμμα χρώματος (σταθμισμένο με τον όγκο των γεωμετριών) είναι ένας απλός τρόπος να περιγραφεί το χρωματικό περιεχόμενο μιας σκηνής. Στον τομέα της επεξεργασίας εικόνας θεωρείται πως το ιστόγραμμα χρώματος (color histogram)

είναι ο απλούστερος τρόπος να περιγραφεί το χρωματικό περιεχόμενο μιας εικόνας χωρίς να χάσουμε όμως σημαντική πληροφορία. Δεν είναι τυχαίο άλλωστε πως το color histogram παραμένει ο πιο συνηθισμένος περιγραφέας εικόνας από το 1993 που προτάθηκε για πρώτη φορά μέχρι σήμερα, παρά τα άλματα που έχουν γίνει από τότε μέχρι σήμερα στο image description. Στην πραγματικότητα, ο μόνος απλούστερος τρόπος να περιγραφεί το χρώμα είναι οι λεγόμενες χρωματικές ροπές (color moments), δηλ στατιστικές ροπές επί του χρώματος (μέση τιμή, διακύμανση κλπ). Τέτοιες στατιστικές όμως τείνουν να είναι ισοπεδωτικές (πχ. μια εικόνα με τα μισά πίξελ μπλε και τα μισά πιξελ κόκκινα έχει μέσο χρώμα το μωβ, το ίδιο με μια εικόνα που είναι 100% μωβ). Το ιστόγραμμα προσφέρει ένα καλό tradeoff απλότητας/λεπτομέρειας, τουλάχιστον για τα πρώτα πειράματα. Για να υπολογίσουμε το ιστόγραμμα χρώματος χρειάζεται αφενός να συγκεντρώσουμε τα επί μέρους χρώματα που υπάρχουν σε όλα τα αντικείμενα μιας σκηνής, αφετέρου να υπολογίσουμε τον όγκο του κάθε αντικειμένου αλλά και αθροιστικά, ούτως ώστε να σταθμίσουμε τις τιμές των χρωμάτων. Για να το κάνουμε λοιπόν αυτό, διαιρούμε κάθε ζώνη του φάσματος των χρωμάτων RGB [0, 256), [0, 256), [0, 256) σε 8 επί μέρους τμήματα ([0, 32), [32, 64) ..[224, 256)). Οπότε τελικά έχουμε διαιρέσει όλο το φάσμα RGB σε 8*8*8 = 512 επί μέρους κελιά, όπου το πρώτο αντιστοιχεί στις τιμές R:[0, 32) , G:[0, 32), B:[0, 32), το δεύτερο στις τιμές R:[32, 64), G:[0, 32), B:[0, 32) κ.ο.κ μέχρι το $512^{ο}$: R:[224, 256), G:[224, 256), B:[224, 256). Στη συνέχεια, από κάθε αντικείμενο – γεωμετρία της σκηνής υπολογίζουμε τον όγκο της, και το χρωματικό κελί στο οποίο αντιστοιχεί και τον προσθέτουμε σε αυτό το κελί. Όταν διατρέξουμε όλες τις σκηνές, αθροίζουμε τις τιμές όλων των κελιών για να βρούμε δηλαδή το άθροισμα όλων των όγκων της σκηνής και διαιρούμε την τιμή του κάθε κελιού με το άθροισμα αυτό. Οπότε τελικά τα κελιά έχουν άθροισμα 1 και η τιμή κάθε κελιού δείχνει το ποσοστό του "όγκου" της σκηνής που περιέχει το αντίστοιχο χρώμα. Πιο συγκεκριμένα, εμείς εκτελέσαμε δύο παραλλαγές αυτού του ερωτήματος για να πάρουμε συγκεντρωτικά στοιχεία από όλη τη βάση. Στην πρώτη, υπολογίζουμε το ιστόγραμμα χρώματος όλων των σκηνών και εξάγουμε το μέσο ποσοστό για το κάθε κελί, για τις σκηνές στις οποίες συναντάται. Παίρνουμε συνεπώς ένα μέτρο για το πόσο κυρίαρχο ρόλο παίζει το κάθε χρωματικό κελί όταν βρίσκεται σε μια σκηνή. Στη δεύτερη, παίρνουμε το ιστόγραμμα χρώματος συγκεντρωτικά για όλες τις σκηνές της βάσης. Παίρνουμε δηλαδή το άθροισμα των τιμών κάθε κελιού για κάθε σκηνή και το διαιρούμε με το συνολικό αριθμό των σκηνών. Είναι φανερό λοιπόν πως από τη στιγμή που κάθε σκηνή μπορεί να περιέχει από δεκάδες έως και χιλιάδες διαφορετικές γεωμετρίες, τα ερωτήματα αυτά, σε μια βάση δεδομένων με χιλιάδες σκηνές, αποτελούν μια πολύ απαιτητική σε υπολογιστικούς πόρους διαδικασία.

Για να εκτελέσουμε τα ερωτήματα αυτά χρησιμοποιήσαμε δύο μεθόδους. Η πρώτη είναι η παραδοσιακή μέθοδος όπου παίρνουμε από το πεδίο metadata του κάθε αρχείου της βάση, την αντίστοιχη mp7 περιγραφή του ως ένα αρχείο json. Στη

συνέχεια κάνουμε parsing στο αρχείο αυτό ώστε να βρούμε τα πεδία που αντιστοιχούν στις γεωμετρίες της σκηνής και τα αντίστοιχα χρώματα και να πάρουμε τις τιμές τους. Ο αλγόριθμος για τη μέθοδο αυτή φαίνεται παρακάτω:

```
for each Scene s in Database:
        mp7Json = getJson(s)
        for each Descriptor d in parseJson(mp7Json).
            getJsonArray("Descriptors"):
                volume = d.Height  * d. weight * d.
                    depth
                color = d.color
                sumVolumesScene  += volume
                colors[color] += volume
        for each Color c in colors
                c /= sumVolumesScene
```

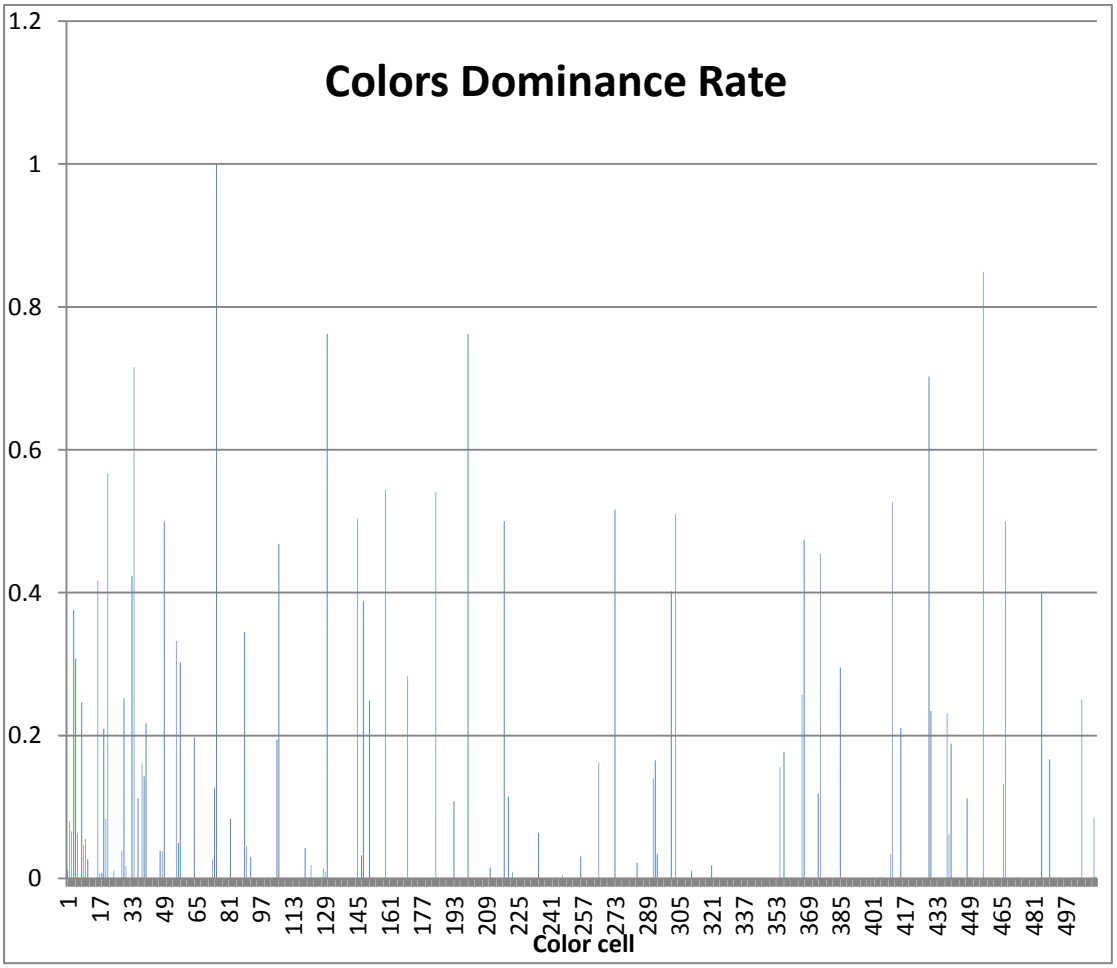
Σχήμα 23 : Color Histogram - Sequential

Στη δεύτερη μέθοδο κάνουμε χρήση του μοντέλου map reduce μέσω του mongodb driver για τη java. Για κάθε περιγραφή αρχείου, από τη στιγμή που είναι σε μορφή json, μπορούμε να προσπελάσουμε τα πεδία του χωρίς να χρειαστεί να το φέρουμε στη μνήμη. Έτσι, στη φάση map κάνουμε emit τα πεδία που μας ενδιαφέρουν, με το κατάλληλο κλειδί, ώστε να τα διαχειριστούμε συναθροιστικά στη φάση reduce. Ο αλγόριθμοι που ακολουθούμε φαίνεται παρακάτω.

```
map1(key, value):
//key: document; value: text of document
       for each Descriptor d in value.Mp7:
              volume = d.Height * d.Depth * d.Width;
              color = d.Color;
              emit(color, volume));


reduce1(key, values):
//key: color values: array of volumes
       sumVolumes = 0;
       for each volume v in values:
              sumVolumes += v;

       return(color, sumVolumes);


map2(key, value):
//key: document; value: text of document (color, sumVolumes) )
       for each tuple t in value:
              emit(null, (t.color, t.volume));


reduce2(key, values):
//key: null; values: array of (color, volume)
       sumVolumesScene = 0;
       for each tuple t in values:
              sumVolumesScene += t.volume;

       for each tuple t in values:
              t.volume /= sumVolumesScene;
       return (values[]);              //array - color
          histogram of the scene
```

Σχήμα 24: Color Histogram – MapReduce

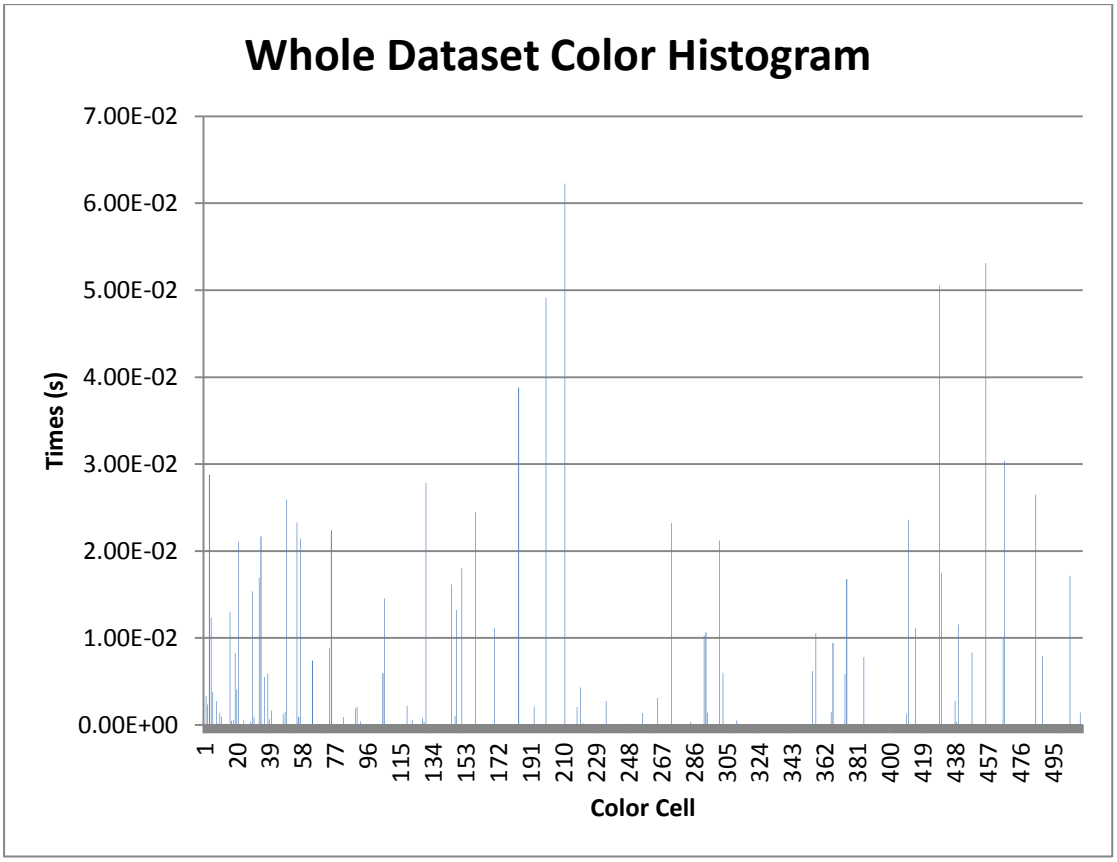
4.4 Αποτελέσματα



Σχήμα 25: Colors Dominance Rate

Εκτελούμε το πρώτο ερώτημα και παίρνουμε τα παραπάνω αποτελέσματα. Βλέπουμε πως οι περισσότερες τιμές κυμαίνονται είτε σε ένα dominance rate κοντά στο 0.5, είτε σε πολύ χαμηλότερες τιμές, κοντά στο 0.1ή και μικρότερες . Αυτό είναι φυσιολογικό να συμβαίνει , αφού σε κάθε σκηνή υπάρχει συνήθως ένα κυρίως αντικείμενο πάνω στο οποίο είναι επικεντρωμένη και πολλά άλλα μικρότερα, τα οποία συνθέτουν το περιβάλλον της αλλά παίζουν δευτερεύοντα ρόλο στη σκηνή. Η πληροφορία αυτή ακριβώς φαίνεται από το dominance rate του κάθε χρωματικού κελιού. Τέλος, παρατηρούμε και λίγα χρωματικά κελιά με πολύ μεγάλες τιμές (>0.8). Αυτές αντιπροσωπεύουν αντικείμενα με σπάνιο χρωματικό περιεχόμενο, τα οποία συναντώνται σε ελάχιστες σκηνές. Η σπανιότητα αυτή του χρώματος έχει άμεση συσχέτιση με τον κυριάρχο ρόλο που παίζουν σε μια σκηνή, εξ' ου και η πολύ υψηλή

αντίστοιχη τιμή.

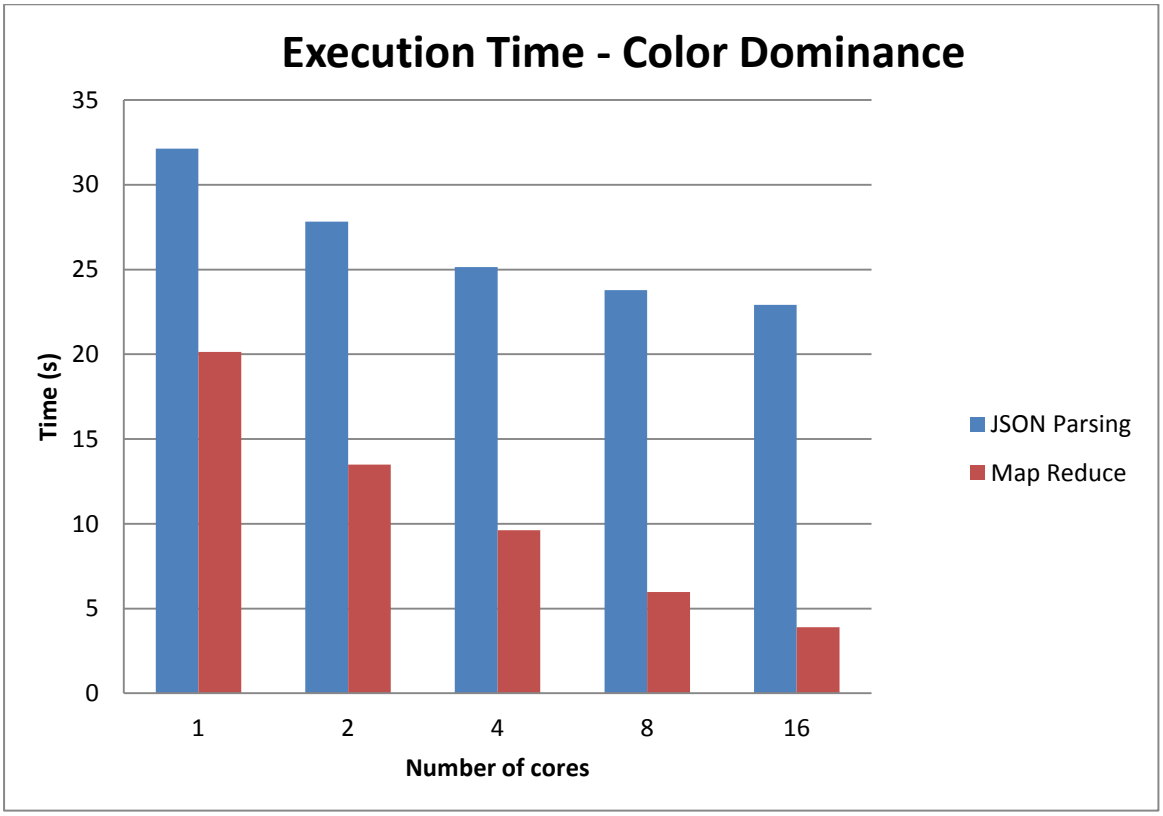


Σχήμα 26: Whole Dataset Color Histogram

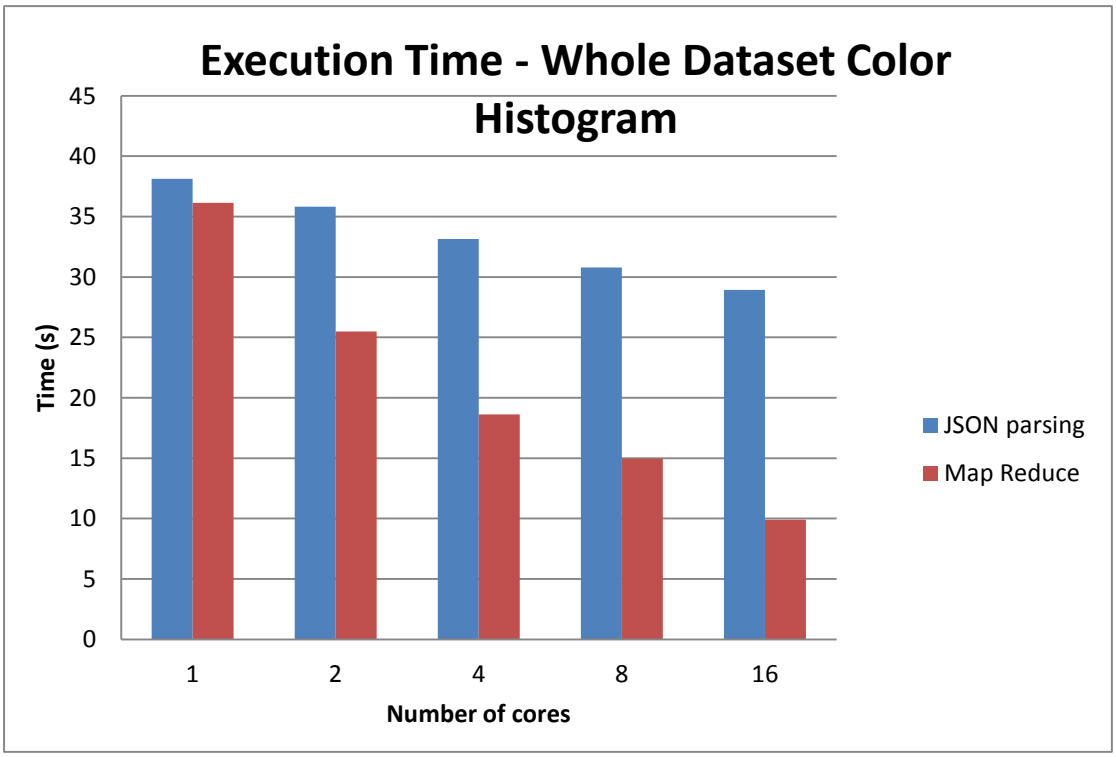
Στο δεύτερο ερώτημα, παίρνουμε την κατανομή των χρωμάτων, αθροιστικά για όλες τις σκηνές της βάσης μας, κανονικοποιημένα ως προς τους αντίστοιχους όγκους. Βλέπουμε πως το χρωματικό κελί με τη μεγαλύτερη συχνότητα στη βάση είναι το κελί 211. Αυτό σύμφωνα με το mapping που έχουμε κάνει αντιστοιχεί σε RGB τιμές: [0, 32), [32, 64), [64, 98). Αυτή είναι μια ανοιχτή απόχρωση του μπλε, πράγμα απολύτως αναμενόμενο, μιας και το dataset που χρησιμοποιήσαμε περιέχει μεγάλο αριθμό ναυτικών αντικειμένων. Πέραν αυτού βλέπουμε κάποιες σχετικά υψηλές τιμές σε χρωματικά κελιά που αντιπροσωπεύουν τα πιο συνηθισμένα χρώματα, και αντίστοιχα πολύ χαμηλές τιμές για χρωματικά κελιά που αντιπροσωπεύουν πιο σπάνια χρώματα. Τέλος, και στα δύο διαγράμματα βλέπουμε αρκετά χρωματικά κελιά να έχουν τιμή 0. Αυτό εξηγείται από την κβαντοποίηση που υπάρχει στα εξαγόμενα

χρώματα από τους περιγραφείς που χρησιμοποιούμε. Δηλαδή, όσο πιο πλούσια είναι η περιγραφή μιας σκηνής, τόσο μικρότερη θα είναι η κβαντοποίηση των χρωμάτων.

Αυτό που έχει περισσότερο ενδιαφέρον όμως είναι οι χρόνοι εκτέλεσης για τα ερωτήματα και στις δύο περιπτώσεις. Στην πρώτη, ο αλγόριθμος που χρησιμοποιούμε είναι πιο ευνοϊκός ως προς το Map Reduce μοντέλο συγκριτικά με τη δεύτερη, καθώς εκτελείται με 2 εργασίες (η δεύτερη με 3). Και για τις δύο περιπτώσεις πάντως αναμένουμε να δούμε καλύτερο χρόνο εκτέλεσης για την Map Reduce υλοποίηση σε σχέση με την πιο παραδοσιακή. Για να εξετάσουμε και το scalability των δύο μεθόδων, κάναμε benchmarking σε VMs με 1, 2, 4, 8 και 16 πυρήνες. Τα αποτελέσματα που πήραμε ως προς το χρόνο εκτέλεσης για τις δύο παραλλαγές του ερωτήματος είναι:



Σχήμα 27: Execution Time - Colors Dominance Rate



Σχήμα 28: Execution Time – Whole Dataset Color Histogram

Αρχικά βλέπουμε τη διαφορά ανάμεσα στο χρόνο των δύο υλοποιήσεων και στις δύο περιπτώσεις. Η διαφορά είναι αρκετά μεγάλη (της τάξης του 300%) και αποδεικνύει ακριβώς ότι η χρησιμοποίηση του προγραμματιστικού μοντέλου Map Reduce είναι μια πολύ καλή λύση για την εξαγωγή πληροφορίας από πολυμεσικά δεδομένα. Επιπλέον φαίνεται πόσο καλύτερα κλιμακώνεται η υλοποίηση με Map Reduce όσο αυξάνεται ο αριθμός των επεξεργαστών. Βλέπουμε πως και στις δύο περιπτώσεις, ο ρυθμός μείωσης του χρόνου εκτέλεσης της Map Reduce υλοποίησης είναι πολύ υψηλότερος συγκριτικά με αυτόν του JSON Parsing. Σε ένα μονοεπεξεργαστικό σύστημα παρατηρούμε ελάχιστες διαφορές ανάμεσα στις δύο υλοποιήσεις. Ωστόσο, θεωρούμε δεδομένο πως τα σύγχρονα υπολογιστικά συστήματα αποτελούνται από μεγάλο αριθμό επεξεργαστών, ιδίως όταν μιλάμε για συστήματα που έχουν σκοπό την αποθήκευση και επεξεργασία Big Data. Στις περιπτώσεις αυτές, οι κατανεμημένες τεχνικές αποτελούν την ιδανικότερη λύση.

4.5 Συμπεράσματα

Η διαχείριση πολύ μεγάλων όγκων δεδομένων είναι ένα από το πιο σημαντικά ανοιχτά ζητήματα στις μέρες μας. Τα δεδομένα που παράγονται από την ολοένα και αυξανόμενη χρήση του διαδικτύου στην καθημερινή μας ζωή αλλά και σε επιχειρηματικά περιβάλλοντα, έχουν αυξηθεί εκθετικά τα τελευταία χρόνια. Για το λόγο αυτό οι τεχνικές που χρησιμοποιούνταν στο παρελθόν δεν επαρκούν πλέον. Στα συστήματα πολυμέσων, λόγω και του μεγάλου εύρους πληροφορίας που μπορεί να περιέχεται σε μια 3d σκηνή ή και σε μια απλή εικόνα, βρισκόμαστε επίσης αντιμέτωποι με αυτό το πρόβλημα. Η χρησιμοποίηση σχεσιακών βάσεων δεδομένων ή xml parsing για την αποθήκευση και αναζήτηση περιεχομένου σε αρχεία πολυμέσων δεν μπορεί πλέον να ανταπεξέλθει στις απαιτήσεις των σύγχρονων συστημάτων Big Data. Το κυριότερο συμπέρασμα που μπορούμε να κρατήσουμε από αυτή την εργασία είναι επομένως η καταλληλότητα των NoSQL βάσεων για αποθήκευση και διαχείριση πολυμεσικών δεδομένων πολύ μεγάλου μεγέθους. Όπως είδαμε, τα μη σχεσιακά συστήματα βάσεων δεδομένων αποτελούν όντως μια πολύ καλή λύση σε αυτό το πρόβλημα. Αρχικά λόγω της ευελιξίας που παρέχουν ως προς τη δομή των δεδομένων που αποθηκεύουμε, κάτι που δεν συναντάμε στα αντίστοιχα σχεσιακά συστήματα. Συγκεκριμένα σε πολυμεσικά δεδομένα το γεγονός αυτό είναι πολύ σημαντικό, καθώς όπως είδαμε και στην περίπτωσή μας, τα δεδομένα που έχουμε προς αποθήκευση και αναζήτηση (περιγραφείς σκηνών), προκύπτουν ύστερα από επεξεργασία των αρχικών δεδομένων (3d σκηνές) σύμφωνα με κάποιο πρότυπο. Συνεπώς, ανάλογα με το πρότυπο που χρησιμοποιείται κάθε φορά αλλά και την αρχική δομή της κάθε σκηνής, μπορεί τελικά να έχουμε περιγραφείς με αρκετές διαφορές μεταξύ τους ως προς τη δομή. Ακόμα, τα τελευταία χρόνια γίνεται κατανοητό πως για να καταφέρουμε να διαχειριστούμε αποδοτικά τεράστιους όγκους δεδομένων, δεν αρκεί ένα υπολογιστικό σύστημα με 1, 2, 4, ή 8 επεξεργαστές, αλλά χρειαζόμαστε μεγάλα κατανεμημένα συστήματα με εκατοντάδες επεξεργαστές. Η κατανεμημένη αποθήκευση δεδομένων όμως, συνάδει απόλυτα με τη χρήση μη σχεσιακών βάσεων. Η MongoDB πιο συγκεκριμένα λόγω των πολύ υψηλών δυνατοτήτων που έχει όσον αφορά την κλιμάκωση αλλά και λόγω του aggregation framework που παρέχει, αποτελεί την καταλληλότερη επιλογή για την περίπτωσή μας, ενώ με τα εργαλεία που έχουν αναπτυχθεί πρόσφατα, όπως είδαμε για παράδειγμα το map reduce, μας παρέχει τη δυνατότητα να εκτελέσουμε πολύπλοκα ερωτήματα σε πολύ μικρό χρόνο.

Πλέον, όντας σε θέση να διαχειριστούμε αποδοτικά πολύ μεγάλο όγκο πολυμεσικών δεδομένων, ανοίγονται πολλοί νέοι ορίζοντες τόσο στον τομέα των πολυμέσων όσο και σε τομείς όπως το machine learning και η τεχνητή νοημοσύνη. Σε

πρώτο στάδιο, στον τομέα των πολυμέσων παρέχεται πλέον η δυνατότητα για content based αναζητήσεις. Μπορούμε δηλαδή να εκτελέσουμε ερωτήματα σχετικά με το περιεχόμενο μιας 3d σκηνής (χρώματα, σχήματα, όγκοι, επιφάνειες κλπ.) πολύ πιο αποδοτικά. Σε δεύτερο στάδιο, το γεγονός αυτό μπορεί να χρησιμοποιηθεί ως βάση για την εξαγωγή meta-πληροφοριών. Για παράδειγμα μας δίνεται η δυνατότητα να εξάγουμε διάφορα patterns για τις τιμές συγκεκριμένων πεδίων ενός τύπου αντικειμένου και να τα χρησιμοποιήσουμε για την κατηγοριοποίηση και ταυτοποίηση τους. Η δυνατότητα αυτή θα μπορούσε σε επόμενο στάδιο να χρησιμοποιηθεί στον τομέα της τεχνητής νοημοσύνης και της ρομποτικής για την αναγνώριση αντικειμένων σε τρισδιάστατο χώρο.

4.6 Σύνοψη

Σε αυτή την εργασία παρουσιάσαμε την αρχιτεκτονική και την υλοποίηση μιας τεχνικής για αποδοτική αποθήκευση και αναζήτηση μεγάλου όγκου πολυμεσικών δεδομένων σε συστήματα υπολογιστικού νέφους.

Πιο συγκεκριμένα, αρχικά κάναμε μια θεωρητική ανάλυση των χρησιμοποιούμενων τεχνολογιών (Cloud Computing, NoSQL Databases, Big Data) και εργαλείων που χρησιμοποιούμε (Mondo DB, MapReduce κλπ.) ώστε να δοθεί στον αναγνώστη το απαραίτητο θεωρητικό υπόβαθρο για την περαιτέρω κατανόηση του αντικειμένου της εργασίας. Στο πρακτικό κομμάτι, υλοποιήσαμε την απαιτούμενη υποδομή ώστε από ένα dataset με 3d σκηνές, να εξάγουμε τις απαραίτητες mp7 περιγραφές και να τις αποθηκεύσουμε με βέλτιστο τρόπο σε μια MongoDB βάση. Στηριζόμενοι στη βάση αυτή που δημιουργήσαμε, εκτελέσαμε ένα query που συναντάται με πολύ μεγάλη συχνότητα σε multimedia συστήματα αλλά και ταυτόχρονα έχει μεγάλες υπολογιστικές απαιτήσεις. Το ερώτημα αυτό έχει να κάνει με την κατανομή των χρωμάτων στις σκηνές που περιέχονται στη βάση. Το ιστόγραμμα χρωμάτων που εξάγαμε αποτελεί τον απλούστερο τρόπο να δούμε το χρωματικό περιεχόμενο των σκηνών χωρίς την απώλεια σημαντικής πληροφορίας. Εκτελέσαμε το query αυτό με δύο τρόπους. Στον πρώτο κάνουμε χρήση της παραδοσιακής τεχνικής αναζήτησης δεδομένων στην MongoDB, όπου βρίσκουμε κάθε φορά το json αρχείο-περιγραφή της σκηνής, και το διατρέχουμε ώστε να κρατήσουμε τις τιμές των πεδίων που μας ενδιαφέρουν. Στο δεύτερο κάνουμε χρήση της τεχνικής map reduce για κατανεμημένη εκτέλεση ερωτημάτων. Τα αποτελέσματα από την εκτέλεση και των δύο μεθόδων δείχνουν πως η τεχνική αυτή αποτελεί μία πολύ καλή λύση για την εξαγωγή χρήσιμης πληροφορίας σε συστήματα πολυμέσων.

Βιβλιογραφία

[1] R. Lämmel, "Google's MapReduce Programming Model — Revisited".
[2] P. A. Bernstein, V. Hadzilacos and N. Goodman, "Concurrency Control and Recovery in Database Systems".
[3] S. Gilbert and N. A. Lynch, "Perspectives on the CAP Theorem".
[4] Y. Saito and M. Shapiro, "Optimistic Replication".
[5] J.-H. Kang, C.-S. Kim and E.-J. Ko, "An XQuery engine for digital library systems".
[6] D. Tjondronegoro and Y.-P. P. Chen, "Content-Based Indexing and Retrieval Using MPEG-7 and X-Query in Video Data Management Systems".
[7] N. Fatemi, M. Lalmas and T. Rölleke, "How to retrieve multimedia documents described by MPEG-7".
[8] M. Yoshikawa and T. Amagasa, "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases".
[9] Y. Chu, L.-T. Chia and S. S. Bhowmick, "Looking at Mapping, Indexing & Querying of MPEG-7 Descriptors in RDBMS with SM3".

Παράρτημα

```
/*
 *
 *
Copyright (C) 2014  Georgios Tentes

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to
deal in the Software without restriction, including without limitation the
rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
sell copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN THE SOFTWARE.
*/


package Diplomatikh;



import java.io.File;

import java.io.IOException;

import java.net.UnknownHostException;

import java.util.Scanner;


import org.json.JSONException;

import org.json.JSONObject;

import org.json.XML;


import com.mongodb.BasicDBObject;

import com.mongodb.DB;

import com.mongodb.DBCollection;

import com.mongodb.DBCursor;

import com.mongodb.DBObject;

import com.mongodb.Mongo;
```

```
import com.mongodb.MongoException;
import com.mongodb.gridfs.GridFS;
import com.mongodb.gridfs.GridFSDBFile;
import com.mongodb.gridfs.GridFSInputFile;
import com.mongodb.util.JSON.*;
import java.nio.file.Path;




/**
 *
 * @author George
 */

/*Dino os orisma to path tou zip kai dhmiourgeitai to antistoixo bucket sto
gridfs */


public class Create_Buckets {


   private static final String dbName = "Savage";
   private static final String LOCALHOST = "Localhost";
   private static final String MP7 = "mp7";
   private static final String ZIP = "zip";
   private static final String collectionName = "Savage.files";


    public static void main(String[] args) throws JSONException {


     try {
            Mongo mongo = new Mongo(LOCALHOST, 27017);
            DB Mydb = mongo.getDB(dbName);


            String Path = args[0];
            int index = Path.lastIndexOf('\\');
            String newFileName = Path.substring(index+1);
            String mp7File="";


            for(File f: new File(Path).listFiles()){
                if(f.getName().endsWith(MP7))
                    mp7File =  f.getPath();
```

```
            }


            File zipFile = new File(Path + "//" + newFileName+"."+ ZIP);
            System.out.println("File: " + Path + newFileName + "." + ZIP +
"\n " + MP7 + " " + mp7File);
            GridFS gfs = new GridFS(Mydb, dbName);
            GridFSInputFile gfsFile = gfs.createFile(zipFile);
            gfsFile.setFilename(newFileName);
            gfsFile.save();                  //Apothikevo to arxeio


            /*
            *Eisago to mp7 sta metadata tou arxeiou se morfh json, stin
files collection
            */



            /*testing with reading xml as a string*/
            String text = new Scanner( new File(mp7File)
).useDelimiter("\\A").next();
            System.out.println(""+text);
            JSONObject xmlJSONObj = XML.toJSONObject(text);
            BasicDBObject toAppend = new BasicDBObject();


            Object o = com.mongodb.util.JSON.parse(xmlJSONObj.toString());
            DBObject dbObj = (DBObject) o;


            toAppend.append("$set", new BasicDBObject().append("metadata",
o));
            DBCollection c = Mydb.getCollection(collectionName);


            BasicDBObject query = new BasicDBObject();
            query.append("filename", newFileName);
            c.update(query, toAppend);
        }
         catch (UnknownHostException e) {
     e.printStackTrace();
   } catch (MongoException e) {
     e.printStackTrace();
   } catch (IOException e) {
     e.printStackTrace();
```

```
    }
      }
}
```

```
package Diplomatikh;

import java.io.File;
import java.io.IOException;

import org.apache.commons.io.FileUtils;


public class CreateZips {

    private static String savageHome = "C:\\Diplomatikh\\SavageSample";
    private static String zipsHome = "C:\\Diplomatikh\\zipsSample";

    public static void main(String args[]){
        //inputPath: Savage/AircraftFixedWing/AV8B-Harrier-
UnitedStates/[mp7_name].mp7
        //     homeFolder/lvl1/lvl2
        //outputPath: zips/AircraftFixedWing/AV8B-Harrier-UnitedStates/AV8B-
Harrier-UnitedStates[1..9]/AV8B-Harrier-UnitedStates1.zip and AV8B-Harrier-
UnitedStates1_[mp7_name]
        File savageHomeFolder = new File(savageHome);
        for(File lvl1 : savageHomeFolder.listFiles()){
            for(File lvl2 : lvl1.listFiles()){
                        int i = 0;
                if(lvl2.listFiles() != null){
                for(File mp7File : lvl2.listFiles()){
                    if(mp7File.getName().endsWith(".mp7")){
                        System.out.println("Found mp7\n");
                        i++;
                        String inputFolderPath = lvl2.getPath();
                        String inputMP7Name = mp7File.getName();
                        String outputZipPath = zipsHome + "/" +
lvl1.getName() + "/" + lvl2.getName() + "/" + lvl2.getName() + i;
                        String outputZipName = lvl2.getName() + i + ".zip";
                        //LocalTransformer.main(new String[]{inputX3DPath,
inputX3DName, outputX3DPath});
                        System.out.println(inputFolderPath + ", " +
inputMP7Name + ", " + outputZipPath);
                        File zipFile = new File(outputZipPath);
                        if(!zipFile.exists()){
                            zipFile.mkdirs();
                        }
                        AppZip.main(new String[]{inputFolderPath,
outputZipPath + "/" + outputZipName});
                        File sourceMp7 = new File(inputFolderPath + "/" +
inputMP7Name);
                        File dest = new File(outputZipPath);
                        try {
                            FileUtils.copyFileToDirectory(sourceMp7, dest);
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                        //arg0 = inputX3D, arg1 = inputX3Dname, arg2 =
outputPath
                    }
                    else{
                    //    System.out.println(""+mp7File.getName());
                    }
                }
            }
            }
        }
    }
 }
```

```
package Diplomatikh;


import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipOutputStream;

public class ZipHelper{

    List<String> fileList;
    private static String OUTPUT_ZIP_FILE;
    private static String SOURCE_FOLDER;

    ZipHelper(){
    fileList = new ArrayList<String>();
    }

    public static void main( String[] args )
    {
        //args[0] - Source Folder, args[1] - Output Zip File (Path+Name)
        ZipHelper ZipHelper = new ZipHelper();
        SOURCE_FOLDER = args[0];
        OUTPUT_ZIP_FILE = args[1];
        ZipHelper.generateFileList(new File(SOURCE_FOLDER));
        ZipHelper.zipIt(OUTPUT_ZIP_FILE);
    }

    /**
     * Zip it
     * @param zipFile output ZIP file location
     */
    public void zipIt(String zipFile){
```

```
     byte[] buffer = new byte[1024];

     try{

        FileOutputStream fos = new FileOutputStream(zipFile);
        ZipOutputStream zos = new ZipOutputStream(fos);

        System.out.println("Output to Zip : " + zipFile);

        for(String file : this.fileList){

            System.out.println("File Added : " + file);
            ZipEntry ze= new ZipEntry(file);
            zos.putNextEntry(ze);

            FileInputStream in =
                       new FileInputStream(SOURCE_FOLDER + File.separator +
file);

            int len;
            while ((len = in.read(buffer)) > 0) {
                zos.write(buffer, 0, len);
            }

            in.close();
        }

        zos.closeEntry();
        //remember close it
        zos.close();

        System.out.println("Done");
    }catch(IOException ex){
       ex.printStackTrace();
    }
   }

    /**
```

```
     * Traverse a directory and get all files,
     * and add the file into fileList
     * @param node file or directory
     */
    public void generateFileList(File node){

        //add file only
    if(node.isFile()){
        fileList.add(generateZipEntry(node.getAbsoluteFile().toString()));
    }

    if(node.isDirectory()){
        String[] subNote = node.list();
        for(String filename : subNote){
            generateFileList(new File(node, filename));
        }
    }


    }

    /**
     * Format the file path for zip
     * @param file file path
     * @return Formatted file path
     */
    private String generateZipEntry(String file){
        return file.substring(SOURCE_FOLDER.length()+1, file.length());
    }
}
```

```
package Diplomatikh;




import java.io.File;


import org.json.JSONException;


public class PopulateMongo {


    private static String zipsHome = "C:\\Diplomatikh\\zips";


    public static void main(String args[]) throws JSONException{

        //path: zips/AircraftFixedWing/AV8B-Harrier-UnitedStates/AV8B-
Harrier-UnitedStates1/AV8B-Harrier-UnitedStates1.zip

        //     zipsHome/lvl1/lvl2/lvl3/zipFile + mp7File

        //Ta paths eksartontai apo to pos tha zipparo ta arxeia, diladh to
deftero scrip

        File zipsHomeFolder = new File(zipsHome);

        for(File lvl1 : zipsHomeFolder.listFiles()){    //AircraftFixedWing
etc

            for(File lvl2 : lvl1.listFiles()){  //AV8B-Harrier-UnitedStates
etc

                for(File lvl3 : lvl2.listFiles()){ //AV8B-Harrier-
UnitedStates1 etc

                    String path = lvl3.getPath();

                    Create_Buckets.main(new String[]{path});

                    //LocalTransformer.main(new String[]{inputX3DPath,
inputX3DName, outputX3DPath});

                    System.out.println(path);

                    }

                }

            }

        }

}
```

```
package Diplomatikh;


import java.io.File;
import java.io.IOException;
import java.net.UnknownHostException;

import com.mongodb.DB;
import com.mongodb.Mongo;
import com.mongodb.gridfs.GridFS;
import com.mongodb.gridfs.GridFSDBFile;

import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;


/**
 *
 * @author Giorgos
 */


public class RetrieveFile {

   private static final String server = "Localhost";
   private static final int port = 27017;
   private static final String dbName  = "Savage";
   private static DB db;
   private static String bucketName = "Savage";


   private String createQuery(String key, String Value){

      if(key.equals("filename"))
         return "{ filename : " + Value + " } ";
      if(key.equals("id"))
```

```
      return "{ _id : " + Value + " }";
   else{
      throw new IllegalArgumentException();
   }
}


private List<GridFSDBFile> getFileById(String id){
   String query = createQuery("id", id);
      GridFS gfsZip = new GridFS(db, bucketName);
      List<GridFSDBFile> ZipForOutput = gfsZip.find(query);
      return ZipForOutput;
}


private List<GridFSDBFile> getFileByName(String name){
   String query = createQuery("filename", name);
      GridFS gfsZip = new GridFS(db, bucketName);
      List<GridFSDBFile> ZipForOutput = gfsZip.find(query);
      return ZipForOutput;
}


public DB getDB() throws UnknownHostException{

   if(db == null){
       Mongo mongo = new Mongo(server, 27017);
          db = mongo.getDB(dbName);
   }

   return db;

}


 public static void main(String[] args){

   try {
```

```
            RetrieveFile getFile = new RetrieveFile();
            String newFileName =
args[0].substring(args[0].lastIndexOf('/')+1, args[0].lastIndexOf('.'));
            //debugging
            System.out.println(newFileName);
            //Start time measurement
            long startTime = System.currentTimeMillis();



            long stopTime = System.currentTimeMillis();


            //End time measurement
            (getFile.getFileByName(newFileName)).get(0).writeTo(new
File("/tmp/zips/" + newFileName));


            long elapsedTime = stopTime - startTime;
            System.out.println(elapsedTime);


          }
         catch (UnknownHostException e) {
           e.printStackTrace();
         } catch (IOException ex) {
            Logger.getLogger(RetrieveFile.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
}
```

```
package Diplomatikh;



/**
 *
 * @author George
 */
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;
import org.exist.xquery.modules.mpeg7.LocalTransformer;

public class TransformationDriver {



    // /home/ntua/Diplomatiki/Savage   to kanoniko
    private static final String C = "C:\\";
    private static final String savageHome = "C:\\Diplomatikh\\Savage";
    private static File DiplomatikhHomeFolder;


    public static void main(String args[]) throws IOException{
        //path: Savage/AircraftFixedWing/AV8B-Harrier-UnitedStates/AV8B-
Harrier-UnitedStates.x3d
        //     homeFolder/lvl1/lvl2/x3dFile
        File savageHomeFolder = new File(savageHome);
        File CFolder = new File(C);


        PrintStream out = new PrintStream(new
FileOutputStream("C:\\Diplomatikh\\output.txt"));
        System.setOut(out);


        for(File ff : CFolder.listFiles())
            if(ff.getName().equals("Diplomatikh"))
                DiplomatikhHomeFolder = new File(ff.getPath());
```

```
        for(File ff : DiplomatikhHomeFolder.listFiles())
            if(ff.getName().equals("Savage"))
                savageHomeFolder = new File(ff.getPath());

        System.out.println(""+savageHomeFolder.getPath());
        for(File lvl1 : savageHomeFolder.listFiles()){
            if(!lvl1.getName().equals("Tools"))
                continue;
          //  System.out.println(""+lvl1.getPath());
            for(File lvl2 : lvl1.listFiles()){
                        if(lvl2.getName().equals("Animation"))
                            continue;

          //      System.out.println(""+lvl2.getPath());
                if(lvl2.listFiles() != null){
                    for(File x3dFile : lvl2.listFiles()){
               //
System.out.println(""+x3dFile.getName());
                        if(x3dFile.getName().endsWith(".x3d")){
                            System.out.println(""+x3dFile.getPath());
                            String inputX3DPath = lvl2.getPath();
                            String inputX3DName = x3dFile.getName();
                            String outputX3DPath = lvl2.getPath();
                            try{
                                LocalTransformer.main(new
String[]{inputX3DPath+"\\", inputX3DName, outputX3DPath+"\\"});
                            }
                            catch(Exception ex){
                                ex.printStackTrace();;
                            }
                     //        System.out.println(inputX3DPath + ", " +
inputX3DName + ", " + outputX3DPath);
                            //arg0 = inputX3D, arg1 = inputX3Dname, arg2 =
outputPath
                        }
                    //     System.out.println("\n");

                    }
                }
            }
```

```
            }
        }
    }
```

```
package Diplomatikh;


import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonArray;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.Mongo;
import com.mongodb.gridfs.GridFS;
import com.mongodb.gridfs.GridFSDBFile;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.UnknownHostException;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.jaxen.function.SumFunction;
import org.json.JSONException;
import org.json.JSONObject;

/**
 *
 * @author Giorgos
 */

public class Query1NoMR {

   private static final int NUM_CELLS = 512;
```

```
    public static void main(String[] args) throws JSONException{

     try {
            Mongo mongo = new Mongo("localhost", 27017);
            DB Mydb = mongo.getDB("X3D");

            BasicDBObject info = new BasicDBObject();

            DBCollection collection = Mydb.getCollection("Savage.files");

            BasicDBObject qu = new BasicDBObject();
            qu.append("filename", "Bear-Russia1");
            DBCursor cursor = collection.find();
            JSONObject obj = new JSONObject();
            Gson gson = new GsonBuilder().create();
            double length, height, depth, width, volume;
            int cell;
            double[] histogram = new double[NUM_CELLS];
            double[] sceneHistogram = new double[NUM_CELLS];
            int[] numOccurences = new int[NUM_CELLS];
            double sceneVolumeSum = 0;

            while(cursor.hasNext()){//return the file specified by
"filename"
             info = (BasicDBObject) cursor.next().get("metadata");     //get
the file metadata - mp7 description
             obj = new JSONObject(info.toString());
          //    System.out.println(obj.toString());
             sceneHistogram = new double[NUM_CELLS];
             for(int i = 0; i < sceneHistogram.length; i++){
                sceneHistogram[i] = 0.0;
             }
             sceneVolumeSum = 0.0;
            JsonParser jsonParser = new JsonParser( );
            try{
            JsonArray DescriptorCollection =
jsonParser.parse(obj.toString())
                .getAsJsonObject().get("Mpeg7")
                .getAsJsonObject().getAsJsonArray("Description").get(1)
                .getAsJsonObject().get("MultimediaContent")
```

```
                .getAsJsonObject().get("StructuredCollection")
                .getAsJsonObject().getAsJsonArray("Collection").get(1)
                .getAsJsonObject().getAsJsonArray("DescriptorCollection");
  //j


               length = DescriptorCollection.size();
               for(int i = 0; i < length; i++){
                try{
                   JsonElement BoundingBox3DSize =
DescriptorCollection.getAsJsonArray().get(i).getAsJsonObject()

.getAsJsonObject().getAsJsonArray("Descriptor").get(0)

.getAsJsonObject().get("BoundingBox3DSize");


                   height =
BoundingBox3DSize.getAsJsonObject().get("BoxHeight").getAsDouble();
                   width =
BoundingBox3DSize.getAsJsonObject().get("BoxWidth").getAsDouble();
                   depth =
BoundingBox3DSize.getAsJsonObject().get("BoxDepth").getAsDouble();
                   volume = height * width * depth;
                   //System.out.println("volume: " + volume);



  //this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollectio
n.Collection[1].DescriptorCollection[j].Descriptor[1].Geometry3D.DominantCol
or3D.Value.Index;"
                   JsonElement RGB =
DescriptorCollection.getAsJsonArray().get(i).getAsJsonObject()
                     .getAsJsonObject().getAsJsonArray("Descriptor").get(1)
                     .getAsJsonObject().get("Geometry3D")
                     .getAsJsonObject().get("DominantColor3D")
                     .getAsJsonObject().get("Value")
                     .getAsJsonObject().get("Index");


                   //Since we reached here, we have gotten both the volume
and the color
                   //Otherwise an exception would have been thrown
                   String[] RGBarr = RGB.getAsString().split(" ");
                   System.out.println(RGB.getAsString());
                   cell = (int) (Math.floor((double)
Integer.parseInt(RGBarr[0]) / 32) + 8 * Math.floor((double)
Integer.parseInt(RGBarr[1]) / 32) + 64 * Math.floor((double)
Integer.parseInt(RGBarr[2]) / 32) );
```

```
                    System.out.println(cell);
                    sceneHistogram[cell] += volume;
                    numOccurences[cell] ++;
                    sceneVolumeSum += volume;
                  }
                  catch(Exception e1){
            //         e1.printStackTrace();
                  }
                 }
               if(sceneVolumeSum == 0.0){
                  System.out.println("Something went wrong");
                  continue;
               }
                 for(int i = 0; i < NUM_CELLS; i++){
                  sceneHistogram[i] /= sceneVolumeSum;
                  histogram[i] += sceneHistogram[i];
                 }
              }
              catch(Exception e){
               e.printStackTrace(); //fields don't exist
              }


              }


            PrintWriter writer = new PrintWriter("Query1NoMR.txt");
              for(int i = 0; i < NUM_CELLS; i++){
               histogram[i] /= numOccurences[i];
               writer.println("[" + i +"] : " + histogram[i] + " occs: " +
numOccurences[i]);
               System.out.println("[" + i +"] : " + histogram[i]);
              }
            writer.close();
      }
          catch (UnknownHostException e) {
      //e.printStackTrace();
    } catch (IOException ex) {
              Logger.getLogger(RetrieveMp7.class.getName()).log(Level.SEVERE,
null, ex);
         }
```

```
        }
    }
```

```
package Diplomatikh;


import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonArray;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.Mongo;
import com.mongodb.gridfs.GridFS;
import com.mongodb.gridfs.GridFSDBFile;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.UnknownHostException;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.jaxen.function.SumFunction;
import org.json.JSONException;
import org.json.JSONObject;

/**
 *
 * @author Giorgos
 */
public class Query2NoMR {



   private static final int NUM_CELLS = 512;
```

```
    public static void main(String[] args) throws JSONException{


     try {
            Mongo mongo = new Mongo("localhost", 27017);
            DB Mydb = mongo.getDB("X3D");


            BasicDBObject info = new BasicDBObject();


            DBCollection collection = Mydb.getCollection("Savage.files");


            BasicDBObject qu = new BasicDBObject();
            qu.append("filename", "Bear-Russia1");
            DBCursor cursor = collection.find();
            JSONObject obj = new JSONObject();
            Gson gson = new GsonBuilder().create();
            double length, height, depth, width, volume;
            int cell;
            double[] histogram = new double[NUM_CELLS];
            double[] sceneHistogram = new double[NUM_CELLS];
            int[] numOccurences = new int[NUM_CELLS];
            double sceneVolumeSum = 0;


            long totalCount = 0;  //Synolikos arithmos twn skhnwn


            while(cursor.hasNext()){//return the file specified by
"filename"
             info = (BasicDBObject) cursor.next().get("metadata");     //get
the file metadata - mp7 description
             obj = new JSONObject(info.toString());
          //    System.out.println(obj.toString());
             sceneHistogram = new double[NUM_CELLS];
             for(int i = 0; i < sceneHistogram.length; i++){
                sceneHistogram[i] = 0.0;
             }
             sceneVolumeSum = 0.0;
            JsonParser jsonParser = new JsonParser( );
            try{
            JsonArray DescriptorCollection =
jsonParser.parse(obj.toString())
```

```
                .getAsJsonObject().get("Mpeg7")
                .getAsJsonObject().getAsJsonArray("Description").get(1)
                .getAsJsonObject().get("MultimediaContent")
                .getAsJsonObject().get("StructuredCollection")
                .getAsJsonObject().getAsJsonArray("Collection").get(1)
                .getAsJsonObject().getAsJsonArray("DescriptorCollection");
  //j


              length = DescriptorCollection.size();
              for(int i = 0; i < length; i++){
               try{
                  JsonElement BoundingBox3DSize =
DescriptorCollection.getAsJsonArray().get(i).getAsJsonObject()

.getAsJsonObject().getAsJsonArray("Descriptor").get(0)

.getAsJsonObject().get("BoundingBox3DSize");


                  height =
BoundingBox3DSize.getAsJsonObject().get("BoxHeight").getAsDouble();
                  width =
BoundingBox3DSize.getAsJsonObject().get("BoxWidth").getAsDouble();
                  depth =
BoundingBox3DSize.getAsJsonObject().get("BoxDepth").getAsDouble();
                  volume = height * width * depth;
                  //System.out.println("volume: " + volume);



  //this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollectio
n.Collection[1].DescriptorCollection[j].Descriptor[1].Geometry3D.DominantCol
or3D.Value.Index;"
                  JsonElement RGB =
DescriptorCollection.getAsJsonArray().get(i).getAsJsonObject()
                    .getAsJsonObject().getAsJsonArray("Descriptor").get(1)
                    .getAsJsonObject().get("Geometry3D")
                    .getAsJsonObject().get("DominantColor3D")
                    .getAsJsonObject().get("Value")
                    .getAsJsonObject().get("Index");


                  //Since we reached here, we have gotten both the volume
and the color
                  //Otherwise an exception would have been thrown
                  String[] RGBarr = RGB.getAsString().split(" ");
                  System.out.println(RGB.getAsString());
```

```
                    cell = (int) (Math.floor((double)
Integer.parseInt(RGBarr[0]) / 32) + 8 * Math.floor((double)
Integer.parseInt(RGBarr[1]) / 32) + 64 * Math.floor((double)
Integer.parseInt(RGBarr[2]) / 32) );

                    System.out.println(cell);

                    sceneHistogram[cell] += volume;

                    sceneVolumeSum += volume;

                    totalCount++;

                  }

                  catch(Exception e1){

            //          e1.printStackTrace();

                  }

                 }

               if(sceneVolumeSum == 0.0){

                  System.out.println("Something went wrong");

                  continue;

               }

                 for(int i = 0; i < NUM_CELLS; i++){

                  sceneHistogram[i] /= sceneVolumeSum;

                  histogram[i] += sceneHistogram[i];

                 }

              }

              catch(Exception e){

               e.printStackTrace(); //fields don't exist

              }


              }


            PrintWriter writer = new PrintWriter("Query2NoMR.txt");

              for(int i = 0; i < NUM_CELLS; i++){

               histogram[i] /= totalCount;

               //writer.println("[" + i +"] : " + histogram[i] + " totalCount:
" + totalCount);

               System.out.println("[" + i +"] : " + histogram[i]);

              }

            writer.close();

      }

          catch (UnknownHostException e) {

      //e.printStackTrace();

   } catch (IOException ex) {
```

```
            Logger.getLogger(RetrieveMp7.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
}
```

```
package Diplomatikh;

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.net.UnknownHostException;

import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.MapReduceCommand;
import com.mongodb.MapReduceOutput;
import com.mongodb.MongoClient;

public class QueryMR1 {

   private String dbName;
   private DB db;
   private MongoClient mongoClient;

   private static final String map1 = "function(){"
         + "try{"
         + "var length =
this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Co
llection[1].DescriptorCollection.length;"
         + "var i = 0;"
         + "var toEmit;"
         + "var UNDERFLOW  = 0.000000001;"
         + "var OVERFLOW  = 10000000;"
         + "var toEmit2 = [];"
         + "for(var j = 0; j < length; j++){"
            +"try {"
            + "var v1 =
this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Co
llection[1].DescriptorCollection[j].Descriptor[0].BoundingBox3DSize.BoxHeigh
t;"
            + "var v2 =
this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Co
llection[1].DescriptorCollection[j].Descriptor[0].BoundingBox3DSize.BoxDepth
;"
            + "var v3 =
this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Co
llection[1].DescriptorCollection[j].Descriptor[0].BoundingBox3DSize.BoxWidth
;"
            + "var volume = v1*v2*v3;"
            + "if(volume < UNDERFLOW || volume > OVERFLOW)"
            + "  volume = 0;"
            + "var RGB =
this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Co
llection[1].DescriptorCollection[j].Descriptor[1].Geometry3D.DominantColor3D
.Value.Index;"
            + "var colors = RGB.split(' ');"
            + "var dim1 = Math.floor(colors[0] / 32);"
            + "var dim2 = Math.floor(colors[1] / 32);"
            + "var dim3 = Math.floor(colors[2] / 32);"
            + "var cell = dim3 * 64 + dim2 * 8 + dim1;"
            + "toEmit =  { cell: cell, volume: volume};"
            + "if(length == 2){"
      //    + "     print(\"length == 1\");"
            + "     toEmit2.push(toEmit);"
            + "     if(!isNaN(toEmit2.volume))"
            + "        emit(this.filename, toEmit2);"
            + "     toEmit2.pop();"
            + "  }"
            + "else{"
            + "     if(!isNaN(toEmit.volume))"
```

```
            + "        emit(this.filename, toEmit);"
            + "   }"
            + "}catch(err){"
//             + "     print(err);"
            + "}"
            + "}"
            + "}"
            + "catch(err2){"
   //         + "     print(err2);"
            + "}"
            + "};";


   private static final String reduce1 = "function(k, values) {"
                + "var sum = 0;"
                + "var maxINT = 100000000;"
                + "var flag = 0;"
                + "var vals = [];"
          //    + "print(\"Executing reduce 1\");"
                + "for (var i = 0; i < values.length; i++){"
                + "   vals.push(values[i]);"
                + "}"
                + "for (var i = 0; i < values.length; i++){"
                + "     if( sum + vals[i].volume > maxINT){"
          //    + "        print(\"Potential Overflow\");"
                + "        sum = maxINT;"
                + "     }"
                + "     else"
                + "        sum += vals[i].volume;"
                + "}"
                + "   for (var i = 0; i < values.length; i++){"
                + "     vals[i].volume = vals[i].volume / sum;"
                + "     if(vals[i].volume <= 0 || vals[i].volume >= 1){"
                + "        flag = 1;"
                + "     }"
                + "    } "
                + "var ret = {v : values} ;"
                + "if(flag !== 0)"
                + "     print(\"Something went wrong\");"
                + "return (k, ret);"
                + "};";

   private static final String map2 = "function(){"
        + "try{"
        + "var OVERFLOW = 1;"
        + "if (typeof(this.value.v) !== 'undefined'){"
    //    + "     print(\"value.v\");"
        + "     length = this.value.v.length;"
        + "     for(var j = 0; j < length; j++){"
        + "        if(this.value.v[j].volume >= 0 && this.value.v[j].volume
<= 1)"
        + "           emit(this.value.v[j].cell, this.value.v[j].volume);"
        + "     }"
        + "   }"
        + "else{"
   //     + "     print(\"value.volume\");"
    //    + "        if(this.value.volume >= 0 && this.value.volume <= 1)"
    //    + "        emit(this.value.cell, this.value.volume);"
        + "   }"
        + "}"
        + "catch(err){"
     // + "print(err);"
        + "}"
        + "};";


   //eksago gia kathe keli to average pososto tou
   private static final String reduce2 = "function(k, values) {"
```

```
                + "var sum = 0;"
                + "var count = 0;"
                + "var MAX_INT = 100000000;"
                + "for (var i = 0; i < values.length; i++){"
                + "  if(!isNaN(values[i])){"

                + "     if( sum + values[i] > MAX_INT){"
      //        + "        print(\"Potential Overflow\");"
                + "        sum = max_INT;"
                + "     }"
                + "     else{"
                + "        sum += values[i];"
                + "     }"
                + "     count++;"
                + "  }"
                + "}"
                + "return (k, sum / count);"
                + "};";



   private static final String outputCollection1 = "out1";
   private static final String outputCollection2 = "out2";


   private void executeQuery(BasicDBObject searchQuery, String
collectionName, String outputCollection ,String map, String reduce) throws
FileNotFoundException{

           MapReduceCommand mr = new
MapReduceCommand(db.getCollection(collectionName), map, reduce,
outputCollection, MapReduceCommand.OutputType.REPLACE, searchQuery);
         //  mr.setFinalize(finalize1);
           MapReduceOutput out =
db.getCollection(collectionName).mapReduce(mr);
           for (DBObject o : out.results()) {
              System.out.println(o.toString());
           }
      }

   public QueryMR1(MongoClient mongoClient, String dbName){
      this.dbName = dbName;
      this.mongoClient = mongoClient;
         this.db = mongoClient.getDB(dbName);
   }

   public static void main(String[] args) throws Exception{

           MongoClient mongoClient = new MongoClient("localhost", 27017);

           QueryMR1 mc = new QueryMR1(mongoClient, "Savage");

           long startTime = System.currentTimeMillis();
           BasicDBObject searchQuery = new BasicDBObject();
           //searchQuery.
          // searchQuery.put("filename", "Bear-Russia1");
           mc.executeQuery(searchQuery, "Savage.files", outputCollection1,
map1, reduce1);
           mc.executeQuery(searchQuery, outputCollection1, outputCollection2,
map2, reduce2);

           long endTime = System.currentTimeMillis();
           System.out.println(endTime - startTime);
   }
}
```

```
package Diplomatikh;

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.net.UnknownHostException;

import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.MapReduceCommand;
import com.mongodb.MapReduceOutput;
import com.mongodb.MongoClient;

public class QueryMR2 {

   private String dbName;
   private DB db;
   private MongoClient mongoClient;

   private static final String map1 = "function(){"
         + "try{"
         + "var length =
this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Co
llection[1].DescriptorCollection.length;"
         + "var i = 0;"
         + "var toEmit;"
         + "var UNDERFLOW  = 0.000000001;"
         + "var OVERFLOW  = 10000000;"
         + "var toEmit2 = [];"
         + "for(var j = 0; j < length; j++){"
            +"try {"
            + "var v1 =
this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Co
llection[1].DescriptorCollection[j].Descriptor[0].BoundingBox3DSize.BoxHeigh
t;"
            + "var v2 =
this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Co
llection[1].DescriptorCollection[j].Descriptor[0].BoundingBox3DSize.BoxDepth
;"
            + "var v3 =
this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Co
llection[1].DescriptorCollection[j].Descriptor[0].BoundingBox3DSize.BoxWidth
;"
            + "var volume = v1*v2*v3;"
            + "if(volume < UNDERFLOW || volume > OVERFLOW)"
            + "   volume = 0;"
            + "var RGB =
this.metadata.Mpeg7.Description[1].MultimediaContent.StructuredCollection.Co
llection[1].DescriptorCollection[j].Descriptor[1].Geometry3D.DominantColor3D
.Value.Index;"
            + "var colors = RGB.split(' ');"
            + "var dim1 = Math.floor(colors[0] / 32);"
            + "var dim2 = Math.floor(colors[1] / 32);"
            + "var dim3 = Math.floor(colors[2] / 32);"
            + "var cell = dim3 * 64 + dim2 * 8 + dim1;"
            + "toEmit =  { cell: cell, volume: volume};"
            + "if(length == 2){"
      //    + "     print(\"length == 1\");"
            + "     toEmit2.push(toEmit);"
            + "     if(!isNaN(toEmit2.volume))"
            + "        emit(this.filename, toEmit2);"
            + "     toEmit2.pop();"
            + "   }"
            + "else{"
            + "     if(!isNaN(toEmit.volume))"
```

```
            + "        emit(this.filename, toEmit);"
            + "   }"
            + "}catch(err){"
//             + "     print(err);"
            + "}"
            + "}"
            + "}"
            + "catch(err2){"
   //         + "     print(err2);"
            + "}"
            + "};";


   private static final String reduce1 = "function(k, values) {"
                + "var sum = 0;"
                + "var maxINT = 100000000;"
                + "var flag = 0;"
                + "var vals = [];"
          //    + "print(\"Executing reduce 1\");"
                + "for (var i = 0; i < values.length; i++){"
                + "   vals.push(values[i]);"
                + "}"
                + "for (var i = 0; i < values.length; i++){"
                + "     if( sum + vals[i].volume > maxINT){"
          //    + "        print(\"Potential Overflow\");"
                + "        sum = maxINT;"
                + "     }"
                + "     else"
                + "        sum += vals[i].volume;"
                + "}"
                + "   for (var i = 0; i < values.length; i++){"
                + "     vals[i].volume = vals[i].volume / sum;"
                + "     if(vals[i].volume <= 0 || vals[i].volume >= 1){"
                + "        flag = 1;"
                + "     }"
                + "    } "
                + "var ret = {v : values} ;"
                + "if(flag !== 0)"
                + "     print(\"Something went wrong\");"
                + "return (k, ret);"
                + "};";

   private static final String map3 = "function(){"
          + "try{"
          + "var OVERFLOW = 1;"
          + "if (typeof(this.value.v) !== 'undefined'){"
     //     + "     print(\"value.v\");"
          + "     length = this.value.v.length;"
          + "     for(var j = 0; j < length; j++){"
          + "        if(this.value.v[j].volume >= 0 && this.value.v[j].volume
<= 1)"
          + "           emit(0, {cell: this.value.v[j].cell, volume:
this.value.v[j].volume});"
          + "     }"
          + "   }"
          + "else{"
    //    + "     print(\"value.volume\");"
          + "        if(this.value.volume >= 0 && this.value.volume <= 1)"
          + "        emit(0, {cell: this.value.cell, volume:
this.value.volume});"
          + "   }"
          + "}"
          + "catch(err){"
     //    + "print(err);"
          + "}"
          + "};";
```

```
   private static final String reduce3 = "function(k, values) {"
         + "var count = values.length;"
        // + "print(count);"
         + "var maxINT = 100000000;"
         + "var flag = 0;"
        //    + "print(\"Executing reduce 3\");"
         + "  for (var i = 0; i < values.length; i++){"
         + "     values[i].volume = values[i].volume / count;"
         + "     if(values[i].volume <= 0 || values[i].volume >= 1){"
         + "        flag = 1;"
         + "     }"
         + "   } "
         + "var ret = {v : values} ;"
         + "if(flag !== 0)"
        //    + "     print(\"Something went wrong\");"
         + "return (k, ret);"
         + "};";

   private static final String map4 = "function(){"
         + "try{"
         + "var OVERFLOW = 1;"
         + "if (typeof(this.value.v) !== 'undefined'){"
         //   + "     print(\"value.v\");"
         + "     length = this.value.v.length;"
         + "     for(var j = 0; j < length; j++){"
         + "        if(this.value.v[j].volume >= 0 && this.value.v[j].volume
<= 1)"
         + "           emit(this.value.v[j].cell, this.value.v[j].volume);"
         + "     }"
         + "  }"
         + "else{"
         //      + "     print(\"value.volume\");"
         + "        if(this.value.volume >= 0 && this.value.volume <= 1)"
         + "        emit(this.value.cell, this.value.volume);"
         + "  }"
         + "}"
         + "catch(err){"
        // + "print(err);"
         + "}"
         + "};";


   //eksago gia kathe keli to average pososto tou
   private static final String reduce4 = "function(k, values) {"
                + "var sum = 0;"
                + "var count = 0;"
                + "var MAX_INT = 100000000;"
                + "for (var i = 0; i < values.length; i++){"
                + "  if(!isNaN(values[i])){"
                + "     if( sum + values[i] > MAX_INT){"
              //    + "        print(\"Potential Overflow\");"
                + "        sum = max_INT;"
                + "     }"
                + "     else{"
                + "        sum += values[i];"
                + "     }"
                + "     count++;"
                + "  }"
                + "}"
                + "return (k, sum);"
                + "};";

   private static final String outputCollection1 = "out1";
   private static final String outputCollection2 = "out2";
   private static final String outputCollection3 = "out3";
   private static final String outputCollection4 = "out4";
```

```
   private void executeQuery(BasicDBObject searchQuery, String
collectionName, String outputCollection ,String map, String reduce) throws
FileNotFoundException{

           MapReduceCommand mr = new
MapReduceCommand(db.getCollection(collectionName), map, reduce,
outputCollection, MapReduceCommand.OutputType.REPLACE, searchQuery);
         //  mr.setFinalize(finalize1);
           MapReduceOutput out =
db.getCollection(collectionName).mapReduce(mr);
           //for (DBObject o : out.results()) {
           // System.out.println(o.toString());
           //}
     }

   public QueryMR2(MongoClient mongoClient, String dbName){
      this.dbName = dbName;
      this.mongoClient = mongoClient;
         this.db = mongoClient.getDB(dbName);
   }

   public static void main(String[] args) throws Exception{

           MongoClient mongoClient = new MongoClient("localhost", 27017);

           QueryMR2 mc = new QueryMR2(mongoClient, "X3D");

           long startTime = System.currentTimeMillis();
           BasicDBObject searchQuery = new BasicDBObject();
           //searchQuery.
          // searchQuery.put("filename", "Bear-Russia1");
           mc.executeQuery(searchQuery, "Savage.files", outputCollection1,
map1, reduce1);
           mc.executeQuery(searchQuery, outputCollection1, outputCollection2,
map3, reduce3);
           mc.executeQuery(searchQuery, outputCollection2, outputCollection3,
map4, reduce4);
           long endTime = System.currentTimeMillis();
           System.out.println(endTime - startTime);
   }
}
```

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.*;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;

import mpegMpeg7Schema2001.Mpeg7Document;

import org.apache.log4j.Logger;
import org.apache.log4j.Priority;
import org.apache.xmlbeans.XmlException;
import org.apache.xmlbeans.XmlOptions;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class Transformation {

    private static Logger logger = Logger.getLogger(Transformation.class);

    public static void main(String[] args) throws IOException, SAXException{

        File inXml = new File(args[0]);

        //*********************************************************
        //ARXH KODIKA GIA TO EXTRACT TON POINTS ENOS INDEXEDFACESET
        //*********************************************************
        String[] tokenizedCoord = null;
        String[] tokenizedPoint = null;

        String crossSectionAttrib = null;
        String spineAttrib = null;
        String scaleAttrib = null;
        String orientAttrib = null;

        List totalPointParts = null;
        File file = null;
        BufferedWriter bw = null;
        ArrayList<String> resultedIFSExtractionList = new ArrayList();
//mine
        ArrayList<String> resultedExtrExtractionList = new ArrayList();
        ArrayList<String> resultedExtrBBoxList = new ArrayList();
        StringBuilder IFSStringBuilder = new StringBuilder();
        StringBuilder ExtrShapeStringBuilder = new StringBuilder();
        StringBuilder ExtrBBoxStringBuilder = new StringBuilder();

        try {
            DocumentBuilderFactory dbFactory =
```

```
DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(inXml);
            doc.getDocumentElement().normalize();

            //if (doc.getElementsByTagName("IndexedFaceSet") != null &&
doc.getElementsByTagName("IndexedFaceSet").getLength() > 0) {
            int shapeCount = 0;
            int extrCount = 0;
            int ifsCount = 0;

            NodeList shpList = doc.getElementsByTagName("Shape");
            for (int temp = 0; temp < shpList.getLength(); temp++) {
                shapeCount += 1;
                Node nNode = shpList.item(temp);
                //System.out.println("Number of Shape in document: " +
temp);
                Element eElement = (Element) nNode;
                //System.out.println("eElement= " + eElement);
                if (eElement.getElementsByTagName("IndexedFaceSet").item(0)
!= null) {
                    ifsCount += 1;
                    if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                        Element ifsElem = (Element)
eElement.getElementsByTagName("IndexedFaceSet").item(0);
                        String coordIndexAttrib =
ifsElem.getAttribute("coordIndex");

                        //System.out.println("IndexedFaceSet coordIndex= " +
coordIndexAttrib);
                        int lastMinusOnePosition =
coordIndexAttrib.lastIndexOf("-1");
                        //System.out.println(lastMinusOnePosition);
                        if (lastMinusOnePosition != -1) {
                            String stringWithoutLastMinusOne =
coordIndexAttrib.substring(0, lastMinusOnePosition);
                            tokenizedCoord =
stringWithoutLastMinusOne.split(" -1 ");
                        } else {
                            tokenizedCoord = new String[1];
                            tokenizedCoord[0] = coordIndexAttrib;
                        }
                        //System.out.println(stringWithoutLastMinusOne);

                        /*
                         for (int i = 0; i < tokenizedCoord.length; i++) {
                         System.out.println(tokenizedCoord[i]);
                         }
                         */
                        Element coordElem = (Element)
eElement.getElementsByTagName("Coordinate").item(0);
                        String pointAttrib =
coordElem.getAttribute("point");
                        //System.out.println("Coordinate point= " +
pointAttrib);
                        tokenizedPoint = pointAttrib.split(" ");
                        totalPointParts = new ArrayList();
                        if (tokenizedPoint.length > 2) {
                            for (int i = 0; i < tokenizedPoint.length; i = i
+ 3) {
                                String pointPart =
tokenizedPoint[i].concat(" " + tokenizedPoint[i + 1] + " " +
tokenizedPoint[i + 2]);
                                //System.out.println(pointPart);
                                //System.out.println("start of next triada:
" + tokenizedPoint[i]);
                                totalPointParts.add(pointPart);
                            }
```

```
                        } else {
                            // Unique case: an IndexedFaceSet without points
(authoring error!)
                            totalPointParts.add("0 0 0");
                            totalPointParts.add("0.0000001 0 0");
                            totalPointParts.add("0 0.0000001 0");

                            tokenizedCoord=new String[1];
                            tokenizedCoord[0]="0 1 2";
                        }


                        /*
                         for (int i = 0; i < totalPointParts.size(); i++) {
                         System.out.println("No" + i + " triada: " +
totalPointParts.get(i));
                         }
                         */
                    }
                    String coordTempFile =
writeCoordIndexToFile(tokenizedCoord, file, bw);
                    String pointTempFile = writePointToFile(totalPointParts,
file, bw);
                    String[] tempFiles = {coordTempFile, pointTempFile};
                    String resultedExtraction =
ShapeIndexExtraction.shapeIndexEctraction(tempFiles);
                    resultedIFSExtractionList.add(resultedExtraction);

                } else if
(eElement.getElementsByTagName("Extrusion").item(0) != null) {
                    extrCount += 1;
                    crossSectionAttrib = null;
                    spineAttrib = null;
                    scaleAttrib = null;
                    orientAttrib = null;
                    if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                        Element extrElem = (Element)
eElement.getElementsByTagName("Extrusion").item(0);
                        crossSectionAttrib =
extrElem.getAttribute("crossSection");
                        //System.out.println(crossSectionAttrib);
                        spineAttrib = extrElem.getAttribute("spine");
                        //System.out.println(spineAttrib);
                        if (extrElem.getAttribute("scale") != null) {
                            scaleAttrib = extrElem.getAttribute("scale");
                        }
                        //System.out.println(scaleAttrib);
                        if (extrElem.getAttribute("spine") != null) {
                            orientAttrib =
extrElem.getAttribute("orientation");
                        }
                        //System.out.println(orientAttrib);
                    }
                    file = new File("Extrusion.txt");
                    String coordTempFile =
writeExtrusionParamsToFile(crossSectionAttrib, spineAttrib, scaleAttrib,
orientAttrib, file, bw);

                    String[] ExtrTempFile = {coordTempFile};
                    String resultedExtraction =
ExtrusionDescription.ExtrusionDescription(ExtrTempFile);
                    file.delete();

                    String ExtrBBox = resultedExtraction.substring(0,
resultedExtraction.indexOf("&") - 1);
                    String ExtrShape =
resultedExtraction.substring(resultedExtraction.indexOf("&") + 1);
```

```
                    //System.out.println(ExtrBBox);
                    //System.out.println(ExtrShape);
                    resultedExtrBBoxList.add(ExtrBBox);
                    resultedExtrExtractionList.add(ExtrShape);

                } else {
//                    System.out.println("oeo");
                }
            }
            //System.out.println(shapeCount + " " + ifsCount + " " +
extrCount);
            //System.out.println(resultedIFSExtractionList.size() + " " +
resultedExtrExtractionList.size() + " " + resultedExtrBBoxList.size());

            /*
             for (int i = 0; i < resultedExtractionList.size(); i++) {
             System.out.println(i + ": " + resultedExtractionList.get(i));
             }
             */
            for (int i = 0; i < resultedIFSExtractionList.size(); i++) {
                IFSStringBuilder.append(resultedIFSExtractionList.get(i));
                IFSStringBuilder.append("#");
                //System.out.println(IFSStringBuilder);
            }

            for (int i = 0; i < resultedExtrExtractionList.size(); i++) {

ExtrShapeStringBuilder.append(resultedExtrExtractionList.get(i));
                ExtrShapeStringBuilder.append("#");

                ExtrBBoxStringBuilder.append(resultedExtrBBoxList.get(i));
                ExtrBBoxStringBuilder.append("#");

                //System.out.println(IFSStringBuilder);
            }

            //System.out.println("Synoliko String: " +
IFSStringBuilder.toString());
        } catch (ParserConfigurationException e) {
        }

        //*********************************************************
        //TELOS KODIKA GIA TO EXTRACT TON POINTS ENOS INDEXEDFACESET
        //*********************************************************


        // 1. Instantiate a TransformerFactory.
        TransformerFactory factory = TransformerFactory.newInstance();
        StreamSource xslStream = new
StreamSource(Transformation.class.getResourceAsStream("x3d_to_mpeg7_transfor
m.xsl"));
        // 2. Use the TransformerFactory to process the stylesheet Source
and generate a Transformer.
        Transformer transformer = null;
        try {
            transformer = factory.newTransformer(xslStream);
        } catch (TransformerConfigurationException exXSL) {
            //System.out.println(getServletContext().getRealPath("/"));
            //System.setProperty("resultFolder",
getServletContext().getRealPath("/"));
            //System.out.println(System.getProperty("resultFolder"));
            logger.log(Priority.FATAL, null, exXSL);

//Logger.getLogger(TransformationServlet.class.getName()).log(Level.SEVERE,
null, exXSL);
        }
        transformer.setErrorListener(new MyErrorListener());
        StreamSource in = new StreamSource(inXml);
```

```
        // 3. Use the Transformer to transform an XML Source and send the
output to a Result object.
        //xmlSystemId is used to pass the full path of the xml file
currently being transformed through the xsl
        String xmlSystemId = inXml.toURI().toString();
        transformer.setParameter("filename", xmlSystemId);
        transformer.setParameter("pointsExtraction",
IFSStringBuilder.toString());
        transformer.setParameter("extrusionPointsExtraction",
ExtrShapeStringBuilder.toString());
        transformer.setParameter("extrusionBBoxParams",
ExtrBBoxStringBuilder.toString());
        String fileNameOnly =
org.apache.commons.io.FilenameUtils.removeExtension(inXml.getName());
        File mp7 = new File(inXml.getParent() + "/" + fileNameOnly +
".mp7");
        String outputXML = mp7.toURI().toString();
        try {
            transformer.transform(in, new StreamResult(outputXML));
            System.out.println("TRANSFORMED SUCCESSFULLY!");
            validate(mp7);
        } catch (TransformerException exTransform) {
            System.out.println("TRANSFORMATION ERROR! The full stack trace
of the root cause is available in the logs of Apache Tomcat.");
            logger.log(Priority.ERROR, null, exTransform);

//Logger.getLogger(TransformationServlet.class.getName()).log(Level.SEVERE,
null, exTransform);
        }

    }

    public static String writeCoordIndexToFile(String[] dataToWrite, File
tempFile, BufferedWriter writer) {
        try {
            tempFile = new File("coordIndex.txt");
            // if file doesnt exists, then create it
            if (!tempFile.exists()) {
                tempFile.createNewFile();
            }
            writer = new BufferedWriter(new FileWriter(tempFile));
            for (int i = 0; i < dataToWrite.length; i++) {
                writer.write(dataToWrite[i]);
                writer.newLine();
            }
            writer.close();
            //System.out.println("SUCCESS!!!" + tempFile.getAbsolutePath());
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return tempFile.getAbsolutePath();
    }

    public static String writePointToFile(List totalPointParts, File
tempFile, BufferedWriter writer) {
        try {
            tempFile = new File("point.txt");
            // if file doesnt exists, then create it
            if (!tempFile.exists()) {
                tempFile.createNewFile();
            }
            writer = new BufferedWriter(new FileWriter(tempFile));
            for (int i = 0; i < totalPointParts.size(); i++) {
                writer.write(totalPointParts.get(i).toString());
                writer.newLine();
            }
            writer.close();
```

```
            //System.out.println("SUCCESS!!! " +
tempFile.getAbsolutePath());
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return tempFile.getAbsolutePath();
    }

    public static String writeExtrusionParamsToFile(String
crossSectionAttrib, String spineAttrib, String scaleAttrib, String
orientAttrib, File tempFile, BufferedWriter writer) {
        try {
            // if file doesnt exists, then create it
            if (tempFile.exists()) {
                tempFile.delete();
            }
            tempFile.createNewFile();
            writer = new BufferedWriter(new FileWriter(tempFile));
            writer.write("CrossSection");
            writer.newLine();
            writer.write(crossSectionAttrib);
            writer.newLine();
            writer.write("Spine");
            writer.newLine();
            writer.write(spineAttrib);
            writer.newLine();
            if (spineAttrib != null) {
                writer.write("Scale");
                writer.newLine();
                writer.write(scaleAttrib);
                writer.newLine();
            }
            if (orientAttrib != null) {

                writer.write("Orientation");
                writer.newLine();
                writer.write(orientAttrib);
                writer.newLine();
            }
            writer.close();
            //System.out.perintln("SUCCESS!!! " +
tempFile.getAbsolutePath());
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return tempFile.getAbsolutePath();
    }

public static boolean validate(File x3dinput) throws IOException{

    //Set up the validation error listener.
    ArrayList validationErrors = new ArrayList();
    XmlOptions xmlOptions = new XmlOptions();
    xmlOptions.setErrorListener(validationErrors);
    //Bind the instance to the generated XMLBeans types. Parse the
transformed document
    //into a Mpeg7 specific document to be able to use all the Mpeg-7
capabilitities
    Mpeg7Document myDoc = null;
    try {
        myDoc = Mpeg7Document.Factory.parse(x3dinput);
    } catch (XmlException exXML) {
        System.out.println("ERROR PARSING MPEG7! The full stack trace of the
root cause is available in the logs of Apache Tomcat.");
        logger.log(Priority.FATAL, null, exXML);
```

```
//Logger.getLogger(ValidationServlet.class.getName()).log(Level.SEVERE,
null, exXML);
    }
    //customize the xmlOptions used to create//parse the xml file
    xmlOptions.setSaveNamespacesFirst();
    xmlOptions.setSavePrettyPrint();
    xmlOptions.setSavePrettyPrintIndent(3);
    xmlOptions.setSaveAggressiveNamespaces();
    final Map<String, String> implicitNamespaces = new HashMap();
    implicitNamespaces.put("xsi", "http://www.w3.org/2001/XMLSchema-
instance");
    xmlOptions.setSaveImplicitNamespaces(implicitNamespaces);
    xmlOptions.setUseDefaultNamespace();
    //The XML to be validated
    //String output = myDoc.xmlText(xmlOptions);
    myDoc.save(x3dinput);
    // During validation, errors are added to the ArrayList for
    // retrieval and printing by the printErrors method.
    boolean isValid = myDoc.validate(xmlOptions);
    // Print the errors if the XML is invalid.
    if (!isValid) {
        System.out.println("INVALID DESCRIPTION!");
        System.out.println("");
        Iterator iter = validationErrors.iterator();
        while (iter.hasNext()) {
            System.out.println(iter.next());
            System.out.println("");
        }
    }
    else
    {
        System.out.println("DESCRIPTION IS VALID!");
    }

    return isValid;
}
}

class MyErrorListener implements ErrorListener {

    private static Logger logger = Logger.getLogger(MyErrorListener.class);

    @Override
    public void warning(TransformerException e) throws TransformerException
{
        logger.log(Priority.WARN, null, e);
        //show("Warning", e);
        //throw (e);
    }

    @Override
    public void error(TransformerException e) throws TransformerException {
        logger.log(Priority.ERROR, null, e);
        //show("Error", e);
        //throw (e);
    }

    @Override
    public void fatalError(TransformerException e) throws
TransformerException {
        logger.log(Priority.FATAL, null, e);
        //show("Fatal Error", e);
        //throw (e);
    }

    private void show(String type, TransformerException e) {
        System.out.println(type + ": " + e.getMessage());
        if (e.getLocationAsString() != null) {
```

```
            System.out.println(e.getLocationAsString());

        }
    }
}
```