ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση συστήματος διάχυσης XML δεδομένων χρησιμοποιώντας αυτόματα πάνω από Δομημένα Δίκτυα Επικάλυψης, με το σύστημα Pastry

Μιχαήλ Α. Νικολάου

Επιβλέποντες: **Εμμανουήλ Κουμπαράκης**, Αναπληρωτής Καθηγητής ΕΚΠΑ
Ίρις Μηλιαράκη, Υποψήφια Διδάκτωρ ΕΚΠΑ

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2008

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση συστήματος διάχυσης XML δεδομένων χρησιμοποιώντας αυτόματα πάνω από Δομημένα Δίκτυα Επικάλυψης, με το σύστημα Pastry

Μιχαήλ Α. Νικολάου

ΑΜ: 1115200400186

ΕΠΙΒΛΕΠΟΝΤΕΣ:

Εμμανουήλ Κουμπαράκης , Αναπληρωτής Καθηγητής ΕΚΠΑ
Ίρις Μηλιαράκη , Υποψήφια Διδάκτωρ ΕΚΠΑ

Περίληψη

Παρουσιάζουμε την υλοποίηση ενός συστήματος διάχυσης XML δεδομένων, το οποίο χρησιμοποιεί μη ντετερμινιστικά πεπερασμένα αυτόματα και εκμεταλλεύεται τα προτερήματα που προσφέρονται από τα δομημένα δίκτυα επικάλυψης. Η ουσία του συστήματος είναι ότι αποτελεί μια κατανεμημένη προσέγγιση στη μεθοδολογία του συστήματος YFilter, ένα μοντέρνο σύστημα XML φιλτραρίσματος βασισμένο σε αυτόματα, το οποίο εφαρμόζουμε πάνω από το δίκτυο επικάλυψης του συστήματος Pastry, υλοποιώντας το μοντέλο δημοσίευσης / συνδρομής. Η αρχιτεκτονική του συστήματος, χρησιμοποιεί την έννοια του *κατανεμημένου* αυτόματου. Ένα κατανεμημένο αυτόματο, ορίζεται σαν ένα κανονικό αυτόματο, το οποίο κατανέμεται πάνω από ένα δίκτυο επικάλυψης, ακολουθώντας ένα σύνολο από κανόνες οι οποίοι αντιστοιχούν τις καταστάσεις του στους κόμβους του δικτύου. Έτσι, ένα αυτόματο που αντιπροσωπεύει μια επερώτηση κατανέμεται πάνω από το δίκτυο Pastry και ο κόμβος ο οποίος αιτήθηκε τον ευρετηριασμό της σημειώνεται ως *συνδρομητής* της. Κάποιος κόμβος ο οποίος δημοσιεύει ένα XML έγγραφο, θεωρείται ο *εκδότης* του. Το αυτόματο που δημιουργήθηκε πάνω από το δίκτυο κατά τον ευρετηριασμό, εκτελείται κατά τη δημοσίευση με οδηγό το XML έγγραφο, ειδοποιώντας τους συνδρομητές σε περίπτωση ικανοποίησης κάποιας επερώτησης που έχουν υποβάλει.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Κατανεμημένα Συστήματα

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ : δίκτυα ομότιμων κόμβων, κατανεμημένοι πίνακες κατακερματισμού, επεξεργασία επερωτήσεων, δημοσίευση / συνδρομή, θεωρία αυτομάτων

Abstract

We present the implementation of an XML data dissemination system which uses nondeterministic finite automata while taking advantage of the various properties of structured overlay networks. The essence of the system is a distributed approach to the YFilter methodology, a state-of-the-art automata-based XML filtering system, which is build on top of the Pastry overlay network, implementing the publish / subscribe pattern. The architecture of the system implements the notion of *distributed automata*. A distributed automaton is defined as a regular automaton which is distributed on top of an overlay network, following a set of rules that map the states of the automaton to the nodes of the network. In this way, an automaton that expresses a query is distributed on top of a Pastry network, and the corresponding peer that issued the indexing request is marked as the *query subscriber*. A *publisher* peer, stands for a peer which publishes an XML document. When publishing, the automaton that was build on top of the network when indexing, is now executed with respect to the XML document, alerting the *query subscribers* of a possible match between a query and the published document.

SUBJECT AREA: Distributed Systems

KEYWORDS: peer-to-peer networks, distributed hash tables, query processing, publish subscribe, automata theory

Αφιερωμένο στο 25 : 48

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον επιβλέποντα Καθηγητή κ. Εμμανουήλ Κουμπαράκη για την εμπιστοσύνη που μου έδειξε στην ανάθεση της εργασίας αυτής, όπως επίσης και το γενικότερα ευχάριστο κλίμα συνεργασίας που δημιούργησε. Επίσης τον ευχαριστώ για τις συμβουλές και τις συζητήσεις μας, καθώς και για τα πολύτιμα σχόλια του που έχουν βελτιώσει σε μεγάλο βαθμό την εργασία αυτή. Ευχαριστώ την Υποψήφια Διδάκτωρ Ίριδα Μηλιαράκη, με την οποία συζητούσα κάθε λεπτομέρεια του συστήματος, καθώς και τις διάφορες φάσεις της υλοποίησης. Επίσης ευχαριστώ το μεταπτυχιακό φοιτητή του ΕΚΠΑ, Χαράλαμπο Νικολάου, για τη βοήθεια που μου παρείχε με βάση τη πείρα του στη χρήση του συστήματος Pastry. Αυτή η πτυχιακή εργασία, αποτελεί τον επίλογο της περιόδου μου σαν προπτυχιακός φοιτητής, και έτσι δεν μπορώ παρά να ευχαριστήσω γονείς, συγγενείς και στενούς φίλους για την υποστήριξή τους.

Περιεχόμενα

Πρόλογος **23**

1 Εισαγωγή **25**

1.1 Στόχοι της Πτυχιακής Εργασίας 30

1.2 Διάρθρωση της Πτυχιακής Εργασίας 31

2 Θεωρητικό Υπόβαθρο και Σχετικές Ερευνητικές Προσπάθειες **33**

2.1 Δίκτυα ομότιμων κόμβων (P2P Networks) 34

2.1.1 Το Μοντέλο Πελάτη-Εξυπηρετητή και το Μοντέλο Δικτύων Ομότιμων . . . 34

2.1.2 Η Επανάσταση των Δικτύων Ομότιμων 35

2.1.3 Εφαρμογές 36

2.1.4 Ανάλυση 38

2.1.5 Υβριδικά Δίκτυα Ομότιμων και Ιεράρχηση 39

2.1.6 Αποτίμηση των Δικτύων Ομότιμων 41

2.2 Κατανεμημένοι Πίνακες Κατακερματισμού 42

2.2.1 Το Πρόβλημα Αναζήτησης στα Δίκτυα Ομότιμων 42

2.2.1.1 Μοντέλο Κεντρικού Καταλόγου (Centralized Directory Model) . 42

2.2.1.2 Μοντέλο Πλημμύρας (Flooded Requests Model) 44

2.2.1.3 Μοντέλο Δρομολόγησης Εγγράφου Document Routing Model . 44

2.2.1.4 Μετάβαση στους ΚΠΚ 45

2.2.2 Ορίζοντας τους Κατανεμημένους Πίνακες Κατακερματισμού 46
2.2.3 Chord 49
2.2.3.1 Περιγράφοντας το Chord 49
2.2.3.2 Μορφολογία και Διάταξη 50
2.2.3.3 Πίνακες Δεικτών 51
2.2.3.4 Αλλαγή στη Σύνθεση του Δικτύου: Συνδέσεις και Αποχωρήσεις . 52
2.2.4 Pastry 54
2.2.4.1 Εισαγωγικά 54
2.2.4.2 Μορφολογία του Pastry 54
2.2.4.3 Η Δομή ενός Κόμβου 55
2.2.4.4 Δρομολόγηση 56
2.2.4.5 Αποδοτικότητα Δρομολόγησης 58
2.2.4.6 Αυτό-οργάνωση και προσαρμογή 59
2.2.5 PAST 61
2.2.5.1 Εξισορρόπηση Φόρτου (Load Balancing) 62
2.3 Το μοντέλο δεδομένων XML 63
2.3.1 Σύνταξη 65
2.3.1.1 Βασικότεροι Κανόνες Σύνταξης 66
2.3.2 Ορθότητα και Εγκυρότητα 67
2.3.3 Η Γλώσσα XPath 68
2.3.4 Εμπλουτισμός XML Γεγονότων με Χαρακτηριστικά Θέσης 69
2.4 Μη ντετερμινιστικά πεπερασμένα αυτόματα 71
2.5 Συστήματα Δημοσίευσης / Συνδρομής 73
2.5.1 Το Βασικό Μοντέλο Διάδρασης 74
2.5.2 Δημοσίευση / Συνδρομή βασισμένη σε Θεματολογία 76

2.5.3 Δημοσίευση / Συνδρομή βασισμένη στο Περιεχόμενο 77
2.6 Συμπεράσματα 78

3 Περιγραφή Συστήματος 79

3.1 Το σύστημα YFilter 79
3.1.1 Συνιστώσες του Συστήματος Φιλτραρίσματος 80
3.1.2 Κατασκευή Αυτόματου από Επερωτήσεις XPath 82
3.1.3 Εκτέλεση Αυτόματου 87
3.2 Η κατανεμημένη προσέγγιση 89
3.2.1 Κατασκευή Αυτόματου 92
3.2.2 Εκτέλεση Αυτόματου 95
3.2.2.1 Επαναληπτική Προσέγγιση 97
3.2.2.2 Αναδρομική Προσέγγιση 98
3.3 Συμπεράσματα 101

4 Υλοποίηση Συστήματος χρησιμοποιώντας το Σύστημα FreePastry 103

4.1 Κατασκευή Αυτόματου 104
4.1.1 IndexQuery() 105
4.2 Εκτέλεση Αυτόματου 107
4.2.1 getTargetState() 108
4.2.2 ExpandState() 108
4.2.3 Επαναληπτική Μέθοδος 110
4.2.3.1 requestTargetStates() 110
4.2.3.2 returnTargetStates() 111
4.2.3.3 publishIterative() 112
4.2.4 Αναδρομική Μέθοδος 113

4.2.4.1 publishRecursive() 113
4.2.4.2 RecExpandState() 113
4.2.5 Παράμετρος l και Replication 116
4.3 Θέματα Υλοποίησης 118
4.4 Συμπεράσματα 119

5 Επίλογος **121**
5.1 Δοκιμές και Πειράματα 121
5.2 Πιθανές Επεκτάσεις 122

Ορολογία **125**

Συντμήσεις - Αρκτικόλεξα **131**

Βιβλιογραφία **133**

Κατάλογος Σχημάτων

1.1 Σχηματική αναπαράσταση συστήματος υλοποίησης 29

2.1 Ποσοστό κόμβων και υπερκόμβων σε μια μέτρηση στο Skype [29] 38

2.2 Παραδείγματα βαθμού αποκέντρωσης διάφορων δικτύων ομότιμων 40

2.3 Στρώματα συστημάτων που μπορούν να εκμεταλλευτούν την προσέγγιση δικτύων ομότιμων [1] 41

2.4 Βήματα στην επικοινωνία ενός υβριδικού δικτύου ομότιμων: (α) Επικοινωνία με τον εξυπηρετητή για ανάκτηση πληροφοριών, (β) Απευθείας επικοινωνία ομότιμων κόμβων [42] 42

2.5 Πώς ένας κόμβος ανακτά κάποιο αρχείο στο μοντέλο επικοινωνίας του Napster . 44

2.6 Μονοδιάστατη δρομολόγηση στο σύστημα Chord [7] 47

2.7 Παράδειγμα δακτυλίου σε ένα ιδεατό δίκτυο Chord [55] 51

2.8 Παράδειγμα δακτυλίου με πίνακα δεικτών σε ένα δίκτυο Chord [55] 52

2.9 Διαδικασία σύνδεσης του κόμβου 26 σε ένα δίκτυο Chord, χρησιμοποιώντας τον κόμβο 32 [55] 53

2.10 Η δομή ενός κόμβου στο σύστημα Pastry 56

2.11 Δρομολόγηση ενός μηνύματος σε δακτύλιο δικτύου $Pastry$ [11] 59

2.12 Η δομή του δέντρου XML για το αρχείο $dblp.xml$ (αγνοώντας τα γνωρίσματα) . . 63

2.13 Το έγγραφο $dblp.xml$ 64

2.14 Εμπλουτισμός στοιχείων XML εγγράφου με χαρακτηριστικά θέσης 70

2.15 Μη ντετερμινιστικό πεπερασμένο αυτόματο που αναγνωρίζει τη γλώσσα των δυαδικών αριθμών με ίσο αριθμό από 1 ή 0 73

2.16 Το βασικό μοντέλο δημοσίευσης / συνδρομής 74

2.17 Αποσύνδεση χωρική, χρονική και συγχρονισμού στο μοντέλο δημοσίευσης / συνδρομής 75

3.1 Η αρχιτεκτονική του συστήματος YFilter [20] 80

3.2 Παράδειγμα εξόδου του Sax parser [20] 81

3.3 Οι τέσσερις βασικές περιπτώσεις για τη μετατροπή μιας γραμμικής XPath επερώτησης σε ένα τμήμα μη ντετερμινιστικού πεπερασμένου αυτόματου [20] 82

3.4 Παραδείγματα συνένωσης τμημάτων αυτομάτων επερωτήσεων XPath [20] 85

3.5 Παραδείγματα συνένωσης αυτομάτων επερωτήσεων σε ένα τελικό αυτόματο [20] . 86

3.6 Παραδείγματα εκτέλεσης αυτόματου [20] 89

3.7 Σχηματική αναπαράσταση συστήματος υλοποίησης 90

3.8 Παραδείγματα συνένωσης επερωτήσεων σε αυτόματο [41] 92

3.9 Παραδείγματα κατανομής καταστάσεων πάνω από ένα δίκτυο Chord [41] 94

3.10 Οι επερωτήσεις του αυτόματου που παρουσιάζεται στο Σχήμα 3.11 95

3.11 Αυτόματο επερωτήσεων Σχήματος 3.2. Η κατάσταση με αριθμό i αντιστοιχεί στον υπεύθυνο για αυτή κόμβο, p_i 96

4.1 Αυτόματο επερώτησης $//a//c$ 118

4.2 Τμήμα αυτόματου επερώτησης 118

Κατάλογος Πινάκων

2.1 Τα δίκτυα ομότιμων και η εμπλοκή τους σε διάφορες υπηρεσίες 37

2.2 Τύποι Μηνυμάτων σε δίκτυα Gnutella 43

2.3 Παράδειγμα πίνακα δρομολόγησης στο δίκτυο Freenet 45

2.4 Παραμετροποίηση του Pastry 56

2.5 Ένας πίνακας δρομολόγησης με αναγνωριστικό κόμβου $65a1x$ με $b = 4$ στο Pastry [11]. Τα ψηφία είναι στο δεκαεξαδικό σύστημα. 58

2.6 Τύποι χαρακτηριστικών XML 65

2.7 Πίνακας για το αυτόματο του Σχήματος 2.15 72

3.1 Η παράμετρος l και πώς επηρεάζει τη κατανομή καταστάσεων στο αυτόματο που παρουσιάζεται στο Σχήμα 3.11 93

Κατάλογος Αλγόριθμων

2.1 Publish(filename): Δημοσίευση αρχείου με ΚΠΚ 47

2.2 Consume(filename): Κατανάλωση αρχείου με ΚΠΚ 47

2.3 Route(Msg with key D): Αλγόριθμος Δρομολόγησης στο Pastry 57

3.1 n.IndexQuery(q,d,s): Αλγόριθμος ευρετηριασμού μίας επερώτησης (Chord) . . . 97

3.2 n.PublishDocument(doc): Δημοσίευση ενός XML εγγράφου με την επαναληπτική μεθοδολογία (Chord) 99

3.3 n.ExpandState($st, event, publisherId$): Επέκταση κατάστασης ακολουθώντας τις μεταβάσεις - επαναληπτική μεθοδολογία (Chord) 99

3.4 n.RecExpandState(st,path,events): Αναδρομική επέκταση καταστάσεων σε κάθε μονοπάτι εκτέλεσης - αναδρομική μεθοδολογία (Chord) 102

4.1 IndexQuery(Query q, Int $Depth$, Subscriber Id, State st, Int l, Bool $isRecursive$): Ευρετηριασμός επερώτησης (Pastry) 106

4.2 getTargetState(State st, String label, int l): Ανάκτηση κατάστασης στόχου (Pastry) 108

4.3 ExpandState(States activeStates, String element, int l): Επέκταση κατάστασεων με βάση στοιχείο XML εγγράφου (Pastry) 109

4.4 requestTargetStates(State $activeState$, String [] $elements$): Αίτηση ανάκτησης καταστάσεων στόχων (Pastry) 110

4.5 returnTargetStates(State st, String[] $elements$): Επιστροφή καταστάσεων στόχων (Pastry) 111

4.6 publishIterative(XML doc): Επαναληπτική δημοσίευση εγγράφου (Pastry) 114

4.7 publishRecursive(XML doc): Αναδρομική δημοσίευση(Pastry) 114

4.8 RecExpandState(String $stName$, Events $docNodes$, $currentIndex$, $parentIndex$): Αναδρομική επέκταση κατάστασης (Pastry) 115

4.9 requestState$_{lmod}$(String stateName): Εκμετάλλευση l παραμέτρου σαν replication factor 117

Πρόλογος

Η πτυχιακή αυτή εκπονήθηκε κατά το ακαδημαϊκό έτος 2008-2009, υπό την επίβλεψη του Αναπληρωτή Καθηγητή του τμήματος Πληροφορικής και Τηλεπικοινωνιών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών, κ. Εμμανουήλ Κουμπαράκη και τη συνεπίβλεψη της Υποψήφιας Διδάκτωρ του ίδιου τμήματος, Ίριδας Μηλιαράκη. Ιδιαίτερο βάρος δόθηκε, εκτός από το παρόν κείμενο, στην εφαρμογή η οποία αναπτύχθηκε παράλληλα με αυτό. Η υλοποίηση του συστήματος, υπάρχει μαζί με άλλο υλικό στο αποθηκευτικό μέσο (CD) το οποίο συνοδεύει το κείμενο και συμπληρώνει τη πτυχιακή εργασία.

Κεφάλαιο 1

Εισαγωγή

Το μοντέλο δημοσίευσης / συνδρομής είναι ένα μοντέλο το οποίο διαχωρίζει τις οντότητες που λειτουργούν στα πλαίσια του σε δύο σύνολα: τους συνδρομητές (subscribers) και τους εκδότες (subscribers). Οι συνδρομητές είναι οντότητες οι οποίες εκδηλώνουν ενδιαφέρον σε κάποιο περιεχόμενο, ενώ οι εκδότες δημοσιεύουν κάποιες πληροφορίες. Αν η πληροφορία που παράγεται από κάποιο εκδότη, ικανοποιεί τα κριτήρια που έχει θέσει κάποιος συνδρομητής, τότε λαμβάνει την αντίστοιχη πληροφορία η οποία έχει εκδοθεί ή πληροφορείται για την ύπαρξή της.

Συστήματα που υλοποιούν αυτό το μοντέλο παρουσιάζονται σαν καλοί υποψήφιοι για την υποστήριξη κατανεμημένων εφαρμογών, ακόμα και σε περιβάλλοντα με δυναμικές αλλαγές, λόγω της ευελιξίας που παρουσιάζει η φύση του μοντέλου [23]. Πολλές δημοφιλείς εφαρμογές ειδοποίησης που βασίζονται στο μοντέλο δημοσίευσης / συνδρομής έχουν εμφανιστεί τα τελευταία χρόνια, εφαρμογές που περιλαμβάνουν υπηρεσίες παρακολούθησης ειδήσεων, ηλεκτρονικό εμπόριο (e-Commerce), παρατήρηση ιστοσελίδων και ιστολογίων (blogs), καθώς και διάχυση $feeds$ ιστοσελίδων (RSS).

Στο σύγχρονο Διαδίκτυο, η Επεκτάσιμη Γλώσσα Σήμανσης (Extendible Markup Language, XML) παρουσιάζεται συχνά σαν η επικρατούσα γλώσσα στην ανταλλαγή δεδομένων. Η XML, επιβάλει πολύ λίγους συντακτικούς κανόνες, ενώ η επεκτασιμότητα και ευελιξία που παρουσιάζει, παρέχουν τεράστιες δυνατότητες εκφραστικότητας στο χρήστη. Επίσης, προσφέρονται οι γλώσσες XPath και XQuery, οι οποίες είναι γλώσσες επερωτήσεων για XML δεδομένα. Λόγω της καθιέρωσης της γλώσσας στο Διαδίκτυο (Internet), ήτανε φυσικό αρκετή έρευνα στα αντίστοιχα επιστημονικά πεδία να επικεντρωθεί γύρω από τη γλώσσα και στο σχεδιασμό αποδοτικών και

κλιμακούμενων συστημάτων XML φιλτραρίσματος (XML Filtering).

Σε ένα σύστημα XML φιλτραρίσματος, ένας χρήστης υποβάλει τις ανάγκες του σε πληροφορία εκφρασμένες σε κάποια γλώσσα επερωτήσεων που αναφέρεται στη δομή αλλά και το περιεχόμενο κάποιου εγγράφου. Στα συστήματα αυτά χρησιμοποιείται ο όρος συνεχής επερώτηση (continuous query), ο οποίος αφορά επερωτήσεις οι οποίες υποβάλλονται μια φορά από το χρήστη και ενώ το περιβάλλον αλλάζει δυναμικά, η επερώτηση συνεχίζει να εκτελείται, πληροφορώντας για την ικανοποίησή της. Έτσι, μπορούμε να φανταστούμε ένα σύστημα XML φιλτραρίσματος σαν ένα σύστημα όπου Q είναι το σύνολο των συνεχών επερωτήσεων που έχουν υποβληθεί, το οποίο δέχεται στην είσοδο ένα σύνολο S εγγράφων και επιστρέφει το σύνολο εγγράφων C, όπου $C \subseteq S$, και C ικανοποιεί το σύνολο επερωτήσεων Q. Πιο συγκεκριμένα, σε ένα τέτοιο σύστημα, οι χρήστες - συνδρομητές, υποβάλλουν επερωτήσεις οι οποίες εκφράζονται σε γλώσσες XPath και XQuery, ζητώντας από το σύστημα να τους ειδοποιήσει στη περίπτωση που κάποιο εισερχόμενο XML έγγραφο γίνει αποδεκτό με βάση τις επερωτήσεις που υπέβαλαν. Οι επερωτήσεις οι οποίες υποβάλλονται από ένα χρήστη, στην ουσία συνθέτουν το προφίλ (profile) του χρήστη στο σύστημα. Στην ουσία, χρησιμοποιούμε τον όρο προφίλ για να αναφερθούμε στα δεδομένα στα οποία εκφράζει ο χρήστης κάποιο ενδιαφέρον. Αυτό, είναι και το γενικότερο πλαίσιο στο οποίο κινείται η εφαρμογή που αφορά αυτή η εργασία, με κάποιες όμως ιδιαιτερότητες που θα εξετάσουμε στη συνέχεια.

Έχουν εμφανιστεί πολλά συστήματα, τα οποία βασίζονται στο σενάριο φιλτραρίσματος που μόλις περιγράψαμε, υιοθετώντας πολλές φορές διαφορετικές προσεγγίσεις, με σκοπό το αποδοτικό φιλτράρισμα XML δεδομένων έναντι σε μεγάλα σύνολα συνεχόμενων επερωτήσεων με κεντρικοποιημένο έλεγχο [5, 19, 12, 9, 31, 27, 35, 48]. Όμως, η ανάγκη για παροχή υπηρεσιών σε κλίμακα Διαδικτύου, καθώς και τα τυπικά προβλήματα που παρουσιάζουν κεντρικοποιημένες εφαρμογές, ώθησαν τους ερευνητές σε αναζήτηση λύσεων σε *κατανεμημένα* περιβάλλοντα.

Ένα κατανεμημένο σύστημα στην ουσία είναι μια εφαρμογή η οποία διαχωρίζεται σε επιμέρους τμήματα, τα οποία τρέχουν ταυτόχρονα σε πολλαπλά υπολογιστικά συστήματα τα οποία βρίσκονται διασυνδεδεμένα σε ένα δίκτυο υπολογιστών. Ο στόχος ενός τέτοιου συστήματος είναι να συνδέει χρήστες και πόρους με ένα κλιμακούμενο και διαφανές τρόπο, να παρέχει ένα πιο εύρωστο σύστημα με αντοχές σε σφάλματα, και ιδανικά, να είναι πιο ισχυρό από κεντρικοποιημένα συστήματα. Έτσι, αναπτύχθηκαν συστήματα όπως τα [54, 14, 13, 24, 21, 26, 60], εφαρμόζοντας κατανεμημένα βασισμένα-στο-περιεχόμενο (content-based) πρωτόκολλα δρομολόγησης,

για τη διάχυση των XML δεδομένων, πάνω από ένα δίκτυο που αποτελείται από XML μεσίτες (XML brokers [1]) ή XML δρομολογητές (routers), οι οποίοι στην ουσία, βασιζόμενοι σε κάποιους κανόνες δρομολόγησης οι οποίοι υπάρχουν στους πίνακες δρομολόγησής τους, κατευθύνουν την ροή σε ένα η περισσότερους προορισμούς. Για παράδειγμα, στο σύστημα ONYX [21], ο κάθε μεσίτης, μπορεί να επικοινωνήσει με όλους τους άλλους μεσίτες στο δίκτυο, καθώς και χρησιμοποιώντας ένα πίνακα δρομολόγησης, να επικοινωνήσει μόνο με τους μεσίτες τους οποίους ενδιαφέρουν τα εισερχόμενα μηνύματα. Σε αυτό το σύστημα, οι πίνακες δρομολόγησης είναι στιγμιότυπα της μηχανής του συστήματος φιλτραρίσματος YFilter [20], το οποίο είναι ένα κεντρικοποιημένο σύστημα XML φιλτραρίσματος.

Τα προαναφερθέντα κατανεμημένα συστήματα, όμως, παρουσιάζουν κάποιες αδυναμίες. Πιο συγκεκριμένα, δεν λαμβάνουν υπ᾽ όψη το φόρτο εργασίας (processing load) που αντιστοιχεί σε κάθε XML μεσίτη, κάτι που ως αποτέλεσμα έχει την μη ισορροπημένη κατανομή φόρτου εργασίας στο δίκτυο (load imbalances). Με τον όρο *φόρτος εργασίας*, στην ουσία αναφερόμαστε σε δύο σημεία: Την επεξεργασία των εισερχόμενων XML εγγράφων με βάση τις εεπερωτήσεις, αλλά και το κόστος αποστολής ειδοποίησης στους συνδρομητές των οποίων τα κριτήρια ικανοποιούνται. Η άνιση κατανομή φόρτου μπορεί να προκαλέσει αρκετά σημαντικά προβλήματα. Καθώς ο αριθμός των επερωτήσεων αυξάνεται, η απόδοση του συστήματος μειώνεται, ενώ η υπερφόρτωση την οποία θα υποστούν ορισμένοι κόμβοι μπορεί να προκαλέσει την υπολειτουργία τους. Σημειώνουμε, ότι ορισμένα συστήματα χρησιμοποιούν κάποια κριτήρια τα οποία έχουν στόχο τη βελτίωση της κατανομής των επερωτήσεων και γενικότερα, των πόρων του συστήματος, χωρίς όμως να αντιμετωπίζουν ευθέως το πρόβλημα εξισορρόπησης φόρτου. Για παράδειγμα, το σύστημα ONYX, εμπεριέχει μια κεντρικοποιημένη συνιστώσα, η οποία χρησιμοποιεί κάποιες μετρικές, έτσι ώστε να επιλέξει σε πια στοιχεία του δικτύου να στείλει εισερχόμενες επερωτήσεις. Τέτοια κριτήρια είναι τοπολογικές αποστάσεις στο δίκτυο, αλλά και διαθέσιμο εύρος ζώνης του κόμβου. Μια άλλη αδυναμία που παρουσιάζουν τα συστήματα αυτά, είναι η εξής: Οι πληροφορίες που κρατάνε τοπικά τα στοιχεία των δικτύων στους πίνακες δρομολόγησής τους, αυξάνονται ανάλογα με τον αριθμό των επερωτήσεων, κάτι που πιθανώς να προκαλεί καθυστερήσεις στην συντήρηση των πινάκων.

Στα προβλήματα αυτά, δίνουν λύση από τη φύση τους οι Κατανεμημένοι Πίνακες Κατακερματισμού (Distributed Hash Table, DHT, ΚΠΚ). Οι ΚΠΚ, είναι μια τάξη αποκεντρωμένων κατανεμημένων συστημάτων, οι οποίοι στην ουσία μπορούν να χειριστούν την διαχείριση πληρο-

[1] `http://www.xml.com/pub/p/213`

φοριών πάνω από ένα δίκτυο, προσφέροντας λειτουργίες αναζήτησης και αποθήκευσής τους, ακριβώς όπως ένας τυπικός πίνακας κατακερματισμού. Χειρίζονται τα δεδομένα του δικτύου, έτσι ώστε δυναμικές αλλαγές στη σύνθεσή του (αποχωρήσεις - αφίξεις κόμβων) να αλλάζουν ελάχιστα την αντιστοιχία πληροφοριών σε κόμβους, κάτι που καθιστά τα εν λόγω συστήματα εύρωστα και ανθεκτικά στις συνεχόμενες αλλαγές που πιθανόν να συμβαίνουν σε κάποιο δίκτυο. Οι ΚΠΚ προσφέρουν μέγεθος πινάκων δρομολόγησης σταθερό ως προς το μέγεθος της πληροφορίας που διακινούν στο δίκτυο, αλλά ανάλογο προς τον αριθμό των κόμβων που συμμετέχουν σε αυτό. Επίσης, προσφέρουν μεθοδολογίες εξισορρόπησης φόρτου, αντιμετωπίζοντας έτσι τις αδυναμίες που έχουν επισημανθεί στις υπόλοιπες κατανεμημένες μεθοδολογίες. Η μεθοδολογία αυτή έχει χρησιμοποιηθεί ήδη με πολλή επιτυχία σε διάφορα συστήματα [8, 30, 59]. Επίσης, η χρήση συστημάτων ΚΠΚ επιτρέπει τη λειτουργία του συστήματος σε μη αξιόπιστα και ετερογενής μηχανήματα του διαδικτύου, χαρακτηριστικό που είναι βοηθητικό στην υιοθέτηση του συστήματος στην παροχή διαδικτυακών υπηρεσιών.

Το σύστημα που αφορά η υλοποίηση της πτυχιακής εργασίας [41], είναι ένα σύστημα διάχυσης XML δεδομένων, το οποίο εκμεταλλεύεται το δίκτυο επικάλυψης[2] που προσφέρουν τα συστήματα ΚΠΚ, έτσι ώστε να παρουσιάσει ένα κατανεμημένο περιβάλλον, το οποίο αντιμετωπίζει όσο το δυνατόν καλύτερα τα προβλήματα που αντιμετώπιζαν άλλα κατανεμημένα συστήματα στο τομέα του XML φιλτραρίσματος.

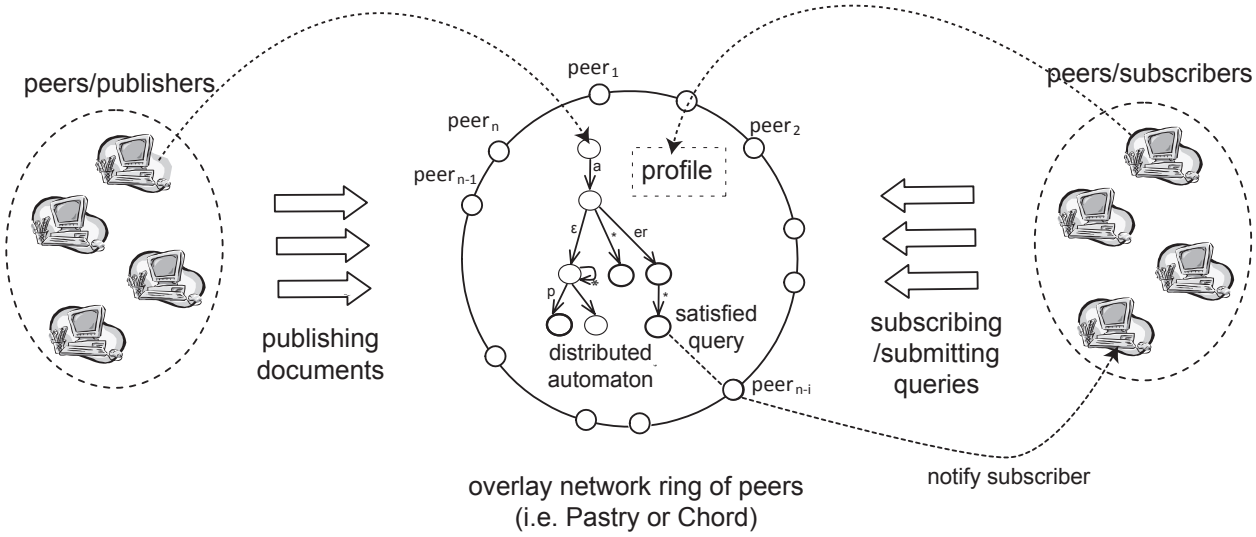
Πιο συγκεκριμένα, η μεθοδολογία που εφαρμόζεται επεκτείνει το επιτυχημένο κεντρικοποιημένο σύστημα YFilter [20], το οποίο χρησιμοποιεί *μη υτετερμινιστικά πεπερασμένα αυτόματα* (Non Deterministic Finite Auomata, NFA) για να αναπαραστήσουν το σύνολο των επερωτήσεων που έχουμε σε ένα σύστημα. Στο YFilter, κάθε έγγραφο XML το οποίο δίνεται σαν είσοδος στο σύστημα, χρησιμοποιείται σαν *οδηγός* για την εκτέλεση του αυτόματου. Η άφιξη σε μία κατάσταση τερματισμού του αυτόματου, σηματοδοτεί την ικανοποίηση μιας ή περισσοτέρων επερωτήσεων (ανεξάρτητα με το αν εξαντλήθηκε η είσοδος, κάτι που συμβαίνει στα παραδοσιακά αυτόματα).

Στη κατανεμημένη προσέγγιση, παρουσιάζεται η κατασκευή, εκτέλεση αλλά και συντήρηση (π.χ. επαύξηση) ενός *κατανεμημένου* αυτόματου (distributed automaton). Ένα *κατανεμημένο αυτόματο*, ορίζεται στο [41] σαν η κατανομή των καταστάσεων που αποτελούν το αυτόματο στους κόμβους που αποτελούν το δίκτυο, χρησιμοποιώντας κάποιους δεδομένους κανόνες. Σε

[2]Με τον όρο δίκτυο επικάλυψης, αναφερόμαστε σε ένα δίκτυο το οποίο λειτουργεί πάνω από ένα άλλο δίκτυο, όπως για παράδειγμα ένα σύστημα ΚΠΚ πάνω από κόμβους του διαδικτύου. Περισσότερες λεπτομέρειες θα αναφερθούν στο κύριο σώμα της εργασίας.

κάθε κόμβο, αντιστοιχούν *τμήματα* (fragments) του αυτόματου, με το μέγεθος τους να αποτελεί παράμετρο του συστήματος, η οποία στην ουσία μπορεί να ρυθμίσει τη κίνηση στο δίκτυο καθώς και το φόρτο που επιβάλλεται στον κάθε κόμβο.

Η μεθοδολογία αυτή, εκτός από τα προαναφερθέντα πλεονεκτήματα, αφαιρεί και την ανάγκη για μια κεντρικοποιημένη συνιστώσα (component), για την ανάθεση επερωτήσεων στους κόμβους του συστήματος. Συγκεκριμένα, όπως θα γίνει καθαρό στη παρουσίαση του συστήματος, η φύση των ΚΠΚ επιτρέπει την άμεση κατανομή των επερωτήσεων στο δίκτυο. Στο [41], παρουσιάζονται δυο μεθοδολογίες εκτέλεσης του *κατανεμημένου* αυτόματου. Μία επαναληπτική και μια αναδρομική. Ενώ στην επαναληπτική, ο κόμβος ο οποίος εκδίδει κάποιο XML έγγραφο είναι υπεύθυνος για τη διαχείριση της εκτέλεσης του αυτόματου, η αναδρομική μέθοδος εκμεταλλεύεται τον έμφυτο παραλληλισμό που παρουσιάζει ένα NFA, καθώς και την κατανεμημένη φύση του δικτύου, έτσι ώστε να εκτελεί ταυτόχρονα περισσότερα του ενός μονοπάτια στο κατανεμημένο αυτόματο που έχει δημιουργηθεί πάνω από το δίκτυο.



Σχήμα 1.1: Σχηματική αναπαράσταση συστήματος υλοποίησης

Στο Σχήμα 3.7, μπορούμε να δούμε μια σχηματική αναπαράσταση του συστήματος που θα μελετηθεί και υλοποιηθεί. Στο κέντρο, παρουσιάζεται το δίκτυο ομότιμων κόμβων το οποίο αποτελεί το σύστημα. Στα δεξιά, βλέπουμε κάποιους από τους κόμβους ομότιμων οι οποίοι αποστέλλουν επερωτήσεις στο σύστημα, οι οποίες ευρετηριάζονται σε ένα *κατανεμημένο* αυτό-

ματο, το οποίο παρουσιάζεται στο κέντρο του δικτύου. Στα αριστερά, κάποιοι κόμβοι δημοσιεύουν έγγραφα στο σύστημα. Κατά τη δημοσίευση, εκτελείται το κατανεμημένο αυτόματο και στη περίπτωση που κάποια επερώτηση ικανοποιηθεί - δηλαδή συναντηθεί κάποια τελική κατάσταση, ο αντίστοιχος συνδρομητής - κόμβος ειδοποιείται.

1.1 Στόχοι της Πτυχιακής Εργασίας

Στη συνέχεια, θα απαριθμήσουμε τους στόχους που θέτει η πτυχιακή εργασία, η οποία αποτελείται από το παρών κείμενο, καθώς και τη συνοδευτική εφαρμογή η οποία έχει αναπτυχθεί.

- Να παρουσιάσει με σημαντική λεπτομέρεια, το θεωρητικό υπόβαθρο και η σχετική βιβλιογραφία που σχετίζεται με το σύστημα το οποίο θα αναπτυχθεί. Αυτό, σημαίνει την σε βάθος επισκόπηση των αντίστοιχων θεματικών ενοτήτων, όπως για παράδειγμα δίκτυα ομότιμων (peer-to-peer), και ΚΠΚ.

- Την παρουσίαση του συστήματος το οποίο θα υλοποιηθεί, κάτι που περιλαμβάνει την παρουσίαση του κεντρικοποιημένου συστήματος YFilter καθώς και τη παρουσίαση του κατανεμημένου συστήματος.

- Την υλοποίηση του συστήματος, χρησιμοποιώντας την γλώσσα προγραμματισμού Java, και το σύστημα DHT Pastry[52]. Συγκεκριμένα, θα χρησιμοποιηθεί η ανοικτού κώδικα υλοποίηση FreePastry, η οποία παρέχει βιβλιοθήκες υλοποιημένες σε Java. Επίσης, για τον έλεγχο ορθότητας θα χρησιμοποιηθεί κατάλληλη βιβλιοθήκη ώστε το σύστημα να είναι σε θέση να παράγει κάποια οπτικοποίηση του αυτόματου (βλέπε Κεφάλαιο 4). Στόχος της υλοποίησης είναι να δοκιμαστεί σε κάποιο πραγματικό σύστημα (testbed), και να είναι κατάλληλη για τη διενέργεια πειραμάτων και τη λήψη μετρήσεων σε κάποιο τέτοιο σύστημα. Επίσης, σκοπός της υλοποίησης είναι να είναι όσο το δυνατόν πιο ευέλικτη, παρέχοντας ελευθερία στις παραμέτρους.

- Τέλος, σκοπός της εργασίας είναι να παρουσιάσει την ίδια την υλοποίηση σε ένα κεφάλαιο, δίδοντας σε ψευδοκώδικα τις βασικές λειτουργίες της υλοποίησης. Επίσης, σκοπός είναι να παρουσιαστούν λεπτομέρειες και τυχόν δυσκολίες που εμφανίστηκαν στην υλοποίηση. Σημειώνεται, ότι η υλοποίηση / ψευδοκώδικάς της, παρουσιάζουν την επέκταση των αλγορίθμων που παρουσιάζονται σε ψευδοκώδικα στο αρχικό σύστημα, έτσι ώστε να εκμεταλλεύονται περιπτώσεις όπου αποθηκεύουμε περισσότερες από μία καταστάσεις του

αυτόματου σε ένα κόμβο, με μεθοδολογίες που έχουν προκύψει μετά από συζητήσεις με τους επιβλέποντες της πτυχιακής εργασίας και συγγραφείς του [41].

1.2 Διάρθρωση της Πτυχιακής Εργασίας

Η διάρθρωση της εργασίας αυτής έχει ως εξής:

- Στο Κεφάλαιο 2, παρουσιάζεται η σχετική βιβλιογραφία και το θεωρητικό υπόβαθρο του συστήματος. Εξετάζουμε δηλαδή τη θεωρία πίσω από τα συστήματα, καθώς και εργαλεία και εφαρμογές τα οποία είναι σχετικά με αυτό.
- Στο Κεφάλαιο 3, παρουσιάζουμε το YFilter [20], καθώς και το κατανεμημένο σύστημα που είναι το επίκεντρο της εργασίας [41].
- Στο Κεφάλαιο 4, αναλύουμε την υλοποίηση του συστήματος, συμβαδίζοντας με τους στόχους 1.1 της πτυχιακής.
- Στο Κεφάλαιο 5, κλείνουμε την εργασία αυτή με μια σύνοψη της παρουσίασης, καθώς και με κάποιες πιθανές επεκτάσεις και μελλοντικές προσπάθειες που μπορούν να προκύψουν.

Στο Παράρτημα, παρουσιάζουμε την ορολογία που χρησιμοποιούμε στην εργασία καθώς και τις διάφορες συντμήσεις.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο και Σχετικές Ερευνητικές Προσπάθειες

Στην ενότητα αυτή, θα παρουσιάσουμε σε βάθος τη γνώση και τις τεχνολογίες, πάνω στις οποίες βασίζεται το σύστημα που αποτελεί το στόχο της υλοποίησης. Θα κάνουμε μια μεγάλη αναφορά στα δίκτυα ομότιμων και στους κατανεμημένους πίνακες κατακερματισμού, αναφερόμενοι κυρίως στα συστήματα Chord και Pastry, με μεγαλύτερο βάρος στο τελευταίο, το οποίο αποτελεί και το σύστημα υλοποίησης της πτυχιακής. Επίσης, θα αναφερθούμε στη γλώσσα XML, στη θεωρία που αφορά τα μη ντετερμινιστικά πεπερασμένα αυτόματα, καθώς και στα μοντέλα δημοσίευσης / συνδρομής. Πιο συγκεκριμένα, η διάρθρωση του κεφαλαίου αυτού έχει ως εξής:

Στην Ενότητα 2.1, παρουσιάζουμε τα δίκτυα ομότιμων, με μια εκτεταμένη επισκόπηση στην ιστορία τους, τις εφαρμογές τους, καθώς και τη θεωρία που τα στηρίζει, ενώ παρουσιάζουμε και μια σύντομη αποτίμησή τους.

Στην Ενότητα 2.2, εισάγουμε την έννοια των κατανεμημένων πινάκων κατακερματισμού. Αναφέρουμε τους λόγους που συντέλεσαν στη δημιουργία τους, ορίζουμε έννοιες που τους αφορούν, καθώς και αναλύουμε σε βάθος τα συστήματα Chord και Pastry, με το Pastry να είναι το σύστημα υλοποίησης.

Στην Ενότητα 2.3, αναφερόμαστε στην επεκτάσιμη γλώσσα σήμανσης, ή αλλιώς γλώσσα XML. Εισάγουμε τα χαρακτηριστικά και το συντακτικό της, ενώ κάνουμε αναφορά στη γλώσσα επερωτήσεων XPath και στον εμπλουτισμό των XML εγγράφων με χαρακτηριστικά θέσης. Στο

σύστημα υλοποίησης, η XML είναι η γλώσσα των εγγράφων που δημοσιεύονται (τα οποία εμπλουτίζονται με χαρακτηριστικά θέσης κατά την αναδρομική εκτέλεση του συστήματος), ενώ η XPath είναι η γλώσσα επερωτήσεων που χρησιμοποιούμε.

Στην Ενότητα 2.4, αναφερόμαστε στο τομέα της Θεωρίας Υπολογισμού (Theory of Computation, TOC), και συγκεκριμένα στη Θεωρία Αυτομάτων επικεντρώνοντας στο τμήμα που αφορά τα μη ντετερμινιστικά πεπερασμένα αυτόματα. Τα αυτόματα, είναι η δομή στην οποία μετατρέπονται οι επερωτήσεις στο σύστημα υλοποίησης.

Τέλος, στην Ενότητα 2.5, εξετάζουμε τα συστήματα δημοσίευσης / συνδρομής, παρουσιάζοντας κάποια σημαντικά χαρακτηριστικά και μεθόδους λειτουργίας τους.

2.1 Δίκτυα ομότιμων κόμβων (P2P Networks)

Τα δίκτυα ομότιμων κόμβων, προφανώς, ικανοποιούν τον ορισμό των δικτύων επικοινωνιών, όπως αυτός δίνεται από τον Tanenbaum: *'Ενα δίκτυο υπολογιστών (computer network), ορίζεται σαν ένα σύνολο από αυτόνομους υπολογιστές, οι οποίοι είναι διασυνδεδεμένοι με μια συγκεκριμένη τεχνολογία* [57], μπορούν δηλαδή, να ανταλλάξουν πληροφορίες μεταξύ τους. Για να δώσουμε έμφαση στο μοντέλο που ακολουθούν τα δίκτυα ομότιμων, είναι καλό να το αντιπαραβάλουμε με ένα από τα επικρατέστερα μοντέλα δικτύωσης που καθιερώθηκαν μέχρι σήμερα, να τονίσουμε δηλαδή τι **δεν** είναι τα δίκτυα ομότιμων. Το μοντέλο αυτό, είναι το μοντέλο πελάτη-εξυπηρετητή.

2.1.1 Το Μοντέλο Πελάτη-Εξυπηρετητή και το Μοντέλο Δικτύων Ομότιμων

Ένα από τα κεντρικά μοντέλα των δικτύων υπολογιστών είναι το μοντέλο πελάτη-εξυπηρετητή (client-server model). Είναι χαρακτηριστικό, ότι οι περισσότερες σημερινές εφαρμογές εφαρμόζουν αυτό το μοντέλο, καθώς και τα κύρια πρωτόκολλα εφαρμογών[1] του διαδικτύου. Συγκεκριμένα, μια εφαρμογή που τρέχει στο δίκτυο, ξεχωρίζει τα συστήματα που βρίσκονται συνδεδεμένα σε αυτό σε πελάτες και εξυπηρετητές. Ο πελάτης, διατυπώνει μια αίτηση, την οποία αναλαμβάνει να ικανοποιήσει ο εξυπηρετητής. Ο εξυπηρετητής επιλέγει εάν θα αποδεχτεί το μήνυμα, το επεξεργάζεται και επιστρέφει την απάντηση στο πελάτη. Καθιερώνεται έτσι, μια αυστηρή ιεραρχία στα στοιχεία που αποτελούν το δίκτυο υπολογιστών.

[1] Κάποια παραδείγματα: τα πρωτόκολλα HTTP, SMTP, Telnet, DNS .

Τα δίκτυα ομότιμων κόμβων, ξεφεύγουν από τη νοοτροπία του μοντέλου πελάτη-εξυπηρετητή. Η αρχιτεκτονική τους, δεν κατατάσσει κόμβους[2] σε πελάτες ή εξυπηρετητές, αλλά θεωρεί ότι οποιοσδήποτε κόμβος του δικτύου μπορεί να παίξει και τον ένα και τον άλλο ρόλο. Έτσι, αφού οι κόμβοι είναι "ίσοι" ο ένας απέναντι στον άλλο, τα δίκτυα ονομάστηκαν δίκτυα ομότιμων. Αξίζει να σημειωθεί, ότι οι αρχικοί κόμβοι[3] του ARPANET, του δικτύου που αργότερα μετεξελίχθηκε στο διαδίκτυο, σχεδιάστηκαν να λειτουργούν σαν ομότιμοι κόμβοι [44]. Η ιδιότητα αυτή, προσφέρει τη δυνατότητα έτσι ώστε κάθε κόμβος να μπορεί να προσφέρει χώρο, ταχύτητα και άλλους πόρους στο δίκτυο. Λόγω της κατανεμημένης φύσης των δικτύων ομότιμων, καλό είναι σε αυτό το σημείο να ορίσουμε επακριβώς τι σημαίνει κατανεμημένο δίκτυο. Σύμφωνα με τον Tanenbaum, ενα κατανεμημένο σύστημα ορίζεται σαν *ένα σύνολο αυτόνομων υπολογιστών το οποίο παρουσιάζεται στους χρήστες του σαν ένα σύστημα* [58]. Αυτή είναι και η βασική διαφορά ενός δικτύου υπολογιστών και ενός κατανεμημένου συστήματος, *ότι σε ένα δίκτυο υπολογιστών αυτή η ενιαία εικόνα απουσιάζει* [57].

Ένας άλλος ορισμός που θα μας χρησιμεύσει, είναι ο ορισμός των δικτύων επικάλυψης (overlay network), γιατί στην ουσία αναφερόμαστε σε δίκτυα επικάλυψης ομότιμων. Ένα δίκτυο επικάλυψης, είναι στην ουσία ένα τεχνητό δίκτυο με λογικές ζεύξεις, το οποίο δομείται πάνω από ένα υπάρχων δίκτυο, με σκοπό να προσφέρει μια δικτυακή υπηρεσία η οποία δεν προσφέρεται στο υπάρχων δίκτυο. Έτσι, μπορούμε να θεωρήσουμε τη σύνδεση στο διαδίκτυο μέσω παραδοσιακής γραμμής (Dial-up) σαν ένα δίκτυο επικάλυψης, πάνω από το τηλεφωνικό δίκτυο. Έτσι, τα περισσότερα δίκτυα ομότιμων είναι δίκτυα επικάλυψης, γιατί προσφέρουν τις υπηρεσίες τους πάνω από ένα άλλο δίκτυο, το Διαδίκτυο.

2.1.2 Η Επανάσταση των Δικτύων Ομότιμων

Το πρώτο ευρείας χρήσης δίκτυο ομότιμων, ήταν το δίκτυο Usenet (1979), το οποίο λειτουργεί ακόμα. Το Usenet, είναι ένα κατανεμημένο δίκτυο στο οποίο οι χρήστες μπορούν να δημοσιεύσουν και να διαβάσουν άρθρα, τα οποία κατατάσσονται σε κάποια κατηγορία (newsgroup). Η επανάσταση των δικτύων ομότιμων κόμβων όμως, άρχισε το 1999 με την έκδοση του πρωτοκόλλου του δικτύου Freenet. Από τότε, πολλά γνωστά συστήματα, όπως το KaZaA, το KAD network, το Napster, το Gnutella αλλά και το Bit Torrent καθιερώθηκαν σαν δίκτυα διαμοιρασμού δεδομένων (File Sharing Systems), αναστατώνοντας τη μουσική βιομηχανία αρχικά, και στη συνέ-

[2]Με τον όρο κόμβος, εννοούμε ένα στοιχείο του δικτύου όπως ένα υπολογιστή, ή ένα δρομολογητή (router).
[3]Οι κόμβοι αυτοί ήταν το UCLA, το SRI, το UCSB, και το πανεπιστήμιο της Utah.

χεια τη βιομηχανία παραγωγής ταινιών. Οι χρήστες των δικτύων ομότιμων μπορούσαν πλέον να διαμοιραστούν αρχεία μουσικής αλλά και ταινίες χωρίς λογοκρισία και χωρίς να έχουν πάντα τα πνευματικά δικαιώματα διακίνησης των αρχείων. Από τότε, ακολούθησαν χιλιάδες μηνύσεις κατά συστημάτων αλλά και χρηστών, με πιο γνωστές τις μηνύσεις[4] κατά του Napster το οποίο αναγκάστηκε να διακόψει τις υπηρεσίες του το 2001.

Γιατί όμως τα δίκτυα ομότιμων κόμβων έχουν τόση μεγάλη απήχηση; Στο [7], γίνεται μια απόπειρα προσέγγισης των πλεονεκτημάτων που προσφέρουν:

- Τα διάφορα φράγματα, όπως οικονομικά και ανθρωπίνου δυναμικού, είναι πολύ χαμηλά, αφού συνήθως δεν απαιτούν ειδικές διοικητικές ή οικονομικές επενδύσεις, αντίθετα με τα αντίστοιχα κεντρικοποιημένα συστήματα.

- Τα δίκτυα ομότιμων κόμβων προσφέρουν ένα τρόπο συγκέντρωσης και χρήσης μιας τρομακτικά μεγάλης (και αυξανόμενης κάθε μέρα) υπολογιστικής και αποθηκευτικής ποσότητας σε υπολογιστές που συμμετέχουν στο Διαδίκτυο.

- Η αποκεντρωμένη και κατανεμημένη φύση των δικτύων ομότιμων τους δίνει τη προοπτική και την ευρωστία (robust) να αντέχουν σφάλματα, αποτυχίες και κακόβουλες επιθέσεις, ιδιότητα που τα κάνει ιδανικά για μακροχρόνια αποθήκευση δεδομένων, καθώς και για χρονοβόρους υπολογισμούς.

2.1.3 Εφαρμογές

Παρόλο που στο ευρύ κοινό τα δίκτυα ομότιμων σημαίνουν "διαμοιρασμός αρχείων", στη πραγματικότητα βρίσκουν πολλές άλλες εφαρμογές. Για παράδειγμα, το εγχείρημα SETI@home, το οποίο αποτελεί πρωτοβουλία του πανεπιστημίου UCLA. Η λέξη SETI είναι ακρώνυμο για τη φράση "Search for Extra-Terrestial Intelligence", δηλαδή *Αναζήτηση για Εξωγήινη Νοημοσύνη*. Στόχος του όμως δεν ήταν μόνο να αναζητήσει νοημοσύνη εκτός γης - όπως μπορεί κάποιος να συμπεράνει από το όνομα του εγχειρήματος, αλλά και να αποδείξει τη βιωσιμότητα και πρακτική σημασία ενός " Κατανεμημένου Υπολογιστικού Πλέγματος" (Distributed Grid Computing). Το SETI, εκμεταλλευόταν τον αδρανή χρόνο (idle time) του επεξεργαστή κάποιου υπολογιστή ο οποίος ήταν συνδεδεμένος στο δίκτυο, έτσι ώστε να εκτελέσει πολύπλοκους και χρονοβόρους υπολογισμούς, όπως επεξεργασία σήματος. Σύμφωνα με την ιστοσελίδα του εγχειρήματος, περισσότεροι από 3 εκατομμύρια υπολογιστές έχουν χρησιμοποιηθεί από το 2001, επιτυγχάνοντας

[4] A&M Records v. Napster, 239 F.3d 1004 (9th Cir. 2001)

συνολικές ταχύτητες που έφταναν τις $23.37 * 10^{12}$ πράξεις κινητής υποδιαστολής το δευτερόλεπτο (sustained terra-flops). Στην ουσία, αυτός είναι και ένας ορισμός του Υπολογιστικού Πλέγματος: "Χρήση πόρων από πολλά υπολογιστικά συστήματα τα οποία βρίσκονται συνδεδεμένα σε ένα δίκτυο με σκοπό τη λύση μεγάλων ή πολύ μεγάλων υπολογιστικών προβλημάτων" [22]. Ένα άλλο διάσημο εγχείρημα που χρησιμοποιεί το υπολογιστικό πλέγμα, είναι το Folding@Home, του πανεπιστημίου Stanford. Αφορά την επεξεργασία πρωτεϊνών και τη κατανόηση λειτουργιών τους και είναι το πρώτο υπολογιστικό σύστημα, το οποίο έχει καταφέρει να ξεπεράσει τις ταχύτητες των 10^{15} πράξεων κινητής υποδιαστολής το δευτερόλεπτο (petaFLOPS). Αξιοσημείωτη είναι η χρήση Υπολογιστικού Πλέγματος από τον Ευρωπαϊκό Οργανισμό για Πυρηνική Έρευνα (CERN) στα διάφορα πειράματά του.

Πίνακας 2.1: Τα δίκτυα ομότιμων και η εμπλοκή τους σε διάφορες υπηρεσίες

	$p2p$ διάδραση χρήστη	$p2p$ εφαρμογές	$p2p$ διαχείριση πληροφοριών
eBay	ΝΑΙ	ΟΧΙ	ΟΧΙ
Napster	ΝΑΙ	ΝΑΙ	ΟΧΙ
Gnutella	ΝΑΙ	ΝΑΙ	ΝΑΙ
Freenet	ΝΑΙ	ΝΑΙ	ΝΑΙ

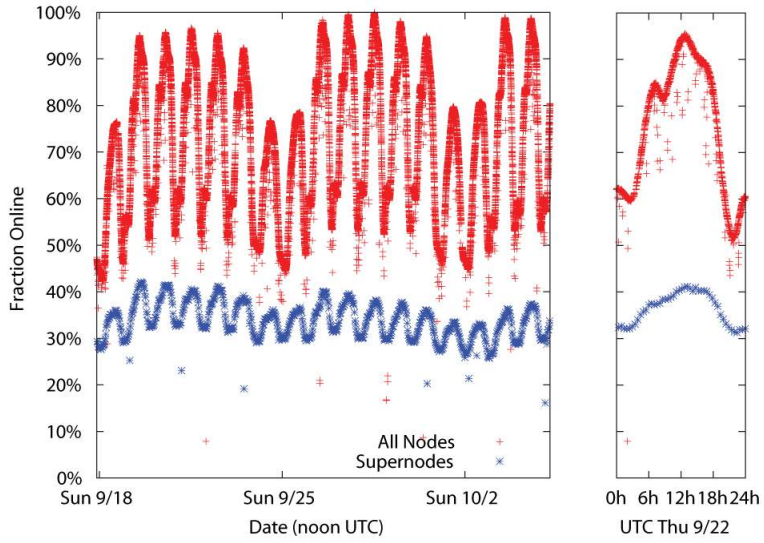
Μια αξιοσημείωτη εφαρμογή των δικτύων ομότιμων η οποία άλλαξε τις ανθρώπινες επικοινωνίες, είναι οι εφαρμογές που επιτρέπουν την επικοινωνία μέσω γραπτών μηνυμάτων (chat) αλλά και την τηλεφωνική συνδιάλεξη χρηστών μέσω του διαδικτύου. Συγκεκριμένα, το Skype, μια τέτοια εφαρμογή, μετράει 309.9[5] εκατομμύρια χρήστες το 2008. Το Skype, χρησιμοποιεί την τεχνολογία VoIP (Voice Over IP), δηλαδή τηλεφωνία μέσω διαδικτύου, η οποία επιτρέπει συνομιλία μέσω υπολογιστή σε πραγματικού χρόνο, πάνω από το διαδίκτυο. Προσφέρει υπηρεσίες συνδιάλεξης, ανταλλαγής αρχείων και ανταλλαγής μηνυμάτων κειμένου. Το Skype οργανώνει τους κόμβους[6] του με βάση το μοντέλο υπερκόμβων (supernodes) [29]. Περιληπτικά, στο μοντέλο αυτό, με βάση κάποια κριτήρια κάποιοι συνηθισμένοι κόμβοι επιλέγονται να λειτουργήσουν σαν υπερκόμβοι. Οι υπερκόμβοι, οι οποίοι μπορούν να λειτουργήσουν και σαν κανονικοί κόμβοι, οργανώνουν ένα δίκτυο μεταξύ τους, μέσω του οποίου οι κανονικοί κόμβοι μπορούν να εκδώσουν μια αίτηση. Όπως θα δούμε στη συνέχεια, τα δίκτυα ομότιμων στα οποία παρατηρούμε

[5] `www.skype.com`

[6] Σε πολλά άρθρα, οι κόμβοι δικτύων ομότιμων λέγονται *servents*, μια λέξη η οποία είναι ετεροσυνθετική, και προέρχεται από τις αγγλικές λέξεις *client* και *server*, που σημαίνουν αντίστοιχα πελάτης και εξυπηρετητής.

μια ιεραρχία στους κόμβους, λέγονται *Ιεραρχικά Δίκτυα Ομότιμων* (hierarchical peer-to-peer networks). Μπορούμε να δούμε το ποσοστό κόμβων και υπερκόμβων στο Skype σε μια μέτρηση [29], στο Σχήμα 2.1.

Άλλες χρήσης δικτύων ομότιμων, αφορούν εφαρμογές βιοπληροφορικής, προγράμματα διαμοιρασμού επιστημονικών βάσεων δεδομένων, στρατιωτικές εφαρμογές, τηλεπικοινωνίες καθώς ακόμη και εφαρμογές πολυμέσων. Η εμπλοκή τους σε διάφορες δημοφιλής υπηρεσίες παρουσιάζεται στο Πίνακα 2.1, ενώ στο Σχήμα 2.3 μπορούμε να δούμε ότι η προσέγγιση των δικτύων ομότιμων μπορεί να εφαρμοστεί σε διάφορα επίπεδα κάποιου συστήματος.



Σχήμα 2.1: Ποσοστό κόμβων και υπερκόμβων σε μια μέτρηση στο Skype [29]

2.1.4 Ανάλυση

Θα συνεχίσουμε, ορίζοντας τυπικά τα δίκτυα ομότιμων. Ένας διαισθητικός ορισμός δίνεται από τον Clay Shirkey [44]:

"Τα δίκτυα ομότιμων αποτελούν μια κλάση εφαρμογών, οι οποίες εκμεταλλεύονται τους πόρους - αποθηκευτικό χώρο, υπολογιστική δύναμη, περιεχόμενο, ανθρώπινη παρουσία - οι οποίοι είναι διαθέσιμη στο διαδίκτυο. Επειδή η πρόσβαση σε αυτές τις αποκεντρωμένες (κατανεμημένες)

πηγές σημαίνει τη λειτουργία σε ένα περιβάλλον ασταθούς συνδεσιμότητα και απρόβλεπτων IP διευθύνσεων (Internet Protocol Address), τα δίκτυα ομοτίμων επιβάλλεται να λειτουργούν εκτός του *Συστήματος Ονομάτων Τομέα* (Domain Name System, DNS) και να έχουν σημαντική ή ολοκληρωτική αυτονομία από κεντρικούς εξυπηρετητές".

Σύμφωνα με τους Aberer και Hauswirth [1], ένα δίκτυο ομότιμων θα πρέπει να πληρεί και τα εξής κριτήρια:

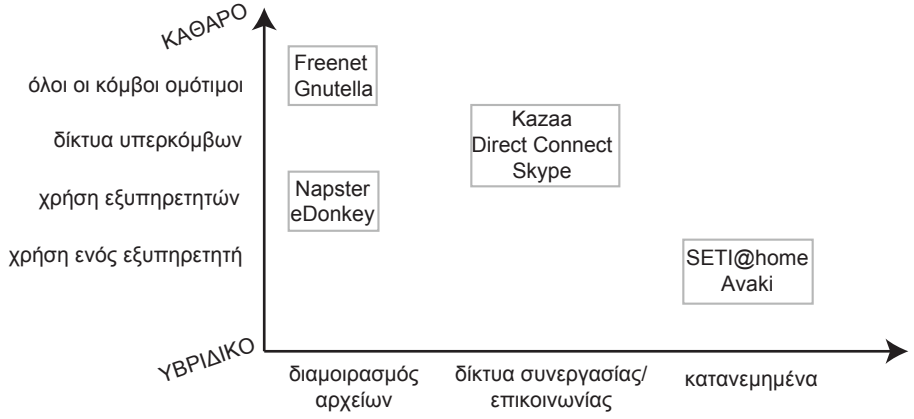
- Η αρχή του διαμοιρασμού πόρων: Όλα τα δίκτυα ομότιμων περιλαμβάνουν με κάποιο τρόπο την έννοια του διαμοιρασμού. Οι πόροι που διαμοιράζονται μπορεί να είναι φυσικοί, όπως αποθηκευτικός χώρος σε ένα σκληρό δίσκο ή εύρος ζώνης κάποιου δικτύου, όπως επίσης και νοητοί ή λογικοί πόροι, όπως υπηρεσίες ή διάφορες μορφές γνώσης. Με βάση αυτή την αρχή, μπορούν να υλοποιηθούν εφαρμογές οι οποίες θα ήταν αδύνατο να υλοποιηθούν πάνω από ένα κόμβο. Η ιδέα αυτή αποτέλεσε τη κινητήρια δύναμη πίσω από τη δημιουργία του Napster.

- Η αρχή της αποκέντρωσης: Η αρχή αυτή αποτελεί άμεση συνέπεια της αρχής διαμοιρασμού πόρων. Μέρη του συστήματος, ή και ολόκληρο το σύστημα δεν είναι πλέον κάτω από κεντρικό έλεγχο. Έτσι, αποφεύγονται πιθανές περιοχές συμφόρησης στο δίκτυο (bottlenecks) καθώς και ενισχύεται η ανθεκτικότητά (robustness) του συστήματος (αφού ο έλεγχος δεν εξαρτάται από ένα και μοναδικό κόμβο). Κάποια πλήρως αποκεντρωμένα συστήματα, αποτελούν το Gnutella και το Freenet.

- Η αρχή της αυτό οργάνωσης: Προφανώς, όταν ενα σύστημα είναι πλήρως αποκεντρωμένο, δεν υπάρχει κεντρικός έλεγχος. Αυτό, σημαίνει ότι κάθε κόμβος θα πρέπει να οργανώνει και να συντονίζει τη συμπεριφορά του, βασιζόμενος στις εσωτερικές πληροφορίες που κατέχει, καθώς και στις πληροφορίες που μπορεί να ανακτήσει από μια γειτονία κόμβων. Η συμπεριφορά, λοιπόν, του συστήματος, είναι στην ουσία η αθροιστική συμπεριφορά των κόμβων που το αποτελούν.

2.1.5 Υβριδικά Δίκτυα Ομότιμων και Ιεράρχηση

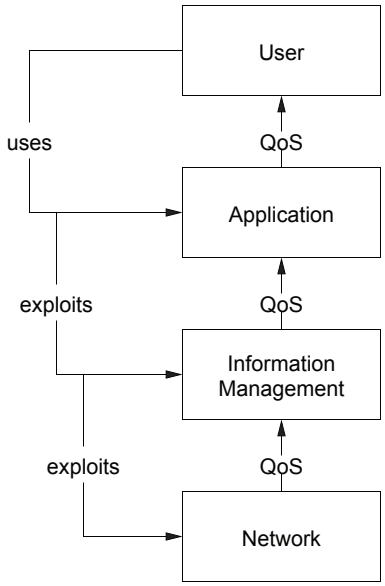
Σε αυτό το σημείο, θα ήταν καλό να παρατηρήσουμε ότι τα δίκτυα ομότιμων διαχωρίζονται σε καθαρά (pure) και υβριδικά (hybrid) [42]. Στα καθαρά δίκτυα ομότιμων, δεν υπάρχει κανένας κεντρικός έλεγχος. Το μοντέλο ανταποκρίνεται στους ορισμούς, θεωρώντας κάθε κόμβο ένα



Σχήμα 2.2: Παραδείγματα βαθμού αποκέντρωσης διάφορων δικτύων ομότιμων

εντελές ανεξάρτητο μέρος του δικτύου. Παραδείγματα αυτού του τύπου αποτελούν τα δίκτυα Gnutella kai Freenet. Το υβριδικό μοντέλο διαφοροποιείται στο εξής: Ένας κεντρικός κόμβος προσεγγίζεται αρχικά έτσι ώστε να αντληθούν μετα-πληροφορίες (meta-information). Αυτές οι πληροφορίες μπορεί να αφορούν θέματα ασφάλειας του δικτύου, αλλά και πληροφορίες όπως σε ποιο κόμβο βρίσκεται αποθηκευμένη η τάδε πληροφορία. Από εκείνο το σημείο και μετά, το δίκτυο λειτουργεί σαν ένα καθαρό δίκτυο ομότιμων. Τη διαδικασία αυτή μπορούμε να τη δούμε στο Σχήμα 2.4.

Ο κεντρικός αυτός έλεγχος, μπορεί να πάρει πολλές μορφές. Για παράδειγμα, έχουμε ήδη αναφέρει τη περίπτωση των υπερκόμβων (Skype). Κάποιοι κόμβοι, με βάση κάποια κριτήρια που αφορούν το εκάστοτε δίκτυο, επιλέγονται σαν υπερκόμβοι. Μέσω αυτών, οι απλοί κόμβοι μπορούν να ανακτήσουν σημαντικές πληροφορίες και να συνεχίσουν την επικοινωνία τους στο δίκτυο. Μπορεί επίσης, να υπάρχουν ένας ή περισσότεροι κόμβοι που λειτουργούν σαν εξυπηρετητές, έτσι ώστε οι υπόλοιποι κόμβοι να ανακτούν πληροφορίες για τη περαιτέρω επικοινωνία τους. Για παράδειγμα στο Napster, ο κόμβος επικοινωνεί με τον εξυπηρετητή, ανακτά πληροφορίες για το κόμβο στον οποίο είναι αποθηκευμένη η πληροφορία που ψάχνει και στη συνέχεια, επικοινωνεί απευθείας με τον κόμβο αυτό. Οι υβριδικές προσεγγίσεις (βλέπε και Σχήμα 2.2), εισάγουν μια ιεράρχηση στο σύστημα (γιαυτό καλούνται και ιεραρχικά δίκτυα ομότιμων) και με αυτό το τρόπο αυξάνουν τη κλιμάκωση (scalability) του συστήματος, επιτρέπουν δηλαδή στο σύστημα να εξυπηρετήσει πολύ μεγάλο φόρτο πληροφορίας και κόμβων.

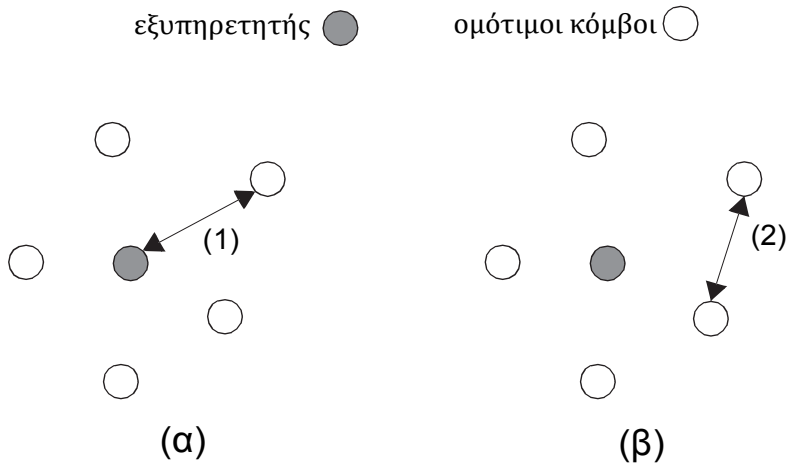


Σχήμα 2.3: Στρώματα συστημάτων που μπορούν να εκμεταλλευτούν την προσέγγιση δικτύων ομότιμων [1]

2.1.6 Αποτίμηση των Δικτύων Ομότιμων

Σίγουρα, η μέχρι τώρα θεωρητική αλλά και πρακτική περιγραφή των δικτύων ομότιμων, περνάει ένα αισιόδοξο και γεμάτο αυτοπεποίθηση κλίμα στον αναγνώστη. Το ενθουσιώδες κλίμα αυτό, καλλιεργείται από την απήχηση των εφαρμογών στον επιστημονικό και όχι μόνο κόσμο, το εύρος των επιτυχημένων ερευνητικών έργων και το μέλλον που προδιαγράφει ο τομέας. Η πραγματική πρόκληση όμως, βρίσκεται στο σχεδιασμό των πρωτοκόλλων αυτών των δικτύων και στην σχεδίαση και την υλοποίηση των συστημάτων που τα αποτελούν. Ο σχεδιαστής, δεν καλείται να σκεφτεί μόνο μία *έξυπνη ιδέα*. Στην ουσία, καλείται να δημιουργήσει ένα *αξιόπιστο* σύστημα, χρησιμοποιώντας πολλές φορές *αναξιόπιστους* υπολογιστές. Αυτό, γιατί συνήθως τα δίκτυα ομότιμων, στηρίζονται πάνω σε φθηνούς, αναξιόπιστους, ανεξάρτητους και πολλές φορές αφιερωμένους σε άλλους σκοπούς υπολογιστές, οι οποίοι τις περισσότερες φορές ανήκουν σε διαφορετικές οργανώσεις. Ως τρανό παράδειγμα, μπορούμε να πάρουμε τις επιστημονικές προσπάθειες που βασίζονται στη τεχνολογία του Υπολογιστικού Πλέγματος. Στις προσπάθειες αυτές, συμμετέχουν οι υπολογιστές χρηστών οι οποίοι μπορεί να αφιερώνουν το 99% του αποθηκευτικού χώρου του υπολογιστή τους, της επεξεργαστικής δύναμής του καθώς και του διαθέσιμου εύρους ζώνης της δικτυακής τους σύνδεσης σε άλλους σκοπούς. Το πρωτόκολλο του δικτύου ομότιμων, θα πρέπει να εκμεταλλεύεται στο έπακρο το 1% που αναμένει, κτίζοντας έτσι ένα εύρωστο και

κλιμακούμενο υπέρ-υπολογιστικό σύστημα, με ίσως εκατοντάδες χιλιάδες τέτοιους χρήστες.



Σχήμα 2.4: Βήματα στην επικοινωνία ενός υβριδικού δικτύου ομότιμων: (α) Επικοινωνία με τον εξυπηρετητή για ανάκτηση πληροφοριών, (β) Απευθείας επικοινωνία ομότιμων κόμβων [42]

2.2 Κατανεμημένοι Πίνακες Κατακερματισμού

2.2.1 Το Πρόβλημα Αναζήτησης στα Δίκτυα Ομότιμων

Τα πρώτα δίκτυα ομοτίμων τα οποία εκμεταλλεύονταν πόρους όπως αποθηκευτικός χώρος και εύρος ζώνης μέσω του διαδικτύου, υιοθετούσαν διάφορες μεθοδολογίες, έτσι ώστε να αντιμετωπίσουν ένα βασικό πρόβλημα: Πώς ο κόμβος Α ο οποίος έψαχνε τη πληροφορία Β, θα ανακάλυπτε ποιος κόμβος (έστω Χ) μπορούσε να τον προμηθεύσει με αυτή τη πληροφορία. Είναι φανερό, οτι στην ουσία το πρόβλημα αυτό αποτελεί μια δικτυακή προσαρμογή ενός απο τα γνωστότερα προβλήματα στο χώρο της Πληροφορικής, το πρόβλημα αναζήτησης (lookup). Στη συνέχεια, θα δούμε περιληπτικά πώς κάθε ένα από τα πρώτα δίκτυα ομότιμων αντιμετώπιζε το πρόβλημα αυτό.

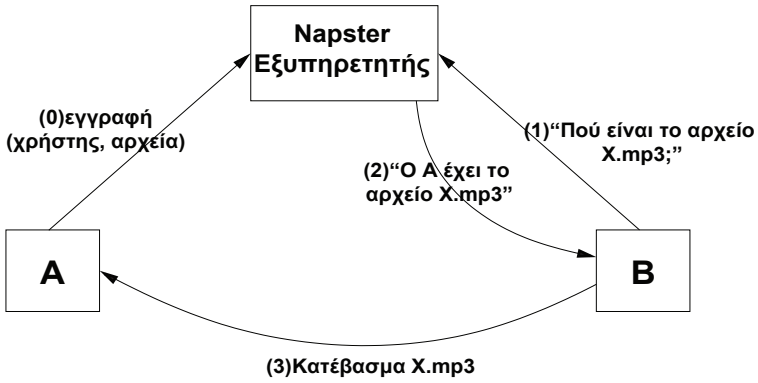
2.2.1.1 Μοντέλο Κεντρικού Καταλόγου (Centralized Directory Model)

Αυτό το μοντέλο υιοθετήθηκε από το Napster. Το Napster διατηρούσε ένα κεντρικό εξυπηρετητή ο οποίος αποθήκευε μια λίστα αρχείων (ευρετήριο) και τους κόμβους στους οποίους βρισκότανε αυτά τα αρχεία. Έτσι, ο κάθε κόμβος μπορούσε να εκδώσει επερωτήσεις, και μαζί με τα αποτελέσματα να λάβει και τους κόμβους που θα πρέπει να επικοινωνήσει για να λάβει το

Πίνακας 2.2: Τύποι Μηνυμάτων σε δίκτυα Gnutella

Τύπος	Περιγραφή	Περιλαμβανόμενες Πληροφορίες
$Ping$	Ανακοίνωση διαθεσιμότητας και αναζήτηση άλλων κόμβων	Καμία
$Pong$	Απάντηση σε ένα $Ping$	IP διεύθυνση, αριθμός θύρας ($Port$) του αντίστοιχου κόμβου· αριθμός και μέγεθος διαμοιραζόμενων αρχείων κόμβου
$Query$	Αίτηση για αναζήτηση	Ελάχιστο εύρος ζώνης κόμβου που απαντάει· Κριτήρια αναζήτησης
$QueryHit$	Επιστρέφεται από κόμβους που έχουν αναζητήσει το αρχείο αριθμός αποτελεσμάτων	IP διεύθυνση, θύρα και εύρος ζώνης του κόμβου που απαντάει σύνολο αποτελεσμάτων
$Push$	Αιτήσεις για κατέβασμα αρχείων για κόμβους που βρίσκονται πίσω από αναχώματα ασφαλείας ($Firewalls$)	αναγνωριστικό κόμβου· αριθμός αρχείου· IP και θύρα στην οποία θα αποσταλεί το αρχείο

αρχείο. Η αναζήτηση στο μοντέλο αυτό, ορίζεται σαν μια μέθοδος *δομημένης αναζήτησης* (structured lookup), δηλαδή αναζήτηση κατά την οποία ο κάθε κόμβος διατηρεί ενα καλά ορισμένο σύνολο πληροφοριών σχετικά με τους άλλους κόμβους του δικτύου [7]. Η ύπαρξη του κεντρικού ελέγχου σε αυτή τη μεθοδολογία, έκανε το σύστημα ευάλωτο σε επιθέσεις αλλά και λιγότερο εύρωστο και ανθεκτικό, παρουσίαζε δηλαδή προβλήματα ευρωστίας (robustness)και κλιμάκωσης (scalability). Η βάση του συστήματος μπορεί να έφτανε τεράστια μεγέθη, όπως επίσης και αποτελούσε ένα μοναδικό σημείο αποτυχίας (single point of failure), δηλαδή εάν για κάποιο λόγο σταματούσε να λειτουργεί, τότε σταματούσε να λειτουργεί ολόκληρο το δίκτυο. Στο Σχήμα 2.5, παρουσιάζεται η διαδικασία αναζήτησης και ανάκτησης ενός αρχείου στο δίκτυο. Αρχικά, ο χρήστης A εγγράφεται γνωστοποιώντας τα αρχεία του. Στο βήμα 1 κάποιος άλλος χρήστης (B) αναζητεί ένα αρχείο εκδίδοντας μια αίτηση στον εξυπηρετητή. Στο βήμα 3 ο εξυπηρετητής γνωστοποιεί το κάτοχο του αρχείου και τελικά, στο βήμα 4 ο B το κατεβάζει από τον A.



Σχήμα 2.5: Πώς ένας κόμβος ανακτά κάποιο αρχείο στο μοντέλο επικοινωνίας του Napster

2.2.1.2 Μοντέλο Πλημμύρας (Flooded Requests Model)

Το μοντέλο αυτό, το οποίο εφαρμόστηκε αρχικά από το δίκτυο Gnutella[7], απέρριπτε τη νοοτροπία κεντρικού έλεγχου αλλά χρησιμοποιούσε μηνύματα πλημμύρας (flooding), μηνύματα δηλαδή τα οποία στέλνονταν σε όλους τους κόμβους και ζητούσαν κάποια πληροφορία. Πιο συγκεκριμένα, το πρωτόκολλο του Gnutella ήταν ένα απλό πρωτόκολλο για αναζήτηση σε κατανεμημένο δίκτυο. Χρησιμοποιούσε πέντε βασικούς τύπους μηνυμάτων, με βασικό χαρακτηριστικό το πεδίο "χρόνου ζωής" (Time To Live, TTL). Κάθε κόμβος που λάμβανε μια επερώτηση, μείωνε κατά ένα το χρόνο αυτό (εφ᾽ όσον ήταν μεγαλύτερος του μηδενός) και με τη προϋπόθεση ότι δεν είχε ξανασυναντήσει τον κόμβο - αποστολέα του μηνύματος, προωθούσε το μήνυμα στη γειτονία κόμβων του. Η τελευταία προϋπόθεση αφορούσε την αποφυγή βρόγχων. Προφανώς, σε κάθε λήψη μηνύματος επερώτησης, ο κόμβος ελέγχει τη τοπική λίστα αρχείων του για να αποφασίσει εάν έχει κάποιο αρχείο το οποίο ικανοποιεί την επερώτηση. Η μεθοδολογία αυτή παρείχε ένα πιο ανθεκτικό δίκτυο, αλλά ήτανε σίγουρα πολύ λιγότερο αποδοτική. Οι τύποι μηνυμάτων που υποστηρίζονται από το Gnutella παρουσιάζονται στο Πίνακα 2.2.

2.2.1.3 Μοντέλο Δρομολόγησης Εγγράφου Document Routing Model

Το μοντέλο αυτό, εισήχθηκε από το δίκτυο Freenet[8]. Διαισθητικά, η βασική ιδέα του Freenet έχει ως εξής: Αντιστοιχούσε ένα κλειδί σε κάθε αρχείο, και η δρομολόγηση στο δίκτυο γινότανε με βάση το κλειδί αυτό. Αρχεία με παρόμοια κλειδιά ομαδοποιούνταν σε μια συστάδα (cluster) κόμβων. Έτσι, με κάποια ευρετικά, οι κόμβοι στους οποίους γινότανε αναζήτηση κάποιου

[7]Το δίκτυο Gnutella δημιουργήθηκε αρχικά από την Nullsoft, εταιρία που δημιούργησε το παγκόσμια γνωστό πρόγραμμα αναπαραγωγής αρχείων mp3 Winamp, με (αρχικό) σκοπό την ανταλλαγή συνταγών μαγειρικής

[8]`http://freenetproject.org/`

Πίνακας 2.3: Παράδειγμα πίνακα δρομολόγησης στο δίκτυο Freenet

Κλειδί	Δεδομένα	Διεύθυνση
8e47683isdd0932uje89	ZT38hwe01h02hdhgdzu	tcp/125.45.12.56:6474
456r5wero04d903iksd0	Rhweui12340jhd091230	tcp/67.12.4.65:4711
f3682jkjdn9ndaqmmxia	eqwe1089341ih0zuhge3	tcp/127.156.78.20:8811
wen09hjfdh03uhn4218	erwq038382hjh3728ee7	tcp/78.6.6.7:2544
712345jb89b8nbopledh		tcp/40.56.123.234:1111
d0ui43203803ujoejqhh		tcp/128.121.89.12:9991

αρχείου μειώνονταν δραστικά. Για να το επιτύχει αυτό, διατηρούσε πίνακες δρομολόγησης σε κάθε κόμβο (routing tables). Όπως βλέπουμε και στο Πίνακα 2.3 ένας πίνακας δρομολόγησης στο Freenet περιέχει κλειδιά που αντιστοιχούν σε δεδομένα καθώς και διευθύνσεις γειτονικών κόμβων, έτσι ώστε να μπορεί να προωθεί αιτήσεις σε αυτούς. Επίσης, το Freenet εισήγαγε την έννοια των αντιγράφων (replicas). Αρχεία τα οποία είχαν μεγάλη ζήτηση, μεταφέρονταν σε κόμβους στους οποίους είχαν περισσότερες πιθανότητες να βρεθούν, σύμφωνα πάντα με τους αλγόριθμους αναζήτησης του Freenet. Λόγω των πινάκων που διατηρεί, η μέθοδος λέγεται οτι χρησιμοποιεί *αδόμητους πίνακες δρομολόγησης, (unstructured routing tables)*.

2.2.1.4 Μετάβαση στους ΚΠΚ

Οι δύο προηγούμενες μέθοδοι αναζήτησης, εισάγουν την έννοια των συμμετρικών αλγορίθμων αναζήτησης (symmetric lookup algorithms). Σε αυτή τη περίπτωση, όσον αφορά την αναζήτηση, κανένας κόμβος δεν είναι πιο σημαντικός από τον άλλο, ενώ ο κάθε κόμβος τυπικά συμμετέχει μόνο σε ένα μικρό αριθμό μονοπατιών στο σύστημα.

Η παρουσία των προηγουμένων συστημάτων και τα προβλήματα που το κάθε ένα παρουσίαζε, οδήγησαν στην περαιτέρω έρευνα με σκοπό τη δημιουργία ενός συστήματος που θα συνδύαζε τα πλεονεκτήματα του κάθε ενός από αυτά. Η κατεύθυνση που έδωσε το Freenet και οι υπόλοιπες απόπειρες, θα αναπτυσσόταν περισσότερο και θα κατέληγε σε αυτό που καλούμε Κατανεμημένους Πίνακες Κατακερματισμού (Distributed Hash Tables, DHT, ΚΠΚ). Η φιλοσοφία αυτή, υλοποιήθηκε από συστήματα όπως τα CAN [50], Chord [55], Kademlia [40], Pastry [52], Tapestry [33], Viceroy [38], P-Grid [2] και Bamboo [51].

Το BitTorrent[α΄] είναι ίσως το πιο συχνά χρησιμοποιούμενο πρωτόκολλο για διαμοιρασμό αρχείων μέσω διαδικτύου στις μέρες μας. Η πρώτη υλοποίησή του, κυκλοφόρησε τον Ιούλη του 2001 από τον Bram Cohen. Είναι μια μέθοδος κατανομής τεράστιων ποσών δεδομένων πάνω από το διαδίκτυο, απαλλάσσοντας τον διακινητή από το κόστος σε εύρος ζώνης και υλικό χρησιμοποιώντας την εξής νοοτροπία: Τα δεδομένα που ο διακινητής θέλει να μοιραστεί, χωρίζονται σε κομμάτια (chunks) δεδομένων. Κάθε ένας από τους παραλήπτες ενός τέτοιου κομματιού, μόλις παραλάβει το κομμάτι, γίνεται πλέον διακινητής του, και αναλαμβάνει την αποστολή σε νέους παραλήπτες. Έτσι, ο αρχικός διακινητής, χρειάζεται να στείλει μόνο μία φορά κάθε κομμάτι, όμως το κομμάτι αυτό καταλήγει σε όλους τους παραλήπτες.

Για τη λειτουργία του πρωτοκόλλου, δημιουργείται ένα μικρό αρχείο (.torrent) το οποίο περιέχει πληροφορίες για τα αρχεία τα οποία διακινούνται, καθώς και τον ιστοχώρο στον οποίο έχει δημοσιευτεί. Συνήθως, ο εξυπηρετητής ο οποίος "βοηθάει" τους ομότιμους κόμβους να συνδεθούν ο ένας με τον άλλο, καλείται tracker. Μέσω αυτού, μπορεί ένας κόμβος να εντοπίσει τους άλλους κόμβους στο διαδίκτυο οι οποίοι συμμετέχουν στο εκάστοτε torrent. Εναλλακτικά, μια πιο αποκεντρωμένη προσέγγιση ακολουθείται με την υιοθέτηση ΚΠΚ, όπου κάθε κόμβος λειτουργεί και σαν tracker.

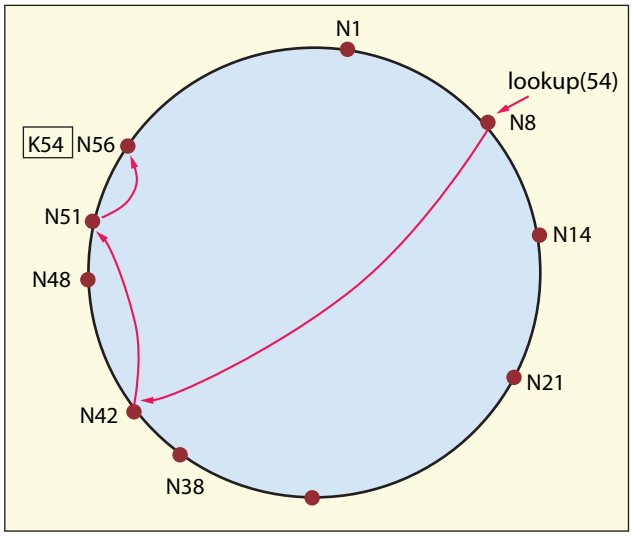
Εκτός από τον tracker και τους ΚΠΚ, μια άλλη μεθολογία που χρησιμοποιεί το πρωτόκολλο για την ανακάλυψη κόμβων, αφορά την "ανταλλαγή ομότιμων κόμβων" (peer exchange). Με αυτή τη μεθοδολογία, κάθε κόμβος ανταλλάσσει μηνύματα με τους υπόλοιπους και μαθαίνει για άλλους κόμβους τους οποίους γνωρίζουν ένας ή περισσότεροι άλλοι κόμβοι, αλλά όχι ο ίδιος.

[α΄] http://www.bittorrent.com/

2.2.2 Ορίζοντας τους Κατανεμημένους Πίνακες Κατακερματισμού

Προφανώς, ένας πίνακας κατακερματισμού φαίνεται να είναι μια ελκυστική λύση στο πρόβλημα αναζήτησης που παρουσιάστηκε στα παρθενικά δίκτυα ομότιμων. Μία καταχώρηση σε ένα πίνακα κατακερματισμού αποτελείται από ένα ζευγάρι κλειδιού και τιμής (key-value pair). Τα κλειδιά μπορεί να είναι οποιαδήποτε μοναδικά αριθμητικά πεδία, ενώ η τιμή μπορεί να είναι ένα αρχείο δεδομένων - ή ένας δείκτης σε κάποιο χώρο μνήμης που περιέχει τη πληροφορία.

Θεωρητικά, ένας ΚΠΚ είναι απαραίτητο να εφαρμόζει μια και μόνο συνάρτηση: Την $lookup(Key)$ [7], η οποία επιστρέφει την τοποθεσία στο εκάστοτε δίκτυο, του κόμβου ο οποίος είναι υπεύθυνος



Σχήμα 2.6: Μονοδιάστατη δρομολόγηση στο σύστημα Chord [7]

για το κλειδί (Key). Μόνο με αυτή τη συνάρτηση, μπορεί πολύ απλά να υλοποιηθεί ένα σύστημα διαμοιρασμού αρχείων πάνω από ένα ΚΠΚ. Θα περιγράψουμε μια πιθανή υλοποίηση στη συνέχεια, χρησιμοποιώντας μια συνάρτηση κατακερματισμού όπως η SHA1 [46, 37][9].

Αλγόριθμος 2.1 Publish(filename): Δημοσίευση αρχείου με ΚΠΚ

```
1: var hash ← SHA1(filename)
2: var node = lookup(hash)
3: node.store(filename)
```

Αλγόριθμος 2.2 Consume(filename): Κατανάλωση αρχείου με ΚΠΚ

```
1: var hash ← SHA1(filename)
2: var node = lookup(hash)
3: node.retrieve(filename)
```

Ο αλγόριθμος $publish$, δέχεται σαν όρισμα το όνομα του αρχείου που θα δημοσιεύσει ο κόμβος ($publisher$). Χρησιμοποιώντας μια συνάρτηση κατακερματισμού, ανακτά το κλειδί του ονόματος αρχείου, και χρησιμοποιώντας την συνάρτηση $lookup$ του ΚΠΚ, βρίσκει το σωστό κόμβο για τη

[9]Η συνάρτηση SHA-1είναι μια πολύ κοινή συνάρτηση κατακερματισμού, η οποία παράγει κλειδιά μήκους 160 bit

δημοσίευσή του. Αποστέλλει το αρχείο στο κόμβο αυτό. Ο καταναλωτής κόμβος, θα εκτελέσει τη δεύτερη συνάρτηση. Η μόνη διαφορά είναι ότι θα ανακτήσει το αρχείο από το κόμβο στον οποίο βρίσκεται.

Στη συνέχεια, θα δούμε κάποια άλλα θέματα στα οποία θα πρέπει να μπορεί ένας ΚΠΚ να ανταποκρίνεται [7]:

- Αντιστοίχηση κλειδιών σε κόμβους με ίση κατανομή φόρτου (load balancing): Στο δίκτυο, κάθε κλειδί και κόμβος διακρίνονται από ένα *αναγνωριστικό* (identifier, ID)που είναι ένας αριθμός που αποτελείται από $m-bits$. Κάθε κλειδί αποθηκεύεται σε έναν ή περισσότερους κόμβους, οι οποίοι βρίσκονται πιο κοντά στο κλειδί του στο χώρο αναγνωριστικών (ID space). Ο όρος "κοντά", είναι σχετικός με την υλοποίηση του συστήματος. Περισσότερα για τη συνάρτηση απόστασης θα δούμε στη συνέχεια.

- Προώθηση μιας αίτησης $lookup(Key)$ στο κατάλληλο κόμβο: Κάθε κόμβος Q ο οποίος λαμβάνει την αίτηση, θα πρέπει να την προωθεί σε κάποιο κόμβο U του οποίου η απόσταση είναι πιο κοντά στο κλειδί από τον Q. Έτσι, ο ΚΠΚ μπορεί να εγγυηθεί ότι τελικά η αίτηση θα καταλήξει στο σωστό κόμβο.

- Συνάρτηση απόστασης ($distance\ function$): Σε αυτό το σημείο θα ορίσουμε τη συνάρτηση απόστασης. Η απόσταση, είναι μια έννοια στην οποία αναφερθήκαμε στα προηγούμενα σημεία και η σημασία που αποδίδεται σε αυτή αφορά το εκάστοτε σύστημα. Για παράδειγμα, το σύστημα Chord, ορίζει τη συνάρτηση απόστασης ως την αριθμητική διαφορά μεταξύ δύο αναγνωριστικών. Στα συστήματα Pastry και Tapestry, η συνάρτηση απόστασης επιστρέφει τον αριθμό των κοινών προθεματικών bit μεταξύ δύο αναγνωριστικών. Στο δίκτυο Kademlia, η απόσταση ορίζεται σαν η XOR των bit των δύο αναγνωριστικών (bitwise xor). Σύμφωνα πάντα και με το προηγούμενο σημείο, σε κάθε βήμα της διαδικασίας $lookup$, η απόσταση μειώνεται.

- Προσαρμοστικό κτίσιμο πινάκων δρομολόγησης: Για να συντηρηθεί η διαδικασία της αναζήτησης, θα πρέπει ο κάθε κόμβος να γνωρίζει κάποιες πληροφορίες για τη γειτονία του, έτσι ώστε να μπορεί να προωθήσει την αναζήτηση. Αυτό, σημαίνει ότι θα πρέπει να διατηρεί κάποιους πίνακες δρομολόγησης, οι οποίοι θα ανανεώνονται συνεχώς και θα προσαρμόζονται στη δυναμική μορφή που μπορεί να έχει ένα δίκτυο ομότιμων, με κόμβους να αποχωρούν και να εισέρχονται στο δίκτυο. Η δρομολόγηση μπορεί να είναι μονοδιάστατη (Σχήμα 2.6) είτε πολυδιάστατη.

Στη συνέχεια, θα αναφερθούμε σε κάποια γνωστά συστήματα που προσφέρουν λειτουργικότητα δικτύου ομότιμων κόμβων και υλοποιούν κάποια διεπαφή Κατανεμημένου Πίνακα Κατακερματισμού. Τα βασικά συστήματα στα οποία θα αναφερθούμε είναι το Chord [55] και το Pastry [52]. Ιδιαίτερη αναφορά θα γίνει στο τελευταίο, αφού είναι και το σύστημα πάνω στο οποίο έχει υλοποιηθεί η εφαρμογή της πτυχιακής.

2.2.3 Chord

2.2.3.1 Περιγράφοντας το Chord

Το σύστημα Chord[55], σύμφωνα και με τη θεωρία, υλοποιεί τη βασική λειτουργία των ΚΠΚ: Την $lookup(Key)$. Μέσω αυτής της συνάρτησης, αντιστοιχεί ένα οποιοδήποτε κλειδί σε κάποιο κόμβο του δικτύου. Το μοντέλο του συστήματος κατανέμει τα κλειδιά σε ίση κατανομή μεταξύ των κόμβων (load balance), χρησιμοποιώντας μια παραλλαγή του συνεπούς κατακερματισμού (consistent hashing), μια μέθοδο κατακερματισμού όπου τα κλειδιά δεν αλλάζουν σημαντικά καθώς αυξομειώνεται το μέγεθος του πίνακα (στη συγκεκριμένη περίπτωση, καθώς αυξομειώνεται ο αριθμός των κόμβων).

Η συνάρτηση κατακερματισμού του συστήματος, αναθέτει ένα αναγνωριστικό *m bit* σε κάθε κόμβο και κλειδί που ανήκει στο σύστημα. Το m επιλέγεται να είναι αρκετά μεγάλο, έτσι ώστε να μην υπάρχουν περισσότερα του ενός στοιχεία του δικτύου με το ίδιο αναγνωριστικό. Η συνάρτηση κατακερματισμού SHA1 που όπως έχει προαναφερθεί δίνει 160 bit, είναι κατάλληλη για μια τέτοια χρήση. Σημειώνεται, ότι το αναγνωριστικό για τα στοιχεία του δικτύου (π.χ. αρχεία) δίνεται από το αποτέλεσμα της συνάρτησης κατακερματισμού για το κλειδί που αντιστοιχεί σε αυτά (π.χ. το όνομα του αρχείου), ενώ για τους κόμβους το αναγνωριστικό αφορά το αποτέλεσμα της συνάρτησης κατακερματισμού για την διαδικτυακή διεύθυνση (IP address) του κόμβου.

Σε ένα σύστημα Chord με N-κόμβους, κάθε κόμβος διατηρεί πληροφορίες για $O(NlogN)$ άλλους κόμβους, και επιλύει τα υπόλοιπα $lookup$ με $O(NlogN)$ μηνύματα. Έτσι, το σύστημα αντιμετωπίζει το πρόβλημα της κλιμάκωσης (scalability), αφού το κόστος αυξάνεται μόνο λογαριθμικά. Επίσης, το σύστημα προσαρμόζεται στις αλλαγές στη τοπολογία του (κόμβοι που συνδέονται / αποσυνδέονται από το δίκτυο), καθώς και προσφέρει ευελιξία στα κλειδιά, αφού δεν επιβάλλει κανένα περιορισμό σε αυτά.

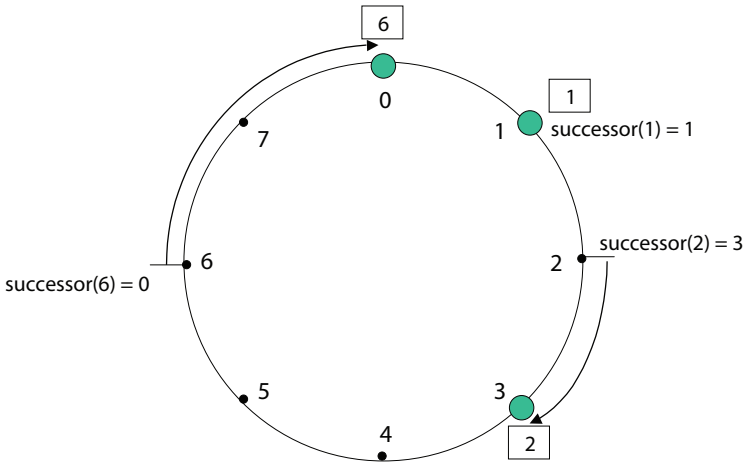
2.2.3.2 Μορφολογία και Διάταξη

Στη συνέχεια, θα περιγράψουμε το βασικό πρωτόκολλο του συστήματος, το οποίο καθορίζει τη διαδικασία εντοπισμού κάποιου κλειδιού, καθώς καλείται τη διαδικασία προσαρμογής στη περίπτωση σύνδεσής ή αποχώρησης κάποιου κόμβου.

Το Chord, ομαδοποιεί τα αναγνωριστικά σε ένα ιδεατό δακτύλιο (ring) αναγνωριστικών. Ο δακτύλιος αυτός, διατάσσει τα αναγνωριστικά εφαρμόζοντας πάνω τους τη συνάρτηση $modulo(2^m)$, επιστρέφοντας έτσι αριθμούς στο κλειστό διάστημα $[0, 2^m - 1]$. Διαισθητικά, ένα αναγνωριστικό κάποιου κλειδιού key, έστω $SHA1(key)$ εάν η συνάρτηση κατακερματισμού που χρησιμοποιούμε είναι η $SHA1$, ανατίθεται στο πρώτο κόμβο μετά το $SHA1(key)$, ακολουθώντας την φορά των δεικτών του ρολογιού (clockwise, cw), πάνω στον ιδεατό δακτύλιο αναγνωριστικών. Πιο αυστηρά, το κλειδί key θα ανατεθεί στο πρώτο κόμβο (πάνω στον ιδεατό δακτύλιο) του οποίου το αναγνωριστικό είναι ίσο ή ακολουθεί το αναγνωριστικό του κλειδιού, $SHA1(key)$. Ο κόμβος στον οποίο έχει ανατεθεί το κλειδί key, καλείται σύμφωνα με το πρωτόκολλο, διάδοχος κόμβος του key και συμβολίζεται με $successor(key)$.

Στο Σχήμα 2.7, βλέπουμε ένα παράδειγμα του τρόπου ανάθεσης αναγνωριστικών κλειδιών σε κόμβους στο σύστημα Chord. Χρωματισμένοι εμφανίζονται οι τρεις κόμβοι του συστήματος, με αναγνωριστικό 0, 1 και 3. Στο παράδειγμα, μπορούμε να δούμε ότι ο διάδοχος κόμβος του κλειδιού 6, δηλαδή ο επόμενος κόμβος σε φορά αντίστοιχη των δεικτών του ρολογιού είναι ο κόμβος 0. Αντίστοιχα, για το κλειδί 1, ο διάδοχος είναι ο κόμβος 1 και για το κλειδί 2, ο κόμβος 3. Στο συγκεκριμένο δίκτυο, τα κλειδιά 4 ως και 7 και 7 ως 0 έχουν για διάδοχο κόμβο τον 0, το κλειδί 1 έχει για διάδοχο κόμβο τον 1, ενώ τα κλειδιά 2 και 3 τον κόμβο 3.

Όπως έχουμε ήδη αναφέρει, το σύστημα Chord εκμεταλλεύεται τις δυνατότητες που προσφέρει η τεχνική του συνεπούς κατακερματισμού. Πιο συγκεκριμένα, έστω ότι ο κόμβος i συνδέεται στο δίκτυο. Τότε, ένα υποσύνολο των κλειδιών τα οποία προηγουμένως είχαν ανατεθεί στον διάδοχο κόμβο του κόμβου i, ($successor(hash(IP_i))$), θα ανατεθούν τώρα στον κόμβο i. Αντίστοιχα, όταν ένας κόμβος i αποχωρεί από το δίκτυο, τα κλειδιά που αντιστοιχούσαν σε αυτόν, τώρα ανατίθενται στον διάδοχό του.

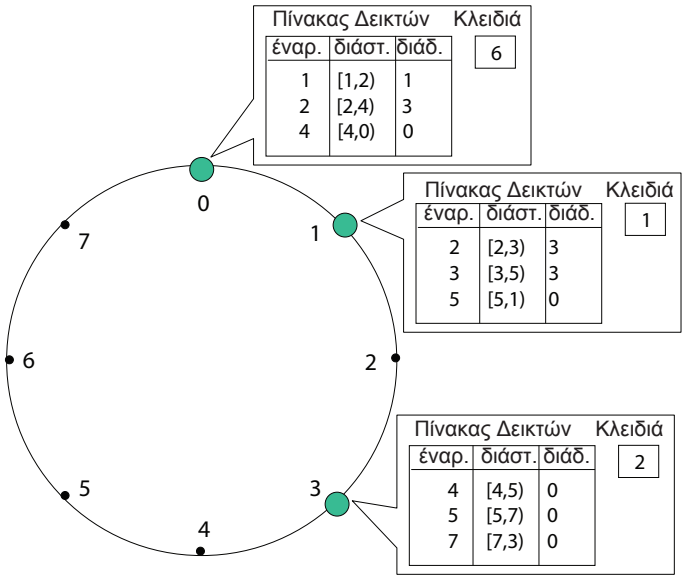


Σχήμα 2.7: Παράδειγμα δακτυλίου σε ένα ιδεατό δίκτυο Chord [55]

2.2.3.3 Πίνακες Δεικτών

Ένας κόμβος, θα πρέπει να είναι σε θέση να ικανοποιήσει οποιοδήποτε αίτημα αναζήτησης. Το Chord, για να αντιμετωπίσει αυτό το πρόβλημα, διατηρεί ένα πίνακα δρομολόγησης σε κάθε κόμβο n, με μήκος το πολύ m, ο οποίος καλείται πίνακας δεικτών (finger table). Στην i-οστή εγγραφή του, ο πίνακας περιέχει πληροφορίες για τον πρώτο κόμβο s ο οποίος διαδέχεται τον κόμβο n κατά τουλάχιστον 2^{i-1} στον ιδεατό δακτύλιο του δικτύου. Με άλλα λόγια s = successor($n + 2^{i-1}$), όπου $1 \leq i \leq 2^m$. Η καταχώρηση αυτή, καλείται και i-οστός δείκτης (i-th finger). Με το σκεπτικό αυτό, ο πρώτος δείκτης σε ένα πίνακα κάποιου κόμβου n, απέχει τουλάχιστον $n + 2^0 = n + 1$. Έτσι, ο πρώτος δείκτης (first-finger) είναι στην ουσία ο διάδοχος του κόμβου. Είναι σημαντικό να σημειωθεί, ότι όλες οι πράξεις στο $Chord$ γίνονται πάντα $modulo(2^m)$.

Στο Σχήμα 2.8, μπορούμε να δούμε ένα παράδειγμα πινάκων δεικτών σε ένα δίκτυο του συστήματος Chord. Ο κάθε πίνακας, περιέχει τρεις στήλες. Η πρώτη στήλη (έναρξη) αφορά τα αναγνωριστικά $(n + 2^{i-1})mod2^m$, όπου n ο κόμβος και i η γραμμή του πίνακα. Η τρίτη στήλη (διάδοχος), αφορά το διάδοχο κόμβο για το αναγνωριστικό έναρξης, δηλαδή για την i στήλη, *διάδοχος*$_i$ = *διάδοχος(έναρξη*$_i$*)*. Η δεύτερη στήλη, για την i εγγραφή, αφορά το διάστημα από την έναρξη της i εγγραφής, έως την έναρξη της επόμενης, κλειστό από κάτω και ανοικτό από πάνω, δηλαδή *[έναρξη*$_i$*, έναρξη*$_{i+1}$*)*.



Σχήμα 2.8: Παράδειγμα δακτυλίου με πίνακα δεικτών σε ένα δίκτυο Chord [55]

Προφανώς, μόνο και μόνο γνωρίζοντας το διάδοχό του, σε O(n) βήματα μπορεί κάθε κόμβος να αντιστοιχήσει ένα οποιοδήποτε κλειδί στο σωστό κόμβο. Όμως, το Chord, μπορεί να το πετύχει πολύ πιο γρήγορα. Αναλογιζόμενοι ότι κάθε πίνακας κρατάει $O(logN)$ εγγραφές, το εξής θεώρημα προκύπτει[55]:

Θεώρημα. *Με μεγάλη πιθανότητα, ο αριθμός των κόμβων με τους οποίους θα πρέπει να επικοινωνήσουμε για την εύρεση ενός διαδόχου σε ένα δίκτυο N-κόμβων είναι* $O(logN)$.

Ο όρος "με μεγάλη πιθανότητα" προφανώς δεν είναι ξεκάθαρος. Περισσότερα σχετικά με αυτή την ανάλυση μπορούν να βρεθούν στο άρθρο που αφορά το Chord [55].

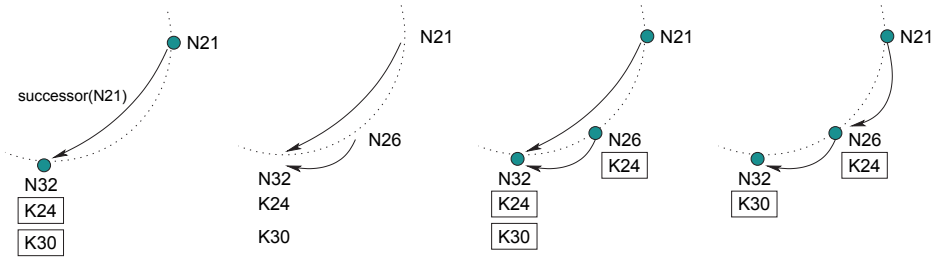
2.2.3.4 Αλλαγή στη Σύνθεση του Δικτύου: Συνδέσεις και Αποχωρήσεις

Όσον αφορά τη σύνδεση ενός κόμβου στο δίκτυο Chord, για ευκολία κάθε κόμβος κρατάει ένα δείκτη στο προηγούμενο κόμβο (predecessor). Με τον όρο προηγούμενο κόμβο, εννοούμε τον αμέσως προηγούμενο κόμβο στον ιδεατό δακτύλιο, με φορά αντίστροφη του ρολογιού (counter clockwise). Κατά την σύνδεση του κόμβου n στο δίκτυο, αρχικοποιείται ο δείκτης στο προηγούμενο κόμβο του n, καθώς και ο πίνακας δεικτών του. Στη συνέχεια, οι υπόλοιποι κόμβοι του δικτύου αναβαθμίζονται έτσι ώστε να αντικατοπτρίζουν την προσθήκη του καινούργιου κόμβου, καθώς και να ειδοποιούνται οι εφαρμογές που τρέχουν πάνω από το Chord. Για τη σύνδεση στο δίκτυο, ο κόμβος n χρησιμοποιεί κάποια γνώση για κάποιο κόμβο ο οποίος ήδη ανήκει στο

δίκτυο n', και από τον οποίο προμηθεύεται τις πληροφορίες που χρειάζεται για να συνδεθεί στο δίκτυο (Σχήμα 2.9).

Κατά τη λειτουργία του δικτύου, διάφοροι αλγόριθμοι τρέχουν περιοδικά για τη συντήρησή του. Ο αλγόριθμος σταθεροποίησης (stabilization algorithm) για παράδειγμα, ζητάει από τον διάδοχό του s να του επιστρέψει τον προκάτοχό του. Εάν ο προκάτοχος του s είναι διαφορετικός από τον n, τότε αυτό σημαίνει ότι είναι ένας καινούργιος κόμβος ο οποίος εισήλθε στο δίκτυο. Σε περίπτωση αποτυχίας κάποιου κόμβου, διατηρείται μια λίστα διαδόχων (successor list), η οποία διατηρεί τους r διαδόχους ενός κόμβου στο δίκτυο. Έτσι, σε περίπτωση αποτυχίας κάποιου κόμβου, ο προηγούμενος του αντιλαμβάνεται την αποτυχία του, και τον αντικαθιστά με την επόμενη εγγραφή του πίνακα διαδόχων, η οποία στην ουσία αποθηκεύει τον διάδοχο του διαδόχου του. Στο [56], αποδεικνύεται ακόμα ότι και σε ένα δίκτυο όπου ο κάθε κόμβος αποτυγχάνει με πιθανότητα 0.5 (επομένως, μπορούμε να θεωρήσουμε ενα δίκτυο όπου οι μισοί κόμβοι αποτυγχάνουν), διατηρώντας πίνακες διαδόχων με μέγεθος $r = \Omega(log(N))$, με μεγάλη πιθανότητα δεν θα υπάρχουνε λάθη στην εύρεση διαδόχων. Σε περίπτωση ομαλής αποχώρησης απο το δίκτυο, τα μηνύματα που αποστέλλονται φροντίζουν τη διατήρηση της συνέπειας στο δίκτυο. Στο [55] αποδεικνύεται το εξής θεώρημα:

Θεώρημα. *Με μεγάλη πιθανότητα, ο αριθμός του μηνυμάτων τα οποία πρέπει να αποσταλούν για τη συντήρηση ενός δικτύου με N κόμβους, κατά την σύνδεση ή αποσύνδεση ενός κόμβου από το δίκτυο, είναι $O(log^2 N)$.*



Σχήμα 2.9: Διαδικασία σύνδεσης του κόμβου 26 σε ένα δίκτυο Chord, χρησιμοποιώντας τον κόμβο 32 [55]

2.2.4 Pastry

Το σύστημα Pastry [52], είναι ένα άλλο σύστημα που προσφέρει ένα δίκτυο ομότιμων και εκμεταλλεύεται τη λειτουργικότητα των κατανεμημένων πινάκων κατακερματισμού. Το σύστημα αυτό είναι το σύστημα το οποίο χρησιμοποιήθηκε στην υλοποίηση της πτυχιακής εργασίας.

2.2.4.1 Εισαγωγικά

Το σύστημα $Pastry$ είναι ένα πλήρως αποκεντρωμένο, κλιμακούμενο και αυτό-οργανωμένο σύστημα που προσφέρει ένα δίκτυο ομότιμων κόμβων, το οποίο προσφέρει τη δυνατότητα δρομολόγησης μηνυμάτων σε επίπεδο εφαρμογής. Υπάρχουν ήδη εφαρμογές κτισμένες πάνω στο $Pastry$, οι οποίες εκμεταλλεύονται τη λειτουργικότητα που προσφέρει, όπως το σύστημα $PAST$, το οποίο είναι ένα σύστημα καθολικής αποθήκευσης, καθώς και το σύστημα $SCRIBE$, το οποίο είναι ένα κλιμακούμενο σύστημα δημοσίευσης / συνδρομής (Publish/Subscribe).

Το $Pastry$, αναθέτει σε κάθε κόμβο ένα μοναδικό αναγνωριστικό (nodeId). Όταν η εφαρμογή η οποία είναι κτισμένη πάνω από το Pastry, θέλει να μεταβιβάσει ένα μήνυμα χρησιμοποιώντας ένα κλειδί (αναγνωριστικό), το Pastry αναλαμβάνει να δρομολογήσει το μήνυμα στο κόμβο του δικτύου ο οποίος βρίσκεται πιο κοντά στο αναγνωριστικό από ότι οποιοσδήποτε άλλος κόμβος. Μάλιστα, το $Pastry$ μπορεί να δρομολογήσει σε οποιονδήποτε από τους k κοντινότερου κόμβους σε κάποιο αναγνωριστικό, γεγονός το οποίο εκμεταλλεύεται το $PAST$, το οποίο διαμοιράζει k αντίγραφα του αρχείου (replicas), στους k αυτούς κόμβους, διατηρώντας έτσι αντίγραφα για ασφάλεια. Όσον αφορά την έννοια της απόστασης, θα δούμε πώς το Pastry φροντίζει να ελαχιστοποιήσει τις μετρικές απόστασης, βασιζόμενο σε κάποια ευρετικά.

2.2.4.2 Μορφολογία του Pastry

Όπως έχουμε αναφέρει, κάθε κόμβος στο δίκτυο Pastry αντιστοιχείται σε ένα αναγνωριστικό (NodeId), 128 bit. Στη τοπολογία του Pastry, το αναγνωριστικό αυτό αντιστοιχεί στη θέση του κόμβου, σε ένα κυκλικό χώρο αναγνωριστικών[10], ο οποίος εκτείνεται από το 0 έως το $2^{128} - 1$. Το αναγνωριστικό αυτό, μπορεί να ανατίθεται τυχαία κατά την σύνδεση ενός κόμβου στο δίκτυο, για παράδειγμα χρησιμοποιώντας μια κρυπτογραφική συνάρτηση κατακερματισμού.

[10] Η διαφορά με τον ιδεατό δακτύλιο του Chord αφορά την παράλειψη των *modulo* υπολογισμών και την εισαγωγή της 2^b βάσης

Υποθέτοντας ότι αναφερόμαστε σε ένα δίκτυο N κόμβων, το δίκτυο μπορεί να δρομολογήσει ένα μήνυμα με αναγνωριστικό $h(key)$ στον κόμβο ο οποίος έχει το αριθμητικά πλησιέστερο αναγνωριστικό σε $\lceil log_b N \rceil$ βήματα. (Το b είναι μια παράμετρος η οποία τυπικά παίρνει τη τιμή 4). Η παράδοση στο σωστό κόμβο είναι εγγυημένη, εκτός εάν $\lfloor \frac{L}{2} \rfloor$ παρακείμενοι (adjacent) κόμβοι αποτύχουν ταυτόχρονα. Πάλι, το L είναι μια παράμετρος με τυπικές τιμές από 16 έως 32.

Τα αναγνωριστικά στο Pastry θεωρούνται σαν μια σειρά ψηφίων στη βάση 2^b. Για να εξασφαλίσει το $Pastry$ ότι το μήνυμα θα καταλήξει στο σωστό κόμβο, κάθε βήμα της δρομολόγησης λειτουργεί ως εξής: Έστω το μήνυμα με αναγνωριστικό K το οποίο βρίσκεται στο κόμβο με αναγνωριστικό N. Έστω ότι το κοινό πρόθεμα του αναγνωριστικού K και του N, έχει μήκος x ψηφία. Ο επόμενος κόμβος στον οποίο θα δρομολογηθεί το μήνυμα, θα έχει αναγνωριστικό το οποίο θα μοιράζεται τουλάχιστον $x + 1$ ψηφία με το K, πάντα στη βάση 2^b.

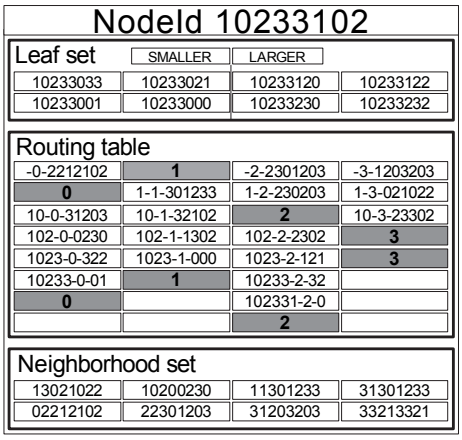
2.2.4.3 Η Δομή ενός Κόμβου

Ένας κόμβος στο Pastry διατηρεί τοπικά κάποιες πληροφορίες για το δίκτυο. Οι πληροφορίες αυτές περιλαμβάνουν το πίνακα δρομολόγησης (routing table), το σύνολο γειτονίας (neighborhood set) και το σύνολο φύλλων (leaf set).

Ο πίνακας δρομολόγησης R ενός κόμβου περιέχει $\lceil log_b N \rceil$ γραμμές και $2^b - 1$ στήλες. Αυτές οι $2^b - 1$ στήλες, έστω στη n-οστή γραμμή στο κόμβο Z, αναφέρονται σε ένα κόμβο X του οποίου το αναγνωριστικό μοιράζεται n ψηφία με τον κόμβο στον οποίο ανήκει ο πίνακας, αλλά διαφέρει στο $n + 1$ ψηφίο το οποίο έχει μια από τις $2^b - 1$ υπόλοιπες τιμές, εκτός από τη τιμή που έχει το $n + 1$ ψηφίο στον κόμβο Z. Ένα παράδειγμα τέτοιου πίνακα παρουσιάζεται στο Πίνακα 2.5.

Όσον αφορά τη παράμετρο b, αυτή λειτουργεί ρυθμιστικά, "ανταλλάσσοντας" (tradeoff) την αύξηση του μεγέθους του πίνακα δρομολόγησης με το μέγιστο αριθμό μεταβιβάσεων (hops) που χρειάζονται για τη δρομολόγηση μεταξύ ενός οποιουδήποτε ζευγαριού κόμβων. Κάποιες χαρακτηριστικές τιμές παρουσιάζονται στο Πίνακα 2.4.

Το σύνολο γειτονίας περιλαμβάνει τους M κόμβους με αναγνωριστικά τα οποία βρίσκονται πιο κοντά στο κόμβο στον οποίο ανήκει το σύνολο, ενώ το σύνολο φύλλων είναι το σύνολο με τους $\lceil \frac{L}{2} \rceil$ κόμβους οι οποίοι είναι αριθμητικά πιο κοντά αλλά μεγαλύτεροι από το αναγνωριστικό του



Σχήμα 2.10: Η δομή ενός κόμβου στο σύστημα Pastry [52]. Βλέπουμε ένα κόμβο με αναγνωριστικό $nodeId$ = 10233102, b = 2 και l = 8. Αφού το b = 2, η βάση του συστήματος και επομένως οι αριθμοί είναι σε βάση $2^2 = 4$. Επίσης, η αρίθμηση των γραμμών αρχίζει από το μηδέν (zero based). Τα σκιασμένα κελιά σε κάθε γραμμή, δείχνουν το ψηφίο του αναγνωριστικού του κόμβου στο οποίο αντιστοιχούν. Ο διαχωρισμός των αναγνωριστικών σε κάθε γραμμή έχει τη μορφή *κοινό πρόθεμα με 10233102-επόμενο ψηφίο-υπόλοιπο αναγνωριστικό*. Οι IP διευθύνσεις παραλείπονται.

Πίνακας 2.4: Παραμετροποίηση του Pastry

Παράμετρος b	Κόμβοι Δικτύου	Μέσο μέγεθος πίνακα δρομολόγησης	Μεταβιβάσεις (Hops)
4	10^6	75	5
4	10^9	105	7

κόμβου και τους $\lceil \frac{L}{2} \rceil$ οι οποίοι είναι αριθμητικά πιο κοντά αλλά μικρότεροι από το αναγνωριστικό. Τυπικές τιμές για τα M και L, είναι 2^b ή $2x2^b$. Μπορούμε να δούμε ένα στιγμιότυπο της δομής κάποιου κόμβου στο Σχήμα 2.10.

2.2.4.4 Δρομολόγηση

Στον Αλγόριθμο 2.3 [52], παρουσιάζεται ο αλγόριθμος δρομολόγησης στο Pastry. Αρχικά, ας ορίσουμε τη σημειολογία του αλγορίθμου:

- Με D συμβολίζουμε το αναγνωριστικό - κλειδί του μηνύματος, ενώ με A το αναγνωριστικό του κόμβου στο οποίο φτάνει το μήνυμα. Σε αυτή τη περίπτωση, καλείται και ο αλγόριθμος.

- Στο R_l^i, αντιστοιχούμε το πίνακα δρομολόγησης R, στην i-οστή στήλη και l-οστή γραμμή. Σημειώνεται ότι $1 \leq i < 2^b$ και $0 \leq l < \lfloor \frac{128}{b} \rfloor$.
- Το L_i αντιστοιχεί τον i-οστό κοντινότερο αριθμητικά κόμβο, ο οποίος ανήκει στο σύνολο φύλλων. Ισχύει ότι $-\lfloor |\frac{L}{2}| \rfloor \leq i \leq \lfloor |\frac{L}{2}| \rfloor$
- Ο δείκτης i στο D_i αναφέρεται στο i-οστό ψηφίο του αναγνωριστικού D.
- Η συνάρτηση $shl(A, B)$, αναφέρεται στο μέγεθος σε ψηφία του κοινού προθέματος μεταξύ των αναγνωριστικών A και B.

Αλγόριθμος 2.3 Route(Msg with key D): Αλγόριθμος Δρομολόγησης στο Pastry

```
1: if L_{-⌊|L|/2⌋} ≤ D ≤ L_{⌊|L|/2⌋} then
2:    //D is withing range of our leaf set
3:    forward to L_i s.th |D − L_i| is minimal
4: else
5:    //use the routing table
6:    let l = shl(D, A)
7:    if R_l^{D_l} ≠ null then
8:       forward to R_l^{D_l}
9:    else
10:      //rare case
11:      forward to T ∈ L ∪ R ∪ M s.th.
12:      shl(T, D) ≥ l
13:      |T − D| < |A − D|
14:   end if
15: end if
```

Βλέπουμε ότι αρχικά ο κόμβος ελέγχει εάν μπορεί να προωθήσει το μήνυμα σε κάποιον από τους κόμβους που ανήκουν στο σύνολο φύλλων του. Αυτό, μπορεί να το εξακριβώσει χρησιμοποιώντας το αναγνωριστικό D του μηνύματος. Εάν ανήκει στο σύνολο φύλλων, τότε προωθεί το μήνυμα στο κόμβο που ανήκει στο σύνολο αυτό, και έχει τη μικρότερη αριθμητικά απόσταση από το αναγνωριστικό του μηνύματος D (Γραμμές 1-3).

Σε διαφορετική περίπτωση, ο κόμβος θα πρέπει να αναφερθεί στο πίνακα δρομολόγησης και να προωθήσει το μήνυμα σε κάποιο κόμβο ο οποίος αναφέρεται στο πίνακα δρομολόγησής του, και μοιράζεται τουλάχιστον ένα περισσότερο ψηφίο στο πρόθεμα του με το κόμβο προορισμό.

Πίνακας 2.5: Ένας πίνακας δρομολόγησης με αναγνωριστικό κόμβου $65a1x$ με $b = 4$ στο Pastry [11]. Τα ψηφία είναι στο δεκαεξαδικό σύστημα.

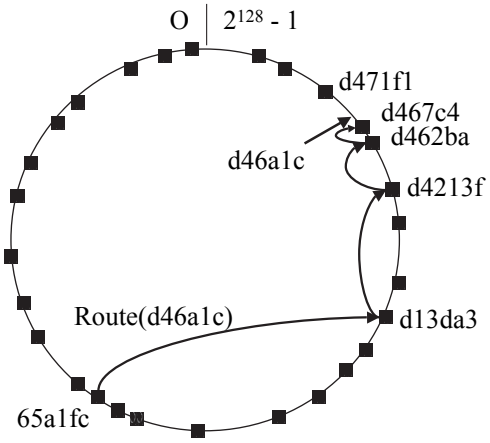
0 x	*1 x*	*2 x*	*3 x*	*4 x*	*5 x*		*7 x*	*8 x*	*9 x*	*a x*	*b x*	*c x*	*d x*	*e x*	*f x*
6 0 x	*6 1 x*	*6 2 x*	*6 3 x*	*6 4 x*		*6 6 x*	*6 7 x*	*6 8 x*	*6 9 x*	*6 a x*	*6 b x*	*6 c x*	*6 d x*	*6 e x*	*6 f x*
6 5 0 x	*6 5 1 x*	*6 5 2 x*	*6 5 3 x*	*6 5 4 x*	*6 5 5 x*	*6 5 6 x*	*6 5 7 x*	*6 5 8 x*	*6 5 9 x*		*6 5 b x*	*6 5 c x*	*6 5 d x*	*6 5 e x*	*6 5 f x*
6 5 a 0 x		*6 5 a 2 x*	*6 5 a 3 x*	*6 5 a 4 x*	*6 5 a 5 x*	*6 5 a 6 x*	*6 5 a 7 x*	*6 5 a 8 x*	*6 5 a 9 x*	*6 5 a a x*	*6 5 a b x*	*6 5 a c x*	*6 5 a d x*	*6 5 a e x*	*6 5 a f x*

Αυτό καλύπτεται στις γραμμές 6-8. Στις σπάνιες περιπτώσεις όπου οι αντίστοιχες καταχωρήσεις στο πίνακα δρομολόγησης είναι κενές ή ο κόμβος στον οποίο θα περνούσε το μήνυμα δεν μπορεί να ανταποκριθεί, τότε ο κόμβος αναλαμβάνει να προωθήσει το μήνυμα σε κάποιο κόμβο ο οποίος έχει τουλάχιστον ίσα ψηφία κοινό πρόθεμα με το αναγνωριστικό όσο ο ίδιος, αλλά έχει μικρότερη αριθμητική απόσταση από αυτό (γραμμές 11-14). Ένας τέτοιος κόμβος πρέπει να υπάρχει μέσα στο σύνολο φύλλων, εκτός εάν $\frac{L}{2}$ κόμβοι αποτύχουν ταυτόχρονα, οπότε η ύπαρξη τέτοιου κόμβου δεν είναι εγγυημένη. Ένα παράδειγμα δρομολόγησης παρουσιάζεται στο Σχήμα 2.11.

Είναι φανερό από τη περιγραφή και μόνο του αλγορίθμου, ότι ο αλγόριθμος συγκλίνει στο κόμβο προορισμό. Αυτό γίνεται γιατί σε κάθε βήμα, ή προωθούμε σε ένα κόμβο με κοινό πρόθεμα αναγνωριστικού κόμβου και αναγνωριστικού κλειδιού τουλάχιστον κατά ένα ψηφίο μεγαλύτερο από το παρών κόμβο ή στη χειρότερη περίπτωση, με ίδια κοινά ψηφία αλλά μικρότερη αριθμητική διαφορά.

2.2.4.5 Αποδοτικότητα Δρομολόγησης

Με δεδομένο ότι δεν έχουν συμβεί πρόσφατες αποτυχίες στους κόμβους και οι πίνακες δρομολόγησης είναι ακριβείς, μπορεί να δειχθεί ότι ο αναμενόμενος αριθμός βημάτων κατά τη δρομολόγηση είναι $\lceil log_{2^b} N \rceil$. Εάν υποθέσουμε ότι σε κάθε περίπτωση ο κόμβος στον οποίο θα δρομολογήσουμε βρίσκεται στο πίνακα δρομολόγησης, τότε σε κάθε βήμα, το σύνολο τον κόμβων το οποίο έχει μεγαλύτερο κοινό πρόθεμα με το αναγνωριστικό μειώνεται κατά 2^b, και



Σχήμα 2.11: Δρομολόγηση ενός μηνύματος σε δακτύλιο δικτύου $Pastry$ [11]

επομένως, τα βήματα είναι λογαριθμικά με βάση λογαρίθμου το παράγοντα 2^b. Προφανώς, εάν το προορισμός ανήκει στο σύνολο φύλλων, τότε απλά χρειάζεται ένα βήμα. Η περίπτωση ο επόμενος (σύμφωνα με τους κανόνες) κόμβος να μην βρεθεί ούτε στο σύνολο φύλλων, ούτε στο πίνακα δρομολόγησης, εξαρτάται από τη παράμετρο $|L|$. Σύμφωνα με το [52], για L ίσο με 2^b και $2 * 2^b$ αντίστοιχα, έχουμε πιθανότητες τόσο μικρές όσο 0.2 και 0.006. Με μεγάλη πιθανότητα, ακόμα και αυτό να συμβεί, θα χρειαστούμε ακόμα ένα βήμα.

Στη περίπτωση όπου έχουμε να αντιμετωπίσουμε πολλές και ταυτόχρονες αποτυχίες κόμβων, ο αριθμός των βημάτων μπορεί το πολύ να είναι ίσος με N. Όπως έχουμε προαναφέρει, η παράδοση του μηνύματος είναι σε κάθε περίπτωση εγγυημένη εάν δεν αποτύχουν ταυτόχρονα $\frac{L}{2}$ κόμβοι, κάτι που επιδεικνύει τη σημασία που έχει η επιλογή της παραμέτρου L.

2.2.4.6 Αυτό-οργάνωση και προσαρμογή

Άφιξη Κόμβου

Προφανώς, κατά την άφιξη ενός κόμβου στο δίκτυο, ο κόμβος αυτός θα πρέπει να ενημερώσει τις τοπικές δομές του, καθώς και να πληροφορήσει άλλους κόμβους για την ύπαρξή του. Θεωρούμε, ότι ο κόμβος γνωρίζει την ύπαρξη ενός άλλου γειτονικού κόμβου Α, ο οποίος είναι ήδη μέρος του συστήματος.

Έστω ότι στο καινούργιο κόμβο ανατίθεται ένα αναγνωριστικό X (για παράδειγμα, χρησιμοποιώντας το αποτέλεσμα μιας κρυπτογραφικής συνάρτησης). Ο καινούργιος κόμβος, ζητάει

από τον κόμβο A ο οποίος ανήκει ήδη στο σύστημα να διαβιβάσει ένα μήνυμα που δηλώνει την πρόθεσή του να συνδεθεί στο δίκτυο (join message), με αναγνωριστικό κλειδί ίσο με το αναγνωριστικό του X. Το μήνυμα, όπως κάθε μήνυμα στο $Pastry$, θα διαβιβαστεί σε κάποιο κόμβο Z, ο οποίος έχει αναγνωριστικό πιο κοντινό στο αναγνωριστικό X του νεοεισερχόμενου κόμβου. Στη συνέχεια, οι κόμβοι A ως Z και όσοι κόμβοι μεσολάβησαν στο μονοπάτι από τον A στον Z, αποστέλλουν τις πληροφορίες οι οποίες είναι αποθηκευμένες τοπικά στο κόμβο τους στον Q. Ο Q, μπορεί να ζητήσει πίνακες από επιπρόσθετους κόμβους εάν χρειάζεται, συμπληρώνει τους πίνακές του και στη συνέχεια πληροφορεί τους κόμβους οι οποίοι χρειάζεται να μάθουν για την σύνδεσή του. Σημειώνεται, ότι στην σπάνια περίπτωση που ο X έχει το ίδιο αναγνωριστικό με τον Z, θα πρέπει να ανατεθεί καινούργιο αναγνωριστικό σε αυτόν.

Πιο συγκεκριμένα, αφού ο A είναι γειτονικός κόμβος του X, ο X μπορεί να αρχικοποιήσει το σύνολο γειτονίας του χρησιμοποιώντας το σύνολο γειτονίας του A. Ο Z έχει το κοντινότερο αναγνωριστικό στο αναγνωριστικό του X και συνεπώς ο Q χρησιμοποιεί το σύνολο φύλλων του σαν βάση για τη δημιουργία του δικού του. Όσον αφορά τη κατασκευή του πίνακα δρομολόγησης, αυτή αρχίζει από τη μηδενική γραμμή (πρώτη). Θα θεωρήσουμε τη πιο γενική περίπτωση, όπου τα αναγνωριστικά των X και A δεν έχουν κανένα κοινό πρόθεμα. Έστω A_i, η i-οστή γραμμή του πίνακα δρομολόγησης του κόμβου A. Οι καταχωρήσεις σε αυτή τη γραμμή δεν έχουν κανένα κοινό πρόθεμα με το αναγνωριστικό του, και επομένως η A_i περιέχει κατάλληλες τιμές για το X_i. Οι άλλες γραμμές του πίνακα δρομολόγησης του A δεν έχουνε καμία χρήση για το X αφού έχουμε υποθέσει ότι οι κόμβοι δεν έχουν κοινό πρόθεμα. Μπορούμε όμως να αναλογιστούμε το εξής: Σε κάθε βήμα της δρομολόγησης του μηνύματος από τον κόμβο A στον Z, ο επόμενος κόμβος έχει ένα κοινό ψηφίο με τον X, ο δεύτερος 2 κ.ο.κ., με δεδομένο ότι κάθε φορά, με περιβάλλον χωρίς αποτυχίες, ο επόμενος κόμβος έχει ένα ψηφίο κοινό με το X περισσότερο από το προηγούμενο. Συνεπώς η i-οστή γραμμή του πίνακα δρομολόγησης του X αντιστοιχεί σε πληροφορίες που λαμβάνονται από τον i-οστό κόμβο στο μονοπάτι A-Z.

Τελικά, ο κόμβος X μεταδίδει αντίγραφα των τοπικών πληροφοριών που έχει συλλέξει, σε όλους τους κόμβους στο πίνακα δρομολόγησης, σύνολο γειτονίας και σύνολο φύλλων του. Το συνολικό κόστος των μηνυμάτων που έχουν ανταλλαχθεί είναι $O(log_{2^b}N)$.

Αποχώρηση Κόμβου

Ένας κόμβος θεωρείται ότι έχει αποχωρήσει από το δίκτυο, εάν οι άμεσοι γείτονες του στο χώρο αναγνωριστικών του Pastry δεν μπορούν να επικοινωνήσουν μαζί του. Η επικοινωνία αυτή

γίνεται περιοδικά για τη διατήρηση της συνέπειας του δικτύου. Για την αντικατάσταση ενός κόμβου ο οποίος έχει αποχωρήσει και ανήκει στο σύνολο φύλλων, ο γείτονάς του στο χώρο διευθύνσεων επικοινωνεί με το ζωντανό κόμβο με το μεγαλύτερο αριθμό i ($index$) στη πλευρά του κόμβου που έχει αποτύχει στο σύνολο φύλλων. Για παράδειγμα, αν ο L_i έχει αποτύχει για $\lfloor -|\frac{L}{2}| \rfloor < i < 0$, τότε θα επικοινωνήσει με το κόμβο με το μεγαλύτερο $|i|$ στη μεριά των $i < 0$, δηλαδή το $L_{-|\frac{L}{2}|}$ από όπου θα λάβει το σύνολο φύλλων και θα διορθώνει το δικό του σύνολο φύλλων. Όσον αφορά το πίνακα δρομολόγησης, ο κόμβος επικοινωνεί με άλλους κόμβους στην ίδια γραμμή με το κόμβο που απέτυχε, με σκοπό να αντικαταστήσει τη στήλη που έχει αποτύχει. Το σύνολο γειτονίας αναβαθμίζεται μέσω μηνυμάτων στα άλλα μέλη του συνόλου.

Επανάκαμψη κόμβου

Σημαντική είναι η έννοια της επανάκαμψης κόμβου (node recovery). Έχουν αναπτυχθεί αλγόριθμοι οι οποίοι μειώνουν το κόστος (overhead) που προκύπτει από προσωρινές αποτυχίες (σφάλματα) που παρουσιάζουν κάποιοι κόμβοι. Ένας κόμβος που επανακάμπτει στο Pastry, αρχικά προσπαθεί να επικοινωνήσει με τους κόμβους οι οποίοι βρίσκονται στο πιο πρόσφατο σύνολο φύλλων του, και να ανακτήσει το σύνολο φύλλων κάθε ενός από αυτούς. Αν το αριθμητικό εύρος των αναγνωριστικών των κόμβων σε κάποιο από αυτά τα σύνολα ακόμα περιλαμβάνει το αναγνωριστικό του κόμβου που επανακάμπτει, τότε ο κόμβος ανανεώνει το σύνολο φύλλων του βασίζοντας την ανανέωση στις πληροφορίες που λαμβάνει, και στη συνέχεια ειδοποιεί τα μέλη του αναβαθμισμένου σύνολου φύλλων του για τη παρουσία του. Διαφορετικά, θα αναγκαστεί να ακολουθήσει τη κανονική διαδικασία πρόσθεσης κόμβου στο δίκτυο.

2.2.5 PAST

Όπως έχουμε αναφέρει, το σύστημα PAST [53] είναι ένα σύστημα φτιαγμένο πάνω από το Pastry. Στο PAST, αναθέτονται στους κόμβους αποθήκευσης (δηλαδή σε κόμβους οι οποίοι συμμετέχουν και συνεισφέρουν στο αποθηκευτικό μέγεθος του συστήματος) και στα αρχεία μοναδικά αναγνωριστικά, με μια ομοιόμορφη κατανομή, καθώς και αντίγραφα (replicas) του αρχείου στους αντίστοιχους κόμβους, δημιουργώντας έτσι ένα αυτο-οργανωμένο δίκτυο επικάλυψης. Οι δύο βασικές λειτουργίες που ορίζει το PAST, είναι οι λειτουργίες τις εισαγωγής και αναζήτησης, όπως ορίζονται στο [53]:

- $fileId$ = Insert($name$, $owner-credentials$, k, $file$)
  Η μέθοδος της εισαγωγής, αποθηκεύει το αρχείο με όνομα $name$, σε ένα αριθμό k κόμβων

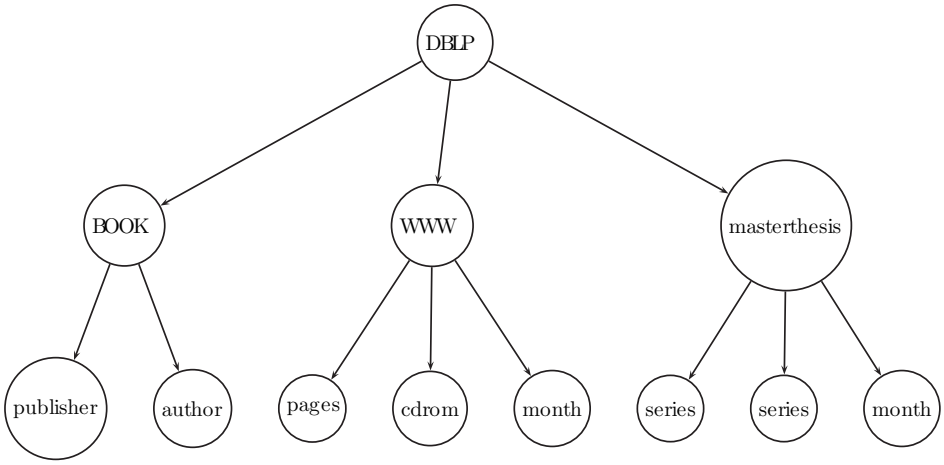
στο δίκτυο $Pastry$ πάνω από το οποίο τρέχει το $PAST$. Η διαδικασία, παράγει ένα αναγνωριστικό πεδίο 160 bit (το οποίο επιστρέφει ως $fileId$), το οποίο μπορεί να χρησιμοποιηθεί για το χαρακτηρισμό του αρχείου. Για τη παραγωγή του $fileId$, χρησιμοποιείται η $SHA-1$ συνάρτηση κατακερματισμού, με παραμέτρους το όνομα του αρχείου, το δημόσιο κλειδί (public key) του ιδιοκτήτη και κάποιους τυχαίους αριθμούς. Η επιλογή αυτή εγγυάται πως με πολύ μεγάλη πιθανότητα, τα αναγνωριστικά θα είναι μοναδικά. Παρ' όλη τη σπανιότητα του περιστατικού, υπάρχει η πιθανότητα δύο αναγνωριστικά να συμπίπτουν. Σε αυτή τη περίπτωση, το δεύτερο χρονικά αρχείο που θα εισαχθεί απορρίπτεται από το σύστημα.

- $file$ = Lookup($fileId$)
  Η μέθοδος αυτή, αξιόπιστα ανακτά ένα αντίγραφο του αρχείου από το οποίο παράχθηκε το αναγνωριστικό $fileId$ το οποίο δέχεται σαν παράμετρο, πάντα με την προυπόθεση ότι το αρχείο αυτό υπάρχει στο δίκτυο σε ένα τουλάχιστον από τους k κόμβους που το αποθηκεύουν. Θεωρητικά, το αρχείο θα ανακτηθεί από τον κόμβο ο οποίο βρίσκεται " κοντινότερα" στο κόμβο ο οποίος ζητάει το αρχείο.

2.2.5.1 Εξισορρόπηση Φόρτου (Load Balancing)

Η εξισορρόπηση φόρτου (ή Load Balancing), είναι μια τεχνική η οποία τείνει να κατανέμει "δίκαια" το φόρτο ενός συστήματος το οποίο αποτελείται από επι-μέρους οντότητες και πόρους, έτσι ώστε να μεγιστοποιήσει την απόδοση του συστήματος στη μονάδα του χρόνου, δηλαδή τη ρυθμαπόδοση (throughput).

Το πρόβλημα αυτό στα δίκτυα ομότιμων, μπορούμε να το συναντήσουμε με δύο τρόπους: Αρχικά, τη κατανομή δεδομένων στους κόμβους. Κάθε κόμβος, μπορεί να αποθηκεύει κάποια δεδομένα ή να υπεύθυνος για κάποια δεδομένα, ανάλογα με το αναγνωριστικό του. Αυτό το κομμάτι, το φροντίζει το ίδιο το Pastry, και στη περίπτωσή μας το $PAST$. Για παράδειγμα, στο $PAST$ τα αναγνωριστικά αρχείων που παράγονται, είναι ομοιόμορφα κατανεμημένα στο χώρο αναγνωριστικών του, προσφέροντας έτσι μια καλή λύση σε αυτή τη συνιστώσα. Το άλλο πρόβλημα αφορά κόμβους οι οποίοι περιέχουν κάποια πληροφορία (για παράδειγμα, κάποιο αρχείο στο $PAST$), στην οποία γίνονται πολλές αιτήσεις για την ανάκτησή της. Έτσι, παρόλο που οι πληροφορίες είναι κατανεμημένες ομοιόμορφα στο δίκτυο, ένα δημοφιλής αρχείο, για παράδειγμα, μπορεί να προκαλέσει μια άνιση επιβάρυνση στον αντίστοιχο κόμβο (ή κόμβους). Το $PAST$, αναλαμβάνει σε αυτή τη περίπτωση να επιλύσει το πρόβλημα. Πέρα από το μηχανισμό $replication$, όπου ένα αρχείο βρίσκεται αποθηκευμένο, εφαρμόζει ένα μηχανισμό "κρυφής



Σχήμα 2.12: Η δομή του δέντρου XML για το αρχείο $dblp.xml$ (αγνοώντας τα γνωρίσματα)

μνήμης" (caching), έτσι ώστε αρχεία στα οποία υπάρχει συχνή πρόσβαση να αποθηκεύονται προσωρινά για γρήγορη ανάκτηση σε κόμβους οι οποίοι διαθέτουν κάποιο ελεύθερο χώρο, καθώς αυτό το αρχείο δρομολογείται στο δίκτυο.

Σε αυτό το σημείο τελειώνουμε την επισκόπηση της θεματικής περιοχής που αφορά τα δίκτυα ομότιμων και τις τεχνολογίες των ΚΠΚ. Στη συνέχεια θα περάσουμε στην επισκόπηση της γλώσσας XML, περιγράφοντας κυρίως τα σημεία που θα μας φανούν χρήσιμα στη περιγραφή του συστήματος, όπως τη σωστή δημιουργία XML εγγράφων και τη γλώσσα XQuery.

2.3 Το μοντέλο δεδομένων XML

Το μοντέλο δεδομένων XML (Extensible Markup Language), ορίζει ένα απλό και αφηρημένο μοντέλο, το οποίο μπορεί εύκολα να επεκταθεί και να προσαρμοστεί στις ανάγκες τις εφαρμογής που θα το υιοθετήσει. Μπορούμε να φανταστούμε ένα έγγραφο XML σαν τη "γραμμική περιγραφή μιας δενδρικής δομής, όπου η πληροφορία που βρίσκεται στους κόμβους μαζί με την ίδια τη δομή του δέντρου εκφράζουν ακριβώς τις πληροφορίες που περιγράφονται σε ένα XML έγγραφο" [17]. Ένα παράδειγμα της αντιστοιχίας αυτής, βρίσκεται στα Σχήματα 2.12 και 2.13.

```
<dblp>
   <book key="defaultCDATA0">
      <publisher>level 2</publisher>
      <author>level 2</author>
   </book>
   <www key="defaultCDATA2">
      <pages>level 2</pages>
      <cdrom>level 2</cdrom>
      <month>level 2</month>
   </www>
   <mastersthesis key="defaultCDATA3">
      <series href="defaultCDATA4">level 2</series>
      <series href="defaultCDATA5">level 2</series>
      <month>level 2</month>
   </mastersthesis>
</dblp>
```

Σχήμα 2.13: Το έγγραφο $dblp.xml$

Η επεκτάσιμη γλώσσα σήμανσης (XML, Hypertext Markup Language) έχει κάποια σημαντικά προτερήματα. Δεν έχει κάποια προκαθορισμένη δομή που θα μπορούσε να περιορίσει το χρήστη της, αλλά προσφέρει τη δυνατότητα στους χρήστες της να προσδιορίσουν τα δικά στοιχεία, όπως αυτοί θέλουν. Ένα καλό παράδειγμα, είναι ότι ένας χημικός μπορεί να αναπαραστήσει τα διάφορα στοιχεία που θα αντιπροσωπεύουν άτομα, μόρια, δεσμούς και αντιδράσεις, ενώ ένας κτηματομεσίτης μπορεί να περιγράψει διαμερίσματα, ενοίκια, προμήθειες και τοποθεσίες [32]. Οι γλώσσες που έχουν αυτό το χαρακτηριστικό λέγονται γλώσσες μετασήμανσης (metamarkup languages). Εκτός από την προσαρμοστικότητα, η γλώσσα αφορά αρχεία κειμένου, κάτι που επιτρέπει την εύκολη ανάγνωσή τους, αλλά και την μεταφορά τους σε πολλά λειτουργικά συστήματα, χωρίς τεχνικά προβλήματα. Στην ουσία, η γλώσσα XML παρέχει τη δυνατότητα δημιουργίας άλλων γλωσσών, βασισμένων σε κανόνες και αρχές που τις διέπουν, όπως για παράδειγμα τη WML (Wireless Markup Language) η οποία χρησιμοποιείται για τη μορφοποίηση και σήμανση εφαρμογών Διαδικτύου για συσκευές χεριού, και τη XUL (XML User Interface Langauge), μια γλώσσα η οποία έχει αναπτυχθεί από τη κοινότητα Mozilla, για τη δημιουργία διεπαφών σε εφαρμογές.

Η ορθότητα των XML εγγράφου, διαχωρίζεται σε δυο επίπεδα: Τα καλά ορισμένα (well defined) και τα έγκυρα (valid). Τα καλά ορισμένα, πρέπει να υπακούν σε όλους τους κανόνες οι οποίοι ορίζονται για το XML[11]. Τα έγκυρα, πρέπει να υπακούν σε κάποιους κανόνες οι οποίοι

[11]Οι προδιαγραφές (specifications) για το XML format , υπάρχουν στο [18]

ορίζονται από το χρήστη ή περιλαμβάνονται σε κάποιο αρχείο Ορισμού Τύπου Εγγράφου (Document Type Definition, DTD).

Πίνακας 2.6: Τύποι χαρακτηριστικών XML

Τύπος Χαρακτηριστικού	Καθορίζει
value1 | value 2 |...	λίστα τιμών που παίρνει χαρακτηριστικό
CDATA	κείμενο αλφαριθμητικών χαρακτήρων
ID	τιμή που προσδιορίζει μοναδικά το στοιχείο
IDREF	Αναφορά σε ID άλλου στοιχείου
IDREFS	λίστα από IDREF τύπους
ENTITY	όνομα οντότητας που ορίζεται στο DTD
ENTITIES	λίστα από ENTITY τύπους
NMTOKEN	έγκυρο όνομα XML
NMTOKENS	λίστα από NMTOKEN τύπους
NOTATION	όνομα που συμβολίζει μια συγκεκριμένη ενέργεια

2.3.1 Σύνταξη

Ένα στοιχείο (element) ορίζεται με τη χρήση ετικετών. Οι ετικέτες, δηλώνουν την έναρξη και τη λήξη του στοιχείου αντίστοιχα, ως εξής:

$$< dblp >, < /dblp > \tag{2.1}$$

Η σύνταξη ενός στοιχείου, είναι η ακόλουθη:

```
1 <name attribute="value">content</name>
```

Βλέπουμε ότι ο χρήστης, μπορεί να ορίσει πέρα από το όνομα του στοιχείου (name), ένα αριθμό απο γνωρίσματα ή χαρακτηριστικά (attributes) που το συνοδεύουν, καθώς και το περιεχόμενό του (content). Ο ορισμός ενός XML εγγράφου είναι αναδρομικός· ένα στοιχείο δηλαδή βρίσκεται μεταξύ των ετικετών έναρξης και λήξης κάποιου άλλου. Αυτή η σχέση δηλώνει και την ιεραρχική παράσταση ενός XML εγγράφου σε δέντρο. Το βάθος της αναδρομής του στοιχείου, στην ουσία ορίζει και τη θέση του, στο αντίστοιχο (ιδεατό) δέντρο.

Στη συνέχεια, θα δώσουμε ένα παράδειγμα αποτύπωσης μιας βάσης δεδομένων σε ενα αρχείο XML:

```
<database>
<item key="123"></key>
<item key="222"></key>
<item key="333"></key>
</database>
```

Κάθε αντικείμενο στη βάση αναπαριστάται από ένα στοιχείο, το οποίο έχει σαν γνώρισμα το κλειδί. Έτσι, χρησιμοποιούμε την ετικέτα του στοιχείου για να αναφερθούμε στο εκάστοτε στοιχείο, όπως κάλλιστα θα μπορούσαμε να δημιουργούσαμε ένα επιπλέον εμφωλιασμένο στοιχείο κάτω από το στοιχείο item και να το ονομάζαμε key, αν και σε τέτοια περίπτωση θα ήτανε δυσκολότερο να διακρίνουμε το στοιχείο που ψάχνουμε.

Ο Mozilla Firefox[α΄] είναι ένας από τους δημοφιλέστερους πλοηγητές (browsers) που υπάρχουν. Χρησιμοποιεί σε μεγάλο βαθμό ένα τύπο XML, ο οποίος έχει οριστεί από τη κοινότητα Mozilla που αναπτύσσει τον πλοηγητή, και αφορά την παρουσίαση της διεπαφής του στον χρήστη. Ο τύπος καλείται XUL, από τα αρχικά XML User Interface Language, δηλαδή γλώσσα διεπαφής XML. Με βάση αυτή τη γλώσσα, οι χρήστες μπορούν να προσφέρουν επιπρόσθετη λειτουργία στο πλοηγητή και να φτιάξουν τη διεπαφή της προέκτασης αυτής. Όμως αυτή δεν είναι η μόνη χρήση του XML στο πλοηγητή. Πόροι που χρησιμοποιεί, όπως εικόνες, καθώς και πληροφορίες για οντότητες που ανήκουν στο σύστημα, παρουσιάζονται με τη μορφή XML αρχείων. Ακολουθεί ένα παράδειγμα της γλώσσας XUL από μια επέκταση (add-on) του Firefox.

```
<tooltip id="uoamail-tooltip" orient="vertical">
                    <description id ="uoamail-tooltip-l" value="Not Logged In" />
                    <description id ="uoamail-tooltip-d1" value="Initialized" />
                    <description id ="uoamail-tooltip-d2" value="UoAmail" />
                    <description id ="uoamail-tooltip-d3" value="xx:xx" style="color: red;"/>
</tooltip>
```

[α΄] www.firefox.com/

2.3.1.1 Βασικότεροι Κανόνες Σύνταξης

Στη συνέχεια, θα αναφερθούμε σε κάποιους από τους βασικότερους κανόνες σύνταξης της γλώσσας:

- Όλα τα στοιχεία της XML πρέπει να έχουν ετικέτες τέλους, εκτός από τη δήλωση:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Η οποία δεν θεωρείται τμήμα του XML εγγράφου, και έχει την εξής σύνταξη:

```
<?xml version="number" [ encoding="encoding" ] [standalone = yes|no]?>
```

με το πρώτο χαρακτηριστικό να δείχνει την έκδοση, το δεύτερο, το οποίο είναι και προαιρετικό, να δείχνει την κωδικοποίηση, ενώ το τρίτο, πάλι προαιρετικό, να παίρνει τη τιμή yes εάν χρησιμοποιείται κάποιο DTD.

- Όταν κάποιο στοιχείο δεν έχει περιεχόμενο, η ετικέτα έναρξης μπορεί να συμπτυχθεί με την ετικέτα λήξης, δηλαδή το:

  ```
  <nocontentforme></nocontentforme>
  ```

  αντικαθιστάτε από το:

  ```
  <nocontentforme/>
  ```

- Οι ετικέτες είναι case-sensitive, δηλαδή διαφέρουν αν είναι γραμμένες σε κεφαλαία ή μικρά γράμματα.

- Σε συνέπεια με τη δενδρική αναπαράσταση του εγγράφου, θα πρέπει να υπάρχει ένα μοναδικό στοιχείο ρίζα, ενώ τα υπόλοιπα στοιχεία θα ορίζονται σαν παιδιά - περιεχόμενο της ρίζας.

- Κάθε στοιχείο μπορεί να έχει στοιχεία παιδιά, και κάθε στοιχείο μπορεί να έχει γνωρίσματα, των οποίων η τιμή αναφέρεται μέσα σε εισαγωγικά.

- Τα σχόλια στην XML έχουν την εξής σύνταξη:

  ```
  <!-- I 'm a comment! -->
  ```

2.3.2 Ορθότητα και Εγκυρότητα

Όπως έχουμε ήδη πει, υπάρχουν τα καλά ορισμένα και τα έγκυρα έγγραφα XML. Ένα έγγραφο, το οποίο υπακούει τους κανόνες που έχουμε ορίσει μέχρι τώρα, είναι καλά ορισμένο. Για να είναι το έγγραφο έγκυρο, θα πρέπει να ικανοποιεί κάποιους περιορισμούς οι οποίοι θέτονται σε αυτό. Για παράδειγμα, μπορεί να περιορίζει τις τιμές που μπορούν να πάρουν οι ετικέτες ή και οποιοδήποτε άλλο περιορισμό που αφορά τη δομή και το περιεχόμενό του, όπως για παράδειγμα να προσδιορίζει στοιχεία τα οποία είναι απαραίτητο να υπάρχουν στο έγγραφο. Ακολουθεί ένα παράδειγμα DTD εγγράφου:

```
1  <!ELEMENT person (name+, profession*, phone?)>
2  <!ELEMENT name EMPTY>
3  <!ATTLIST name first CDATA #REQUIRED
4                 last  CDATA #REQUIRED>
5  <!ELEMENT profession EMPTY>
6  <!ATTLIST profession value CDATA #REQUIRED>
```

Κάθε γραμμή του εγγράφου, ξεκινάει με θαυμαστικό (!). Στη συνέχεια, ακολουθεί μία από τις δεσμευμένες λέξεις κλειδιά (keywords) όπως ELEMENT, ATTLIST, ENTITY, οι οποίες προσδιορίζουν το είδος που καθορίζουν. Για παράδειγμα, η πρώτη γραμμή χρησιμοποιεί σύμβολα τα οποία συναντούμε σε κανονικές εκφράσεις (Regular Expressions). Το $+$ σημαίνει ένα ή περισσότερα, το "$*$" μηδέν ή περισσότερα, ενώ το "?" σημαίνει ένα ή κανένα. Επομένως, με απλά λόγια, η πρώτη γραμμή ορίζει ότι ένα πρόσωπο (person), έχει ένα ή περισσότερα ονόματα, οσαδήποτε επαγγέλματα - ακόμα και κανένα, και ένα ή καθόλου τηλέφωνα. Επίσης, δηλώνει ότι όλα τα ονόματα πρέπει να προηγούνται των επαγγελμάτων, καθώς και ότι τα στοιχεία αυτά είναι άδεια, δηλαδή δεν έχουν κάποιο περιεχόμενο. Οι ορισμοί στις γραμμές 3 και 6, καθορίζουν τα γνωρίσματα. Ο τύπος τους είναι ανοικτός (CDATA) όπως και ο ορισμός REQUIRED ορίζει ότι είναι υποχρεωτικά (διάφοροι τύποι χαρακτηριστικών εμφανίζονται στο Πίνακα 2.6). Ένα έγκυρο αρχείο XML, σύμφωνα με το παραπάνω, είναι το εξής:

```
1 <person>
2   <name first="Alan" last="Turing"/>
3   <profession value="computer scientist"/>
4   <profession value="mathematician"/>
5   <profession value="cryptographer"/>
6 </person>
```

2.3.3 Η Γλώσσα XPath

Η γλώσσα XPath αποτελεί ένα εργαλείο για τη πλοήγηση στη δενδρική δομή ενός XML εγγράφου και για την επιλογή κόμβων, με βάση κάποια κριτήρια. Στην ουσία, χρησιμοποιείται σαν μια απλή γλώσσα επερωτήσεων. Μία έκφραση XPath αποτελείται από βήματα τοποθεσίας (location steps), τα οποία έχουν την εξής σύνταξη:

$$locationstep \rightarrow axis\ nodetest\ [predicate]^* \tag{2.2}$$

όπου: Axis (άξονας) μπορεί να είναι ένα από τα / ή //. Τα δύο αυτά στοιχεία, ορίζουν εάν ο κόμβος στον οποίο αναφερόμαστε μπορεί να είναι άμεσο παιδί ή απλά απόγονος του προ-

ηγούμενου κόμβου, σε οποιοδήποτε βάθος. Το nodetest, μπορεί να είναι το όνομα κάποιου κόμβου του εγγράφου, ή ο χαρακτήρας "$*$", ο οποίος αντιστοιχεί σε οποιοδήποτε όνομα κόμβου. Στη συνέχεια, ακολουθούν μηδέν ή περισσότερα κατηγορήματα (predicates). Τα κατηγορήματα έχουν τη μορφή

$$value_1 \; operator \; value_2 \tag{2.3}$$

όπου το $value_1$ μπορεί να είναι ένα στοιχείο (element), γνώρισμα ή η συνάρτηση $position()$, ο τελεστής ανήκει στο σύνολο τελεστών σύγκρισης $=, >, \geq, <, \leq, <>$ ενώ η τιμή $value_2$, μπορεί να είναι μια τιμή στοιχείου, μια τιμή γνωρίσματος ή ένας αριθμός ο οποίος αναφέρεται στη θέση ενός στοιχείου.

Μια γραμμική επερώτηση μονοπατιού (linear path query) q, αποτελείται από μια σειρά βημάτων τοποθεσίας. Κάποια παραδείγματα επερωτήσεων XPath που αναφέρονται στο [41] είναι τα εξής:

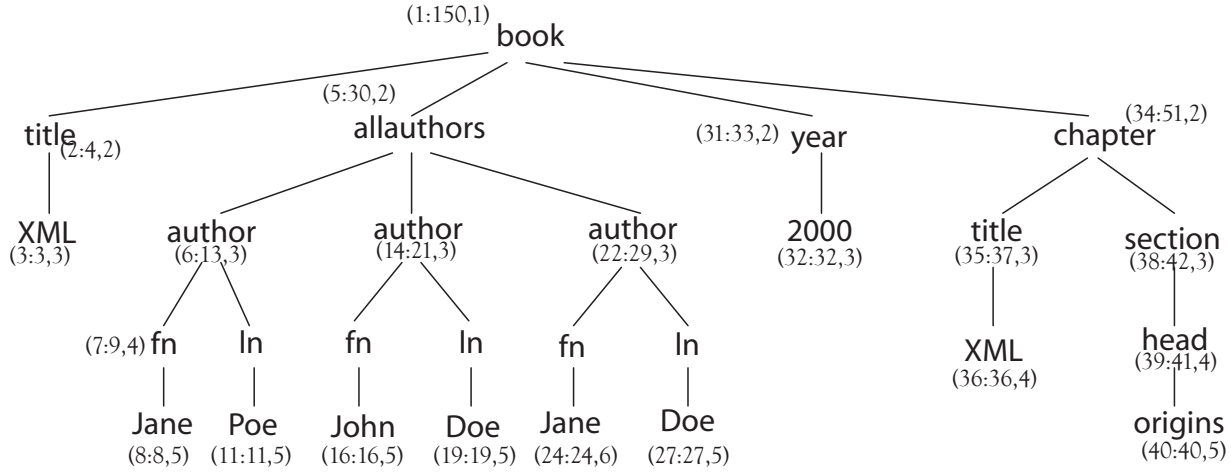
```
q1: /dblp/phdthesis/[year=2007]
q2: /dblp/*/author[@name="John Smith"]
q3: //*/[school="University of Athens"]
```

Η πρώτη επερώτηση επιλέγει κόμβους στοιχείων με όνομα phdthesis οι οποίες δημοσιεύτηκαν το 2007. Η δεύτερη επιλέγει οποιαδήποτε δημοσίευση έχει σαν author το όνομα "John Smith", ενώ η τρίτη επιλέγει οποιαδήποτε δημοσίευση έγινε από συγγραφείς που ανήκουν στο Πανεπιστήμιο Αθηνών.

2.3.4 Εμπλουτισμός XML Γεγονότων με Χαρακτηριστικά Θέσης

Στα [15, 9], παρουσιάζεται μια μέθοδος εμπλουτισμού των γεγονότων τα οποία περιέχονται σε κάποιο XML έγγραφο, με σκοπό τον αποδοτικό έλεγχο δομικών σχέσεων μεταξύ στοιχείων. Για παράδειγμα, τον έλεγχο εάν κάποιο στοιχείο είναι παιδί κάποιου άλλου, απόγονός του ή γονιός του.

Ποιο συγκεκριμένα, η μέθοδος αυτή εμπλουτίζει κάθε γεγονός του εγγράφου με τρία στοιχεία, προσθέτοντας το ζευγάρι (L:R, D) στην αναπαράσταση. Τα L και R, λαμβάνονται με την μέτρηση λέξεων από την αρχή του XML εγγράφου, μέχρι τις ετικέτες έναρξης και λήξης του στοιχείου το οποίο αφορά το ζευγάρι αντίστοιχα, ενώ το D αφορά το βάθος του στοιχείου στο έγγραφο, αν σκεφτούμε την δενδρική του μορφή. Το παράδειγμα ενός τμήματος ενός τέτοιου εγγράφου, παρουσιάζεται στο Σχήμα 2.14.



Σχήμα 2.14: Εμπλουτισμός στοιχείων XML εγγράφου με χαρακτηριστικά θέσης

Ο έλεγχος των δομικών χαρακτηριστικών μπορεί με βάση την αναπαράσταση αυτή να γίνει ως εξής: Έστω το στοιχείο - κόμβος n_1, το οποίο αντιστοιχεί στην αναπαράσταση $(L_1 : R_1, D_1)$, και ο κόμβος n_2 ο οποίος αντιστοιχεί στην $(L_2 : R_2, D_1)$. Τότε, ο κόμβος n_1 μπορεί να είναι πρόγονος του n_2 και ο n_2 απόγονος του n_1 αν και μόνο αν ισχύει $L_1 < L_2$ και $R_2 < L_1$. Αυτό πρέπει να ισχύει γιατί ο πρόγονος θα πρέπει να αρχίζει τουλάχιστον μία λέξη πριν από τον απόγονο ($L_1 < L_2$), καθώς και ο απόγονος θα πρέπει να τελειώνει πριν τελειώσει ο πρόγονος ($R_2 < R_1$). Μπορούμε εύκολα να κατανοήσουμε τη σύλληψη των κριτηρίων αυτών, εάν φανταστούμε το XML έγγραφο, και παραλληλίσουμε τη σχέση μεταξύ κόμβων στη δενδρική αναπαράσταση, η οποία αντιστοιχεί στο βάθος εμφωλιασμού στο ίδιο το έγγραφο. Προφανώς, με τον όρο τελειώνει, εννοούμε συναντάει γεγονός End of Element, ενώ με τον όρο αρχίζει, Start of Document, κάτι που αντιστοιχεί στις ετικέτες τέλους και έναρξης. Επίσης, για να συμπεράνουμε ότι ένας κόμβος είναι γονιός κάποιου απογόνου του, θα πρέπει το D του γονιού να είναι μικρότερο κατά μία μονάδα αυτού του απογόνου, αφού ισχύει $D_{parent} = D_{child} - 1$.

Για παράδειγμα, μπορούμε να παρατηρήσουμε τις σχέσεις στο Σχήμα 2.14. Ο κόμβος $author$, με στοιχεία $(6 : 13, 3)$, είναι απόγονος του κόμβου $book$, με στοιχεία $(1 : 150, 1)$, αφού ισχύει, ότι $L_{book} = 1 < 6 = L_{author}$ και $R_{author} = 13 < 150 = r_{book}$. Επίσης, μπορεί να παρατηρηθεί, ότι ο κόμβος $author$ είναι γονιός του κόμβου fn με χαρακτηριστικά $(7 : 9, 4)$, αφού $L_{author} = 6 < 7 = L_{fn}$ και $R_{fn} = 9 < 13 = R_{author}$ και $D_{fn} = 4 = 3 + 1 = D_{author} + 1$.

Έχοντας περιγράψει και το μοντέλο δεδομένων XML, στη συνέχεια θα περιγράψουμε κάποια στοιχεία από τη θεωρία αυτομάτων τα οποία θα χρησιμοποιήσουμε για να περιγράψουμε την αναπαράσταση των επερωτήσεων XQuery στο σύστημα.

2.4 Μη ντετερμινιστικά πεπερασμένα αυτόματα

Ένα μη ντετερμινιστικό πεπερασμένο αυτόματο, είναι ένα βασικό εργαλείο του τομέα της *Θεωρίας Υπολογισμού*. Στην ουσία, είναι μια μηχανή καταστάσεων, που έχει το χαρακτηριστικό ότι μπορεί να υπάρχουν πολλές επόμενες καταστάσεις για ένα ζευγάρι *(κατάστασης, συμβόλου)*, οπότε υπάρχουν και πολλά μονοπάτια που αντιστοιχούν σε μια είσοδο στο αυτόματο. Από αυτό το χαρακτηριστικό τους, πηγάζει και η δικαιολόγηση του μη ντετερμινισμού: Θεωρητικά, το αυτόματο μπορεί να "μαντέψει" και να ακολουθήσει το σωστό μονοπάτι εκτέλεσης στο αυτόματο, εφ' όσον αυτό υπάρχει στο αυτόματο. Έχει όμως αποδειχτεί, ότι κάθε μη ντετερμινιστικό πεπερασμένο αυτόματο, είναι ισοδύναμο με κάποιο ντετερμινιστικό πεπερασμένο αυτόματο (δηλαδή μια μηχανή καταστάσεων που για ένα ζευγάρι (κατάστασης, συμβόλου) παράγει ακριβώς μία επόμενη κατάσταση), και ως ισοδύναμα παράγουν την ίδια γλώσσα, η οποία είναι και κανονική (Regular).

Σύμφωνα με το [34], ένα μη ντετερμινιστικό πεπερασμένο αυτόματο ορίζεται σαν μια πεντάδα

$$A = (Q, \Sigma, \delta, q_0, F) \tag{2.4}$$

όπου το Q αποτελεί ένα πεπερασμένο σύνολο καταστάσεων, το Σ, ένα πεπερασμένο σύνολο συμβόλων εισόδου, το $q_0 \in Q$ αντιστοιχεί στην αρχική κατάσταση (start state), ενώ το σύνολο $F \subseteq Q$, είναι ένα σύνολο τελικών καταστάσεων *ή* καταστάσεων αποδοχής (accept states). Μια τελική κατάσταση (accept state) σε ένα αυτόματο, σηματοδοτεί ότι η συμβολοσειρά εισόδου, εφ' όσον έχει εξαντληθεί, έγινε αποδεκτή από το αυτόματο. Σημειώνεται, ότι στα μη ντετερμινιστικά πεπερασμένα αυτόματα, υπάρχει ένα επιπλέον σύμβολο, το οποίο συμβολίζει τη κενή μετάβαση (e-transition), δηλαδή τη μετάβαση σε μια κατάσταση χωρίς τη κατανάλωση συμβόλου από την είσοδο. Η κατάσταση αυτή συμβολίζεται σαν "ϵ". Τέλος, έχουμε τη συνάρτηση μετάβασης δ, η οποία δέχεται σαν είσοδο μια κατάσταση από το Q και ένα σύμβολο από το Σ, και επιστρέφει ένα υποσύνολο του Q, το οποίο ουσιαστικά αφορά τις επόμενες καταστάσεις της κατάστασης εισόδου, με την κατανάλωση του συμβόλου εισόδου. Συχνά, η σχέση δ δίνεται μέσω ενός πίνακα.

Η επεκτεταμένη συνάρτηση μετάβασης $\hat{\delta}$, διαφοροποιεί τη συνάρτηση μετάβασης ως εξής: αντί να δέχεται στην είσοδό της ένα σύμβολο, δέχεται μια συμβολοσειρά. Εφαρμόζει διαδοχικά την δ, σε κάθε σύμβολο της συμβολοσειράς και επιστρέφει το τελικό σύνολο καταστάσεων. Τυπικά, η επεκτεταμένη συνάρτηση μετάβασης, ορίζεται αναδρομικά ως εξής:
βάση ορισμού:

$$\hat{\delta}(q, \epsilon) = q \tag{2.5}$$

Επαγωγικό βήμα: Έστω ότι η συμβολοσειρά εισόδου w της συνάρτησης, είναι της μορφής $w = xa$, όπου a το τελικό σύμβολο και x η υπόλοιπη συμβολοσειρά. Έστω ότι

$$\hat{\delta}(q, x) = p_1, p_2, ..., p_k \tag{2.6}$$

και ότι

$$\bigcup_{i=1}^{k} \delta(p_i, a) = r_1, r_2, ..., r_m \tag{2.7}$$

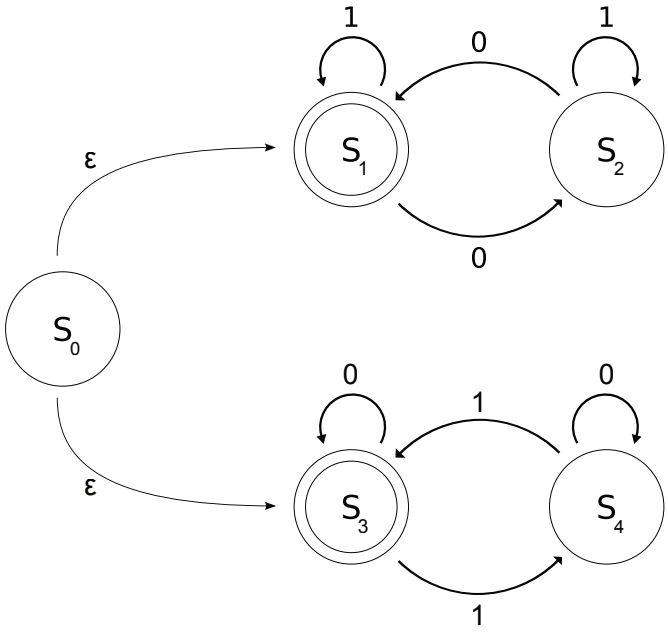
Τότε, $\hat{\delta}(q, w) = (r_1, r_2, ..., r_m)$.

Όπως έχουμε πει, ένα μη ντετερμινιστικό πεπερασμένο αυτόματο παράγει μια γλώσσα, η οποία είναι κανονική. Πιο φορμαλιστικά, ένα τέτοιο αυτόματο δέχεται μία συμβολοσειρά w, εάν υπάρχει τουλάχιστον μία διαδρομή στο αυτόματο (μονοπάτι), το οποίο καταναλώνοντας τα σύμβολα της συμβολοσειράς θα μας οδηγήσει από την αρχική κατάσταση σε κάποια τελική κατάσταση, χωρίς να μας ενδιαφέρει για τα μονοπάτια για τα οποία αποτυγχάνει να γίνει αυτό. Δηλαδή, εάν έχουμε το αυτόματο $A = (Q, \Sigma, \delta, q_0, F)$, τότε η γλώσσα του συμβολίζεται με $L(A)$ και ορίζεται σαν:

$$L(A) = \{w | \hat{\delta}(q_o, w) \cap F \neq \emptyset\} \tag{2.8}$$

Πίνακας 2.7: Πίνακας για το αυτόματο του Σχήματος 2.15

	0	**1**	ϵ
S_0	{}	{}	$\{S_1, S_3\}$
S_1	$\{S_2\}$	$\{S_1\}$	{}
S_2	$\{S_1\}$	$\{S_2\}$	{}
S_3	$\{S_3\}$	$\{S_4\}$	{}
S_4	$\{S_4\}$	$\{S_3\}$	{}



Σχήμα 2.15: Μη ντετερμινιστικό πεπερασμένο αυτόματο που αναγνωρίζει τη γλώσσα των δυαδικών αριθμών με ίσο αριθμό από 1 ή 0

Μπορούμε να δούμε τη λειτουργία ενός τέτοιου αυτόματου, μελετώντας τον Πίνακα 2.7. Ο πίνακας, παρουσιάζει τα αποτελέσματα της συνάρτησης δ με είσοδο τη κατάσταση που βρίσκεται στην i οστή γραμμή και το σύμβολο στην j οστή στήλη. Το ίδιο το αυτόματο παρουσιάζεται στο Σχήμα 2.15. Το αυτόματο αυτό αναγνωρίζει τη γλώσσα των δυαδικών συμβολοσειρών (δηλαδή με αλφάβητο το 0 και το 1) οι οποίες έχουν ίσο αριθμό από 0 ή ίσο αριθμό από 1. Μπορούμε να δούμε ότι το αυτόματο είναι μη ντετερμινιστικό, γιατί π.χ. η αρχική κατάσταση S_0 με είσοδο τη κενή συμβολοσειρά (ϵ) οδηγεί σε δύο καταστάσεις, την S_1 και S_2.

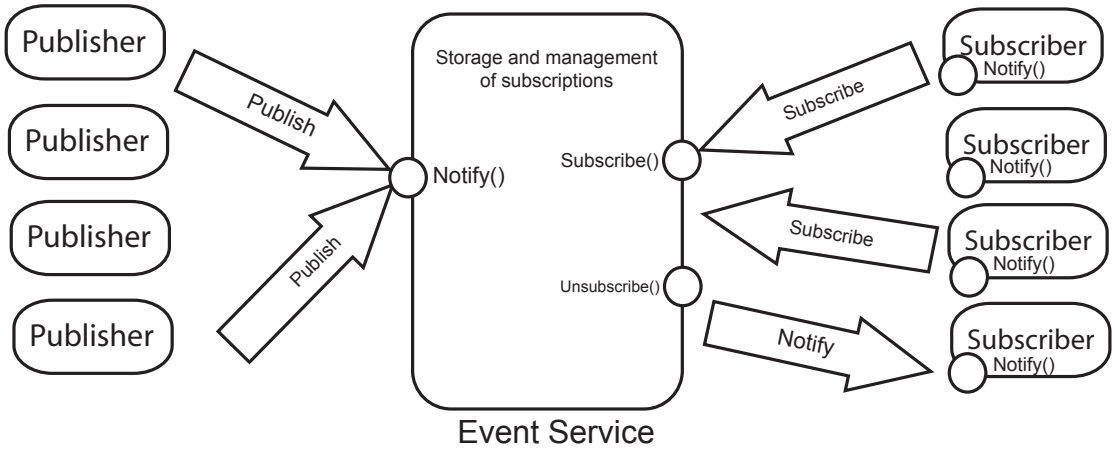
Έχοντας περιγράψει και τα απαραίτητα στοιχεία από τη Θεωρία Υπολογισμού, στη συνέχεια θα εισάγουμε επίσημα τα συστήματα δημοσίευσης / συνδρομής, έτσι ώστε να μπορούμε να περιγράψουμε το μοντέλο δημοσίευσης / συνδρομής στο οποίο στηρίζεται το σύστημα που υλοποιούμε καθώς και τις λεπτομέρειες που το αφορούν.

2.5 Συστήματα Δημοσίευσης / Συνδρομής

Το μοντέλο δημοσίευσης / συνδρομής, είναι ένα μοντέλο καλά προσαρμοσμένο στην φύση των κατανεμημένων συστημάτων μεγάλης-κλίμακας (large scale), και έτσι λαμβάνει μεγάλο ερε-

υνητικό ενδιαφέρων. Σε συστήματα βασισμένα σε αυτό το μοντέλο διάδρασης, οι συνδρομητές εκδηλώνουν το ενδιαφέρον τους σε κάποια γεγονότα, ή ένα μοτίβο (pattern) από γεγονότα, και στη συνέχεια ειδοποιούνται ασύγχρονα για γεγονότα τα οποία ικανοποιούν τα κριτήριά τους και εκδίδονται από εκδότες. Πολλές παραλλαγές του μοντέλου έχουν προταθεί, η κάθε μία προσαρμοζόμενη σε κάποιο μοντέλο η εφαρμογή. Είναι δεδομένο ότι το Διαδίκτυο και ο Παγκόσμιος Ιστός (World Wide Web, WWW) έχει αλλάξει τη μορφή των κατανεμημένων συστημάτων τα τελευταία χρόνια, ωθώντας έτσι τα κατανεμημένα συστήματα να περιλαμβάνουν χιλιάδες οντότητες, πιθανά κατανεμημένες γύρω από το διαδίκτυο. Όπως έχουμε αναφέρει στην ενότητα όπου μελετούμε τα δίκτυα ομότιμων, οι κόμβοι και οι οντότητες που συντελούν αυτά τα συστήματα είναι συνήθως αναξιόπιστα, με περιορισμένους πόρους. Το μοντέλο δημοσίευσης / συνδρομής προσφέρει μια "χαλαρά συζευγμένη" (loosely coupled) μορφή επικοινωνίας η οποία απαιτείται σε τέτοια μεγάλης κλίμακας περιβάλλοντα. Η δύναμη του μοντέλου διάδρασης αυτού, βασίζεται στη πλήρη αποσύνδεση (full decoupling) σε χρόνο, χώρο και συγχρονισμό μεταξύ των συνδρομητών και των εκδοτών.

2.5.1 Το Βασικό Μοντέλο Διάδρασης

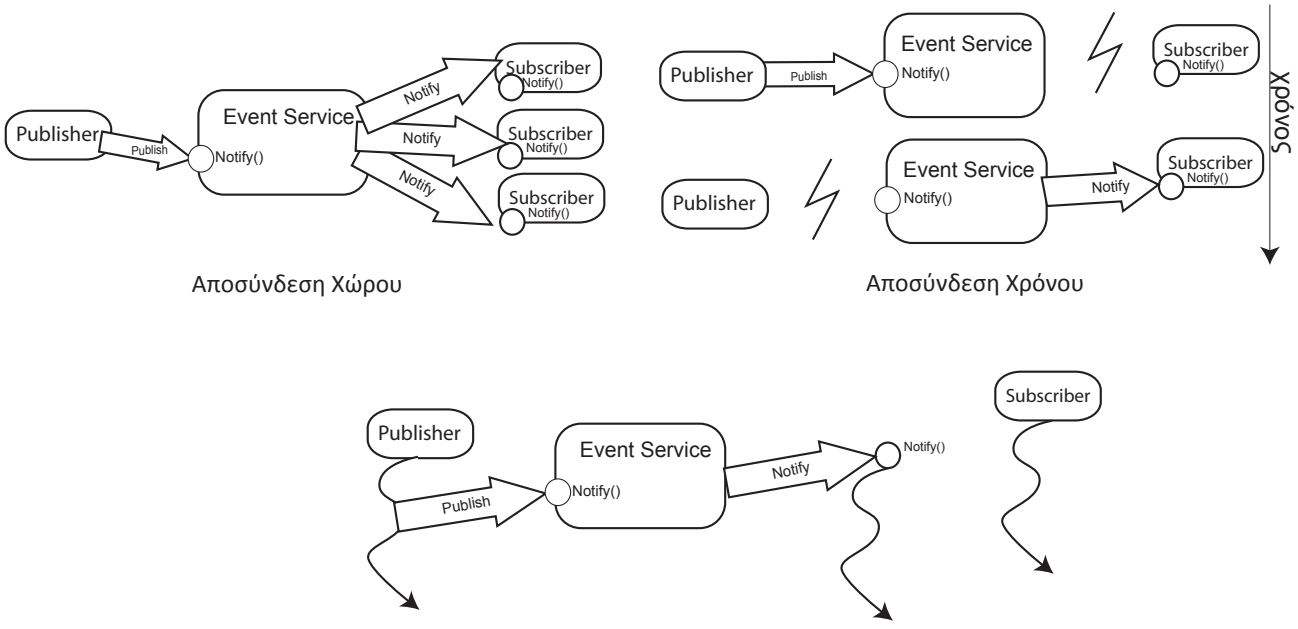


Σχήμα 2.16: Το βασικό μοντέλο δημοσίευσης / συνδρομής

Με απλά λόγια, το μοντέλο δημοσίευσης / συνδρομής παρέχει τη δυνατότητα στους εκδότες να εκδίδουν πληροφορίες και στους συνδρομητές να λαμβάνουν πληροφορίες οι οποίες τους ενδιαφέρουν. Η πληροφορία τυπικά καλείτε γεγονός (event) και η πράξη της παράδοσης της πληροφορίας *ειδοποίηση* (notification). Στο Σχήμα 2.16, βλέπουμε το βασικό σχήμα διάδρασης σε

ένα σύστημα δημοσίευσης / συνδρομής, το οποίο στηρίζεται πάνω σε μια υπηρεσία διαχείρισης γεγονότων, η οποία παρέχει χώρο και διαχειρίζεται τις συνδρομές και την αποδοτική παράδοση των γεγονότων. Γενικότερα, οι συνδρομητές εκδηλώνουν το ενδιαφέρων τους καλώντας μια μέθοδο $subscribe$ στην υπηρεσία γεγονότων, χωρίς να γνωρίζουν τις πηγές που μπορεί να τους παρέχουν πληροφορίες. Η πληροφορία συνδρομής αποθηκεύεται στο σύστημα και δεν προωθείται στους εκδότες, ενώ μια διαδικασία unsubscribe, τερματίζει τη συνδρομή.

Για τη δημιουργία ενός γεγονότος, ένας εκδότης τυπικά καλεί μια διαδικασία $publish()$. Η υπηρεσία γεγονότων μεταφέρει το γεγονός σε όλους τους συνδρομητές οι οποίοι ενδιαφέρονται· μπορεί έτσι να θεωρηθεί σαν μία πληρεξούσια (proxy) υπηρεσία. Σημειώνεται, ότι *όλοι* οι συνδρομητές θα ειδοποιηθούν για την έκδοση ενός γεγονότος που τους ενδιαφέρει, ενώ συχνά, οι εκδότες έχουν τη δυνατότητα να καλέσουν μια διαδικασία $advertise()$, η οποία μπορεί να πληροφορήσει συνδρομητές για τυχόν περιεχόμενο μελλοντικών γεγονότων, αλλά και για να γνωστοποιήσει τη ροή δεδομένων στην υπηρεσία γεγονότων.



Σχήμα 2.17: Αποσύνδεση χωρική, χρονική και συγχρονισμού στο μοντέλο δημοσίευσης / συνδρομής

Η αποσύνδεση που προσφέρει η υπηρεσία γεγονότων μεταξύ συνδρομητών και εκδοτών, μπορεί να διαιρεθεί σε τρεις διαστάσεις, όπως παρουσιάζεται στο Σχήμα 2.17:

- *Αποσύνδεση Χώρου (Space Decoupling)*: Οι οντότητες που συμμετέχουν στην διάσταση, δεν χρειάζεται να γνωρίζουν η μία την ύπαρξη της άλλης. Οι εκδότες, δημοσιεύουν μέσω της υπηρεσίας γεγονότων και οι συνδρομητές ανακτούν τα γεγονότα άμεσα, μέσω της υπηρεσίας. Οι εκδότες συνήθως δεν γνωρίζουν ποιοι είναι οι συνδρομητές, ούτε πόσοι συμμετέχουν στη διάδραση. Ομοίως, οι συνδρομητές δεν γνωρίζουν τον αριθμό ή την ταυτότητα των συνδρομητών.

- *Αποσύνδεση Χρόνου (Time Decoupling)*: Οι οντότητες που συμμετέχουν στο σύστημα, δεν χρειάζεται να είναι ενεργά συμμετέχοντες στη διάδραση το ίδιο χρονικό διάστημα. Συγκεκριμένα, ο εκδότης μπορεί να εκδίδει καθώς οι συνδρομητές δεν είναι συνδεδεμένοι και αντίστοιχα, ο συνδρομητής να ειδοποιηθεί για κάποιο γεγονός που τον ενδιαφέρει, καθώς ο εκδότης δεν είναι συνδεδεμένος.

- *Αποσύνδεση Συγχρονισμού (Synchronization Decoupling)*: Οι εκδότες δεν μπλοκάρουν (block) καθώς παράγουν γεγονότα, και οι συνδρομητές μπορεί να ειδοποιηθούν ασύγχρονα μέσω κάποιας μεθόδου *callback* για κάποιο γεγονός, ενώ διενεργούν σύγχρονα (concurrent) κάποια άλλη διαδικασία. Επίσης, η παραγωγή / κατανάλωση των γεγονότων, δεν συμβαίνει στη κύρια ροή ελέγχου των εκδοτών / συνδρομητών, και έτσι δεν συμβαίνει με σύγχρονο τρόπο.

Η ιδιότητα αυτή της αποσύνδεσης, σύμφωνα με το [23], αυξάνει τη κλιμάκωση του συστήματος, αφαιρώντας όλες τις ρητές εξαρτήσεις μεταξύ των οντοτήτων. Κατ᾽ ακρίβεια, η αφαίρεση αυτών των εξαρτήσεων, θέτει το μοντέλο δημοσίευσης / συνδρομής και την υποδομή της επικοινωνίας που παρέχει καλά προσαρμοσμένη σε κατανεμημένα περιβάλλοντα, τα οποία είναι ασύγχρονα από τη φύση τους, όπως για παράδειγμα σε κινητά περιβάλλοντα [36].

2.5.2 Δημοσίευση / Συνδρομή βασισμένη σε Θεματολογία

Τα αρχικά μοντέλα δημοσίευσης / συνδρομής, βασίζονταν στην έννοια του θέματος ή θεματολογίας (topic, subject). Η μεθοδολογία αυτή, χρησιμοποιεί μεθόδους, με σκοπό να χαρακτηρίζει και να κατηγοριοποιεί το περιεχόμενο των γεγονότων. Οι συμμετέχοντες, μπορούν να δημοσιεύσουν και να γραφτούν σε διαφορετικές θεματολογίες, οι οποίες χαρακτηρίζονται από λέξεις-κλειδιά. Τα θέματα, μπορούν να θεωρηθούν σαν υπηρεσίες γεγονότων, οι οποίες έχουν

μοναδικό όνομα και προσφέρουν μέσω διεπαφής λειτουργίες $publish()$ και $subscribe()$. Η θεματολογίες, είναι έννοιες παρόμοιες με την έννοια της ομάδας (group), όπως αυτή ορίζεται στα πλαίσια της *επικοινωνίας σε ομάδες* (group communication) [49]. Συνεπώς, η εγγραφή σε ένα θέμα T από τον συνδρομητή s, μπορεί να θεωρηθεί σαν η πρόσθεση του συνδρομητή s στην ομάδα T, και η δημοσίευση ενός γεγονότος με θέμα T, σαν η μετάδοση ενός γεγονότος στα μέλη της ομάδας T.

2.5.3 Δημοσίευση / Συνδρομή βασισμένη στο Περιεχόμενο

Το μοντέλο της δημοσίευσης / συνδρομής βασισμένο στο περιεχόμενο, αποτελεί στην ουσία μια βελτίωση του μοντέλου βασισμένου σε θεματολογία, το οποίο παρ' όλες τις βελτιώσεις είναι μάλλον στατικό και προσφέρει περιορισμένη εκφραστικότητα. Η παραλλαγή του συστήματος η οποία είναι βασισμένη στο περιεχόμενο [10], παρέχει μια βελτίωση, βασίζοντας της συνδρομές στο πραγματικό περιεχόμενο των γεγονότων. Με άλλα λόγια, τα γεγονότα δεν κατηγοριοποιούνται με βάση κάποια προκαθορισμένα εξωγενή κριτήρια (π.χ. θεματολογία), αλλά σύμφωνα με τις ιδιότητες ίδιων των γεγονότων. Για παράδειγμα, κάποιες ιδιότητες μπορεί να είναι δομές που περιέχουν γεγονότα [10] ή κάποια μετά-δεδομένα [43].

Οι καταναλωτές, εγγράφονται σε κάποια γεγονότα, ορίζοντας φίλτρα, χρησιμοποιώντας μια γλώσσα συνδρομής. Τα φίλτρα, ορίζουν περιορισμούς, συνήθως στη μορφή ζευγαριών ονόματος-τιμής ιδιοτήτων και χρησιμοποιούν βασικούς τελεστές σύγκρισης ($>$,$<$,$\leq$,$\geq$,$=$,$<>$), οι οποίοι καθορίζουν τα συγκεκριμένα γεγονότα, και μπορούν να συνδυαστούν συνήθως με λογικούς τελεστές, για να σχηματίσουν περίπλοκες μορφές συνδρομών. Υπάρχουν διάφοροι τρόποι για τον καθορισμό αυτών των σχημάτων συνδρομής, όπως οι εξής:

- *Συμβολοσειρά (String):* Η πιο συχνή αναπαράσταση των σχημάτων συνδρομής, είναι με συμβολοσειρές. Τα φίλτρα, θα πρέπει να συμβιβάζονται με μια γραμματική, όπως για παράδειγμα της SQL [43], ή της $XPath$ [6, 20, 41].

- *Πρότυπο Αντικείμενο (Template Object):* Εμπνευσμένο από το ταίριασμα βασισμένο σε δυάδες (tuple-based matching), στο JavaSpaces [25] υιοθετείται μια μέθοδος βασισμένη σε προτυποποιημένα αντικείμενα. Κατά τη συνδρομή, μια οντότητα παρέχει ένα αντικείμενο t, το οποίο καθορίζει ότι η οντότητα ενδιαφέρεται για κάθε γεγονός $\bar{t}$, το οποίο συμμορφώνεται με το τύπο του t και όλα τα γνωρίσματά του τερειάζουν με τα γνωρίσματα του t, εκτός από αυτά που περιέχουν το χαρακτήρα μπαλαντέρ ("*"), που στην ουσία δεν έχουν κάποια τιμή (είναι null).

- *Εκτελέσιμος Κώδικας (Executable Code):* Σε αυτή τη μορφή, οι συνδρομητές παρέχουν ένα αντικείμενο, το οποίο είναι φτιαγμένο έτσι ώστε να φιλτράρει τα γεγονότα κατά το τρέξιμο. Η υλοποίηση αυτού του αντικειμένου αφήνεται συνήθως στον προγραμματιστή της εφαρμογής. Μια διαφορετική προσέγγιση, βασίζεται σε βιβλιοθήκες αντικειμένων φιλτραρίσματος. Ο εκτελέσιμος κώδικας δεν χρησιμοποιείται ευρέος, γιατί τα φίλτρα που δημιουργούνται δεν μπορούν να βελτιστοποιηθούν με κάποιο εύκολο τρόπο.

Σε αυτό το σημείο τελειώνει η αναφορά μας στα συστήματα δημοσίευσης / συνδρομής. Αρκεί να αναφέρουμε ότι το σύστημα το οποίο εξετάζουμε και υλοποιούμε σε αυτή τη πτυχιακή, χρησιμοποιεί στη ουσία το μοντέλο δημοσίευσης / συνδρομής βασισμένο στο περιεχόμενο, χρησιμοποιώντας τη γλώσσα XPath ως γλώσσα συνδρομών.

2.6 Συμπεράσματα

Σε αυτό το κεφάλαιο έχουμε παρουσιάσει σε μεγάλο βάθος το θεωρητικό υπόβαθρο και την ερευνητική δουλειά που έχει γίνει σε τομείς που σχετίζονται άμεσα με το σύστημα [41]. Πιο συγκεκριμένα έχουμε περιγράψει τα δίκτυα ομότιμων και τους ΚΠΚ με έμφαση στο σύστημα Pastry. Στη συνέχεια, περιγράψαμε το μοντέλο δεδομένων XML, τη γλώσσα επερωτήσεων XPath καθώς και κάποιες δομές που βοηθούν στον ευρετηριασμό κάποιου XML εγγράφου. Έχουμε αναφερθεί σε στοιχεία από τη Θεωρία Υπολογισμού τα οποία θα χρειαστούμε, καθώς και στη θεωρία των συστημάτων δημοσίευσης / συνδρομής. Έχοντας αναλύσει τη θεωρία στην οποία στηρίζεται το σύστημα [41], είμαστε σε θέση να ξεκινήσουμε τη περιγραφή του.

Κεφάλαιο 3

Περιγραφή Συστήματος

Σε αυτό το κεφάλαιο, θα περιγράψουμε με λεπτομέρεια το σύστημα το οποίο υλοποιήθηκε σε αυτή τη πτυχιακή. Αρχικά, θα περιγραφεί με λεπτομέρεια το σύστημα YFiter, το οποίο είναι ένα πετυχημένο κεντρικοποιημένο σύστημα XML φιλτραρίσματος. Ακολούθως θα περιγραφεί το σύστημα το οποίο υλοποιήθηκε, το οποίο αφορά μια κατανεμημένη προσέγγιση στο ζήτημα του XML φιλτραρίσματος και πιο συγκεκριμένα στη μεθοδολογία που εισάγει το σύστημα YFiter.

Πιο συγκεκριμένα, στην Ενότητα 3.1 παρουσιάζουμε το σύστημα YFilter και τις βασικές συνιστώσες που το αποτελούν. Περιγράφουμε τη διαδικασία κατασκευής του αυτόματου στο σύστημα, τις δομές που το αντιπροσωπεύουν, καθώς και θέματα επεξεργασίας κατηγορημάτων. Επίσης, αναλύουμε τη διαδικασία εκτέλεσης του κατασκευασμένου αυτόματου με βάση η οποία γίνεται με βάση τα στοιχεία που παράγονται κατά τη διερμηνεία ενός εγγράφου XML.

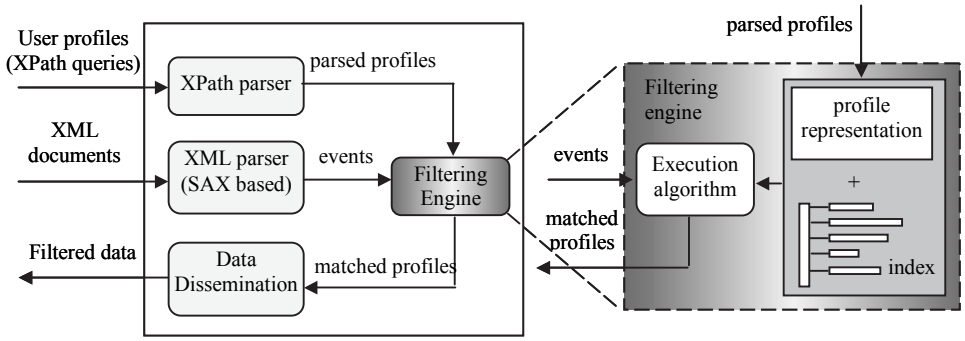
Στην Ενότητα 3.2, παρουσιάζουμε τη κατανεμημένη προσέγγιση, η οποία παρουσιάζεται στο [41]. Περιγράφουμε τον ευρετηριασμό των επερωτήσεων (τη κατασκευή δηλαδή του αυτόματου) και τη κατανομή του πάνω από το κατανεμημένο δίκτυο. Στη συνέχεια, παρουσιάζουμε τις μεθοδολογίες εκτέλεσης του αυτόματου, αναδρομική και επαναληπτική.

3.1 Το σύστημα YFilter

Το σύστημα YFilter [19], είναι το δεύτερο σύστημα μετά το XFilter το οποίο εκμεταλλεύεται μηχανές πεπερασμένων καταστάσεων (Finite State Machines, FSM) και το πρώτο το οποίο χρησιμοποιεί ντετερμινιστικά μη πεπερασμένα αυτόματα για XML φιλτράρισμα. Το σύστημα δέχεται

συνεχής επερωτήσεις[1] (continuous queries) οι οποίες εκφράζονται σε XPath και φτιάχνει σταδιακά ένα μη ντετερμινιστικό πεπερασμένο αυτόματο το οποίο σε κάθε στιγμή αντιπροσωπεύει τις ευρετηριασμένες επερωτήσεις. Κατά τη κατασκευή, κάθε τελική κατάσταση του αυτόματου συσχετίζεται με μία ή περισσότερες επερωτήσεις οι οποίες ικανοποιούνται σε αυτή. Κατά τη δημοσίευση, εκτελεί το αυτόματο πάνω στο XML έγγραφο το οποίο δημοσιεύεται, διατρέχοντας το χρησιμοποιώντας κάποιο συντακτικό αναλυτή βασισμένο σε γεγονότα (Event Based Parser), φιλτράροντας έτσι τη πληροφορία που ενδιαφέρει σύμφωνα με τις επερωτήσεις.

3.1.1 Συνιστώσες του Συστήματος Φιλτραρίσματος



Σχήμα 3.1: Η αρχιτεκτονική του συστήματος YFilter [20]

Σε αυτή την ενότητα θα δούμε τις συνιστώσες του συστήματος XML φιλτραρίσματος YFilter, όπως αυτές παρουσιάζονται στο Σχήμα 3.1.

XPath συντακτικός αναλυτής: Ο συντακτικός αναλυτής XPath, διαβάζει επερωτήσεις γραμμένες σε XPath, τις διερμηνεύει και στέλνει το προφίλ τους στην μηχανή φιλτραρίσματος. Νέα προφίλ μπορούν να προστεθούν στη μηχανή, εν' όσο αυτή δεν επεξεργάζεται κάποιο κείμενο.

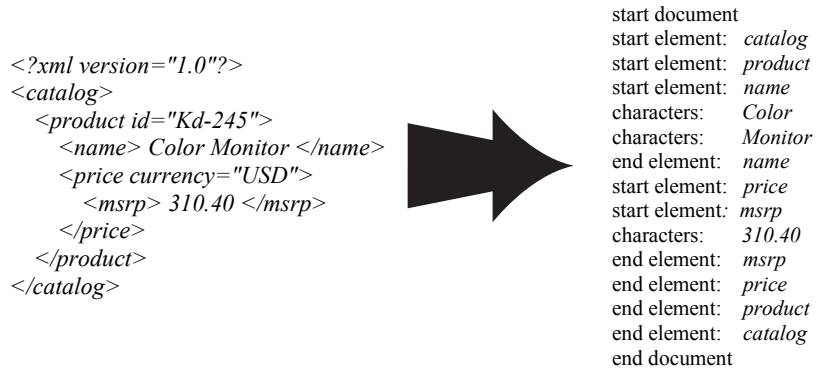
Συντακτικός αναλυτής XML βασισμένος σε γεγονότα: Όταν ένα XML κείμενο φτάνει στο σύστημα, διατρέχεται από ένα συντακτικό αναλυτή για XML γλώσσα, ο οποίος είναι βασισμένος

[1] Ο όρος *συνεχής επερωτήσεις* έχει επεξηγηθεί στο εισαγωγικό κεφάλαιο, αλλά θυμίζουμε ότι αφορά επερωτήσεις οι οποίες διατηρούνται στο σύστημα και ελέγχονται για ικανοποίηση καθώς η κατάσταση του συστήματος αλλάζει δυναμικά, στη περίπτωσή μας καθώς δημοσιεύονται έγγραφα.

σε γεγονότα και βασίζεται στη διεπαφή SAX [Sax Project]. Παράδειγμα εξόδου από το συντακτικό αναλυτή, μπορούμε να δούμε στο Σχήμα 3.2. Ο συντακτικός αναλυτής παράγει μια γραμμική αναπαράσταση των στοιχείων του αρχείου, βασιζόμενο σε γεγονότα όπως το Start of Document και το End of Document, τα οποία σηματοδοτούν την έναρξη και τον τερματισμό του αρχείου, τα Start of Element και End of Element τα οποία αφορούν την έναρξη και το τέλος κάποιου στοιχείου του XML αρχείου, καθώς και πληροφορίες για γνωρίσματα, ονόματα στοιχείων και άλλα. Σημειώνεται, ότι στο σύστημα το φιλτράρισμα αρχίζει κατά τη διάρκεια της παραγωγής των γεγονότων από τον συντακτικό αναλυτή, πολύ πριν ολοκληρωθεί η διαδικασία του φιλτραρίσματος.

Μηχανή Φιλτραρίσματος: Η μηχανή φιλτραρίσματος, λαμβάνει τα προφίλ (δηλαδή τις επερωτήσεις) από το συντακτικό αναλυτή XPath και τα μετατρέπει σε κάποια εσωτερική αναπαράσταση. Εκτός από αυτό, δέχεται τα γεγονότα από το συντακτικό αναλυτή XML και εκτελεί το φιλτράρισμα του κειμένου πάνω στα προφίλ.

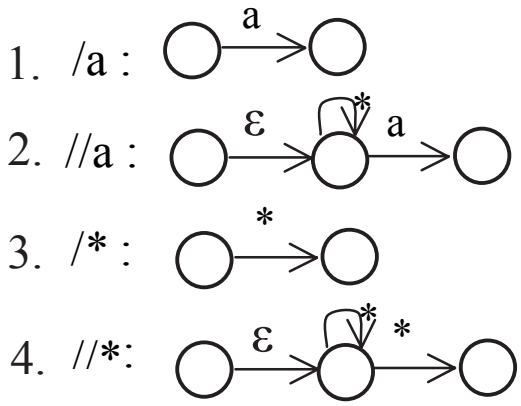
Συνιστώσα Διάχυσης: Όταν κάποιο προφίλ έχει ικανοποιηθεί από κάποιο κείμενο, το κείμενο πρέπει να σταλεί στους αντίστοιχους χρήστες. Η παρούσες υλοποιήσεις του YFilter απλά αποστέλλουν ολόκληρο το έγγραφο στους χρήστες. Διάφοροι μηχανισμοί που φροντίζουν την παράδοση περιγράφονται σε διάφορα άρθρα [3, 4].



Σχήμα 3.2: Παράδειγμα εξόδου του Sax parser [20]

3.1.2 Κατασκευή Αυτόματου από Επερωτήσεις XPath

Η μετατροπή των επερωτήσεων XPath σε πεπερασμένες μηχανές κατάστασης, αναφέρεται αρχικά στο σύστημα XFilter και επαυξήθηκε στο σύστημα YFilter [19], το οποίο χρησιμοποιεί μια αυξητική προσέγγιση στη κατασκευή των αυτομάτων. Η μετατροπή αυτή βασίζεται στη παρατήρηση που γίνεται στο [6], ότι κάθε έκφραση που περιλαμβάνει τους δύο άξονες ("//", "/") μπορεί να γραφεί σαν μια κανονική έκφραση. Μία κανονική έκφραση, παράγει μια κανονική γλώσσα, και σύμφωνα με τη Θεωρία Αυτομάτων, υπάρχει τουλάχιστον ένα αυτόματο το οποίο μπορεί να παράξει τη γλώσσα που εκφράζει η επερώτηση. Σε αυτή την ενότητα, θα δούμε πώς μπορούμε να αναπαραστήσουμε μια XPath επερώτηση σε μια τέτοια μηχανή. Σημειώνεται, ότι αντιμετωπίζουμε περιπτώσεις με γραμμικές επερωτήσεις, δηλαδή επερωτήσεις που δεν έχουν διακλαδώσεις. Σε περίπτωση που έχουμε να αντιμετωπίσουμε επερωτήσεις με διακλαδώσεις, τότε μπορούμε απλά να τις χωρίσουμε σε γραμμικές. Στο [20], παρουσιάζονται οι βασικές περιπτώσεις επερωτήσεων που μπορούμε να συναντήσουμε, καθώς και το πώς μεταφράζονται σε ένα κομμάτι ενός μη ντετερμινιστικού πεπερασμένου αυτόματου.



Σχήμα 3.3: Οι τέσσερις βασικές περιπτώσεις για τη μετατροπή μιας γραμμικής XPath επερώτησης σε ένα τμήμα μη ντετερμινιστικού πεπερασμένου αυτόματου [20]

Στο Σχήμα 3.3, έχουμε τέσσερις περιπτώσεις. Θεωρούμε, ότι το a είναι ένα σύμβολο το οποίο ανήκει στο αλφάβητο το οποίο αποτελείται από όλα τα στοιχεία που είναι ορισμένα σε ένα *ορισμό τύπου εγγράφου*, ενώ παρουσιάζεται και το" $*$", το οποίο είναι ένας χαρακτήρας μπαλαντέρ (wildcard operator), το οποίο αντιστοιχεί σε οποιοδήποτε στοιχείου του εγγράφου. Επίσης, χρησιμοποιούμε την έννοια της κενής μετάβασης, όπως έχει οριστεί στην Ενότητα 2.4.

Στη πρώτη περίπτωση, η επερώτηση αρχίζει με άξονα "/", κάτι που συμβολίζει ότι το επόμενο στοιχείο θα είναι άμεσο παιδί του προηγούμενου κόμβου. Έτσι, αναλογιζόμενοι τους κανόνες λειτουργίας των αυτομάτων, μπορούμε να πούμε ότι, εάν από τη παρούσα κατάσταση δεχτούμε στην είσοδο τη συμβολοσειρά a, τότε μπορούμε να μεταβούμε στην επόμενη κατάσταση, έχοντας ικανοποιήσει το τμήμα αυτό της επερώτησης. Σημειώνεται, ότι μέχρι εδώ το τμήμα αυτόματου που παράχθηκε είναι ντετερμινιστικό.

Στη δεύτερη περίπτωση, έχουμε το ίδιο στοιχείο αλλά ο άξονας της επερώτησης, "//", δηλώνει ότι το στοιχείο a μπορεί να αντιστοιχεί σε οποιοδήποτε απόγονο της κατάστασης που ήδη βρισκόμαστε. Σε αυτό το σημείο θα πρέπει να εισάγουμε το μη ντετερμινισμό στο αυτόματο. Το τμήμα του αυτόματου φτιάχνεται προσθέτοντας μία μετάβαση η οποία καταναλώνει οποιοδήποτε στοιχείο εισόδου ("*") και επιστρέφει στην ίδια κατάσταση (Self Child). Επίσης, το αυτόματο προσθέτει μία μετάβαση με είσοδο το στοιχείο "a", η οποία καταλήγει σε μια κατάσταση στην οποία θεωρούμε ότι έχει ικανοποιηθεί το τμήμα αυτό της επερώτησης. Η διάταξη που έχουμε φτιάξει, αντικατοπτρίζεται στην εξής πρόταση: "Κατανάλωσε οσαδήποτε στοιχεία παραμένοντας στην ίδια κατάσταση, και σε κάποιο a που θα συναντήσεις, πέρασε στην επόμενη". Έχουμε αμέσως συναντήσει το μη ντετερμινισμό, όπως εκφράζεται για το στοιχείο "a", αφού με το στοιχείο στην είσοδο μπορούμε να ακολουθήσουμε δυο μεταβάσεις, τη μετάβαση στην ίδια κατάσταση ("*") και τη μετάβαση στην επόμενη ("a"). Εκτός από τη διάταξη αυτή, η οποία αντιπροσωπεύει πληρέστατα το τμήμα της επερώτησης, έχουμε προσθέσει μια κενή μετάβαση στην αρχή του τμήματος του αυτόματου. Αυτή η κενή μετάβαση δεν αφορά το τμήμα της επερώτησης αυτό καθ' αυτό, αλλά χρησιμεύει στη νοοτροπία που ακολουθεί το YFilter [19], στην αυξητική κατασκευή του αυτόματου που αντιπροσωπεύει τις επερωτήσεις, όπως θα δούμε στις επόμενες παραγράφους. Κάθε τμήμα αυτόματου που παράγουμε θα πρέπει να μπορεί να συνενωθεί με κάποιο άλλο, και εάν δεν τοποθετούσαμε την κενή μετάβαση, η κατάσταση με το βρόγχο ("*") θα εμπόδιζε την ορθή συνένωση αυτού του τμήματος με κάποιο άλλο, όπως για παράδειγμα το τμήμα της πρώτης περίπτωσης. Η κενή μετάβαση είναι το δεύτερο χαρακτηριστικό που ανήκει στο μη ντετερμινισμό.

Στη τρίτη περίπτωση, έχουμε τον χαρακτήρα μπαλαντέρ ("*"), ο οποίος ακολουθεί τον άξονα "/". Αυτό σημαίνει ότι η επερώτηση δέχεται οποιοδήποτε παιδί του στοιχείο (ή αντίστοιχα κατάστασης) στο οποίο βρισκόμαστε. Η αναπαράσταση είναι αντίστοιχη της πρώτης περίπτωσης.

Η τρίτη περίπτωση, αφορά πάλι τον χαρακτήρα μπαλαντέρ, ο οποίος αυτή τη φορά ακολουθεί ένα άξονα "//", ο οποίος αναφέρεται σε οποιοδήποτε απόγονο του στοιχείου που βρισκόμαστε. Η περίπτωση είναι εντελώς αντίστοιχη της δεύτερης που έχουμε εξετάσει.

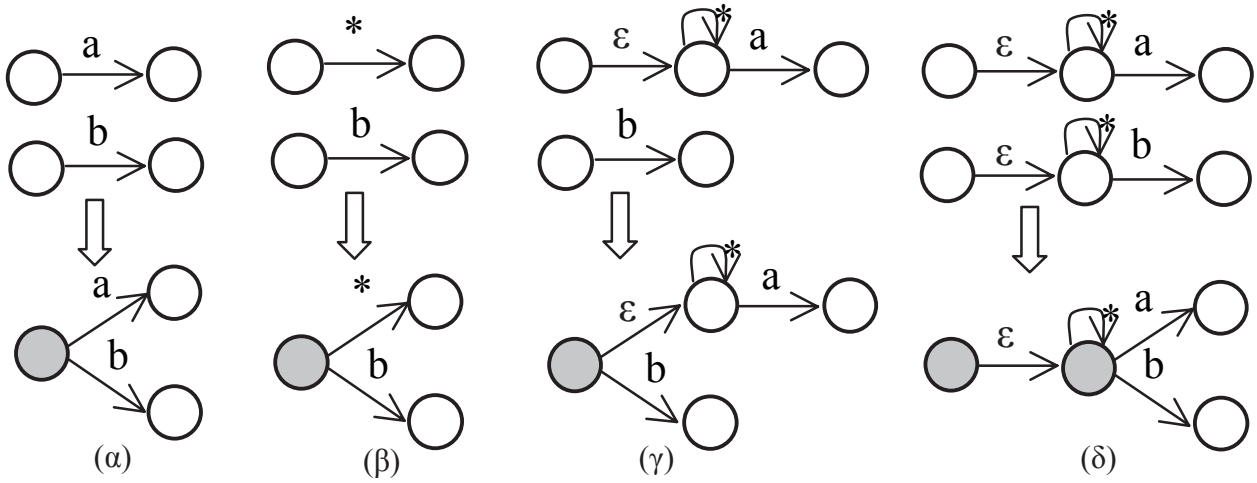
Συνδυασμός τμημάτων αυτομάτων

Όπως έχουμε αναφέρει, η προσέγγιση που ακολουθείται στο [20] είναι μια αυξητική προσέγγιση στη κατασκευή ενός αυτόματου, το οποίο θα συνδυάζει τις επερωτήσεις που ευρετηριάστηκαν στο σύστημα.

Το τελικό αυτόματο για μια έκφραση XPath, δημιουργείται από τη συνένωση των επιμέρους τμημάτων της, θέτοντας σαν τελική κατάσταση του τελευταίου τμήματος της επερώτησης. Αυτό, πηγάζει από το γεγονός ότι επεξεργαζόμαστε γραμμικές επερωτήσεις. Η αυξητική προσέγγιση του YFilter [19], συνενώνει όλες τις επερωτήσεις σε ένα αυτόματο, με τη διαδικασία που θα περιγράψουμε στη συνέχεια.

Για την εισαγωγή ενός νέου τμήματος στο ήδη υπάρχον (που μπορεί να είναι κενό) αυτόματο, θεωρούμε αρχικά ότι όλες οι επερωτήσεις μοιράζονται μια κοινή κατάσταση, τη κατάσταση εκκίνησης (αρχική κατάσταση, start state) του αυτόματου. Αρχίζοντας από αυτή τη κατάσταση, αρχίζουμε να διατρέχουμε το αυτόματο. Δηλαδή, συγκρίνουμε το αυτόματο της έκφρασης που θα εισάγουμε στο ήδη υπάρχον, αρχίζοντας από τη κατάσταση εκκίνησης και ακολουθώντας όποια μετάβαση υπάρχει στο αυτόματο που εισάγεται ταυτόχρονα και στο υπάρχον, εάν βέβαια αυτή υπάρχει. Συνεχίζουμε τη διαδικασία, μέχρι να συμβεί ένα από τα εξής: Είτε να φτάσουμε σε κατάσταση αποδοχής του αυτόματου το οποίοι εισάγουμε είτε να φτάσουμε σε κάποια κατάσταση του αρχικού αυτόματου όπου δεν υπάρχει αντίστοιχη μετάβαση με αυτή που συναντάμε στο αυτόματο που εισάγουμε. Στη πρώτη περίπτωση, σημαίνει ότι δεν χρειάζεται να προσθέσουμε καμία κατάσταση στο υπάρχον αυτόματο. Το αυτόματο που προσπαθήσαμε να εισάγουμε, αποτελεί υποσύνολο του αυτόματου που ήδη υπάρχει. Αρκεί να θέσουμε τη κατάσταση στην οποία φτάσαμε στο αρχικό αυτόματο, σαν κατάσταση αποδοχής (τελική κατάσταση) και να σημειώσουμε ότι αντιστοιχεί στην επερώτηση της οποίας το αυτόματο συνενώθηκε με το αρχικό. Στη δεύτερη περίπτωση, θα πρέπει να δημιουργήσουμε την αντίστοιχη μετάβαση στο αρχικό, και να συνεχίσουμε με το υπόλοιπο αυτόματο.

Στο Σχήμα 3.4, βλέπουμε κάποια παραδείγματα συνένωσης τμημάτων αυτομάτων για τμήματα επερωτήσεων και πώς αυτά συνδυάζονται σε ένα τελικό αυτόματο, σύμφωνα με την αυξητική



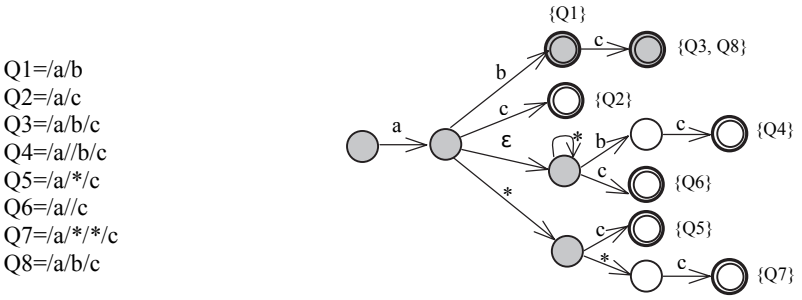
Σχήμα 3.4: Παραδείγματα συνένωσης τμημάτων αυτομάτων επερωτήσεων XPath [20]

προσέγγιση που ακολουθείται στο YFilter [19].

Στην πρώτη περίπτωση, βλέπουμε την συνένωση δυο απλών τμημάτων, κάθε ένα από τα οποία ανήκει στην πρώτη περίπτωση όπως αυτές αναφέρονται στο Σχήμα 3.3. Θεωρώντας ότι η αρχική κατάσταση είναι κενή, δημιουργούμε μία μετάβαση για το κάθε σύμβολο, ξεχωριστή, για να διατηρήσουμε τη διαφορετική σημασιολογία του κάθε συμβόλου. Στη δεύτερη περίπτωση του Σχήματος 3.4 το αποτέλεσμα είναι το ίδιο με τη προηγούμενη. Πρέπει να σημειώσουμε ότι ο χαρακτήρας "*" δεν μπορεί να συνενωθεί με τον "b", ακόμα και αν το "b" ανήκει στα σύμβολα τα οποία αποδέχεται το "*", και αυτό είναι σημαντικό στη διατήρηση της σημασιολογίας των επερωτήσεων. Σε διαφορετική περίπτωση, το αυτόματο θα αποδεχόταν περισσότερες επερωτήσεις από όσες θα έπρεπε.

Στη τρίτη περίπτωση, βλέπουμε τη συνένωση ενός βήματος που περιέχει τον άξονα απογόνου ("//") και ενός βήματος που περιέχει τον άξονα παιδιού ("/"). Εδώ, μπορούμε να παρατηρήσουμε γιατί αναφέραμε προηγούμενος ότι στη περίπτωση του άξονα απογόνου, πρέπει να εισάγουμε μία κενή μετάβαση στην αρχή του τμήματος. Εάν δεν είχαμε τη κενή μετάβαση εδώ, δεν θα μπορούσαμε να συνενώσουμε την αρχική κατάσταση του "//a" με την αρχική κατάσταση του "/b", γιατί στο τέλος αντί την έκφραση "/b" θα είχαμε την έκφραση "//b". Στη τέταρτη περίπτωση, έχουμε τη συνένωση δυο εκφράσεων οι οποίες χρησιμοποιούν για άξονα το "//". Η κενή μετάβαση συνενώνεται στο τελικό αυτόματο ενώ διακλαδώνεται στη συνέχεια.

Στο Σχήμα 3.5, παρουσιάζονται (α) οκτώ επερωτήσεις, ενώ στο (β) μπορούμε να δούμε τη συνένωσή τους στο τελικό αυτόματο. Με διπλό κύκλο σημειώνονται οι τελικές καταστάσεις, ενώ σε αγκύλες καταγράφονται τα αναγνωριστικά των επερωτήσεων οι οποίες ικανοποιούνται στις αντίστοιχες τελικές καταστάσεις.



Σχήμα 3.5: Παραδείγματα συνένωσης αυτομάτων επερωτήσεων σε ένα τελικό αυτόματο [20]

Επεξεργασία Κατηγορημάτων

Όπως έχουμε αναφέρει, η γλώσσα XPath προσφέρει την δυνατότητα χρήσης κατηγορημάτων για το φιλτράρισμα των αποτελεσμάτων των επερωτήσεων. Στο [32], σχολιάζεται η προσθήκη των κατηγορημάτων στο αυτόματο, με τη μορφή μεταβάσεων μεταξύ καταστάσεων, με σύμβολα στις μεταβάσεις τα κατηγορήματα. Η μεθοδολογία όμως αυτή, θα προκαλούσε την τρομερή αύξηση του αριθμού των καταστάσεων αλλά και θα κατέστρεφε τη δυνατότητα της κοινής χρήσης κομματιών του αυτόματου από πολλές εκφράσεις, το κύριο προτέρημα της αναπαράστασης με μη ντετερμινιστικά πεπερασμένα αυτόματα.

Γενικότερα, η επεξεργασία κατηγορημάτων σε συστήματα επεξεργασίας επερωτήσεων έχει γενικότερα μεγάλη επίπτωση στην επίδοση του συστήματος. Γι' αυτό και τα περισσότερα συστήματα φροντίζουν η επεξεργασία να γίνεται όσο πιο γρήγορα κατά την εκτέλεση. Σε αυτή τη νοοτροπία βασίζεται και η μέθοδος "Inline", η οποία παρουσιάζεται στο [32]. Η μέθοδος αυτή, επεξεργάζεται τα κατηγορήματα μόλις η εκτέλεση φτάνει στη κατάλληλη κατάσταση. Παρόμοιες προσεγγίσεις έχουν αναφερθεί [47, 12] για την υποστήριξη της αποτίμησης κατηγορημάτων σε αυτόματα ή σαν επεκτάσεις αλγορίθμων. Μια άλλη προσέγγιση, αφορά τον αλγόριθμο Αναβολής Επιλογής (Selection Postponed, SP), ο οποίος περιμένει ο αλγόριθμος να φτάσει σε μια κατάσταση τερματισμού, και μόνο τότε ελέγχει την ικανοποίηση των κατηγορημάτων που έχει συναντήσει.

Υλοποίηση δομής αυτόματου στο YFilter

Κάθε κατάσταση του αυτόματου στο YFilter, αντιπροσωπεύεται από μια δομή, η οποία περιέχει το αναγνωριστικό της κατάστασης, πληροφορίες για τη κατάσταση (για παράδειγμα εάν είναι κατάσταση αποδοχής κάποιας επερώτησης, ή αν έχει μετάβαση (self child) που επιστρέφει στον εαυτό της). Επίσης, αποθηκεύει τις μεταβάσεις από αυτή τη κατάσταση (εκτός εάν είναι self-child οπότε η πληροφορία αυτή αποθηκεύεται διαφορετικά) και τελικά, εάν η κατάσταση είναι κατάσταση αποδοχής, τις επερωτήσεις οι οποίες ικανοποιούνται σε αυτή. Οι μεταβάσεις στη δομή κάποιας κατάστασης, είναι αποθηκευμένες σε κάποιο πίνακα κατακερματισμού, στη μορφή ζευγαριών [σύμβολο εισόδου, αναγνωριστικό κατάστασης στόχου], όπου το σύμβολο εισόδου μπορεί να είναι οποιοδήποτε στοιχείο (που αντιστοιχεί στην ετικέτα μιας μετάβασης), ενώ το αναγνωριστικό κατάστασης αφορά τη κατάσταση στην οποία μεταβαίνουμε ακολουθώντας αυτή τη μετάβαση. Τονίζεται η ξεχωριστή μεταχείριση της περίπτωσης όπου συναντούμε τον άξονα "// ". Όπως έχει αναφερθεί, στη περίπτωση αυτή προσθέτουμε μία κενή μετάβαση και στην επόμενη κατάσταση ένα βρόγχο, δηλαδή μια μετάβαση στην ίδια κατάσταση με οποιοδήποτε σύμβολο. Ενώ η κενή μετάβαση σημειώνεται κανονικά στο πίνακα μεταβάσεων, ο βρόγχος σημειώνεται διαφορετικά. Αυτό, αφορά την εκτέλεση του αυτόματου με τη μεθοδολογία του YFilter.

3.1.3 Εκτέλεση Αυτόματου

Η εκτέλεση του αυτόματου στο YFilter, ακολουθεί την οδηγούμενη από γεγονότα εκτέλεση. Κατά την διερμηνεία του XML εγγράφου, τα γεγονότα λαμβάνονται από τη μηχανή και εκτελείτε το αυτόματο. Κατά τη λήψη ενός γεγονότος End of Element, το οποίο σηματοδοτεί το τέλος κάποιου στοιχείου του εγγράφου, πρέπει να υπάρχει κάποιος μηχανισμός για την οπισθοδρόμηση στο προηγούμενο σύνολο καταστάσεων του αυτόματου, στις οποίες βρισκόταν πριν το διάβασμα του στοιχείου, έτσι ώστε να συνεχίσει την εκτέλεση, πιθανός με κάποιο αδερφό στοιχείο (sibling) στη δενδρική αναπαράσταση του εγγράφου. Γι᾽ αυτό, στο YFilter χρησιμοποιείται ένας μηχανισμός στοίβας (run-time stack).

Χειρισμός γεγονότος έναρξης εγγράφου: Κατά την άφιξη ενός γεγονότος έναρξης εγγράφου (Start Document), η αρχική κατάσταση του αυτόματου τοποθετείτε στη στοίβα runtime, κάτι που σηματοδοτεί την έναρξη της εκτέλεσης του αυτόματου. Η αρχική κατάσταση τώρα, είναι η ενεργή κατάσταση της εκτέλεσης.

Χειρισμός γεγονότος έναρξης εγγράφου: Η άφιξη ενός τέτοιου γεγονότος αντιστοιχεί στην έναρξη ενός στοιχείου του εγγράφου. Η εκτέλεση του αυτόματου θα πρέπει να συμβαδίζει με αυτή την εξέλιξη, επεκτείνοντας όλες τις μεταβάσεις από τις ενεργές καταστάσεις, εκτελώντας τα επόμενα βήματα για κάθε ενεργή κατάσταση:

- Αρχικά, το όνομα του στοιχείου του οποίου η έναρξη σηματοδοτείται από το γεγονός, αναζητείται στις μεταβάσεις της κατάστασης. Κάθε κατάσταση στόχος που επιστρέφεται από αυτή την αναζήτηση, προστίθεται σε ένα σύνολο καταστάσεων, το σύνολο των καταστάσεων στόχων (target states).

- Στη συνέχεια, αναζητάμε το σύμβολο "*" στο πίνακα μεταβάσεων της κατάστασης. Εάν υπάρχει, τότε ανακτούμε τη κατάσταση στόχο και τη προσθέτουμε και αυτή στο σύνολο καταστάσεων στόχων. Αυτό, γίνεται επειδή ο χαρακτήρας "*" μπορεί να ταιριάξει με οποιοδήποτε στοιχείο.

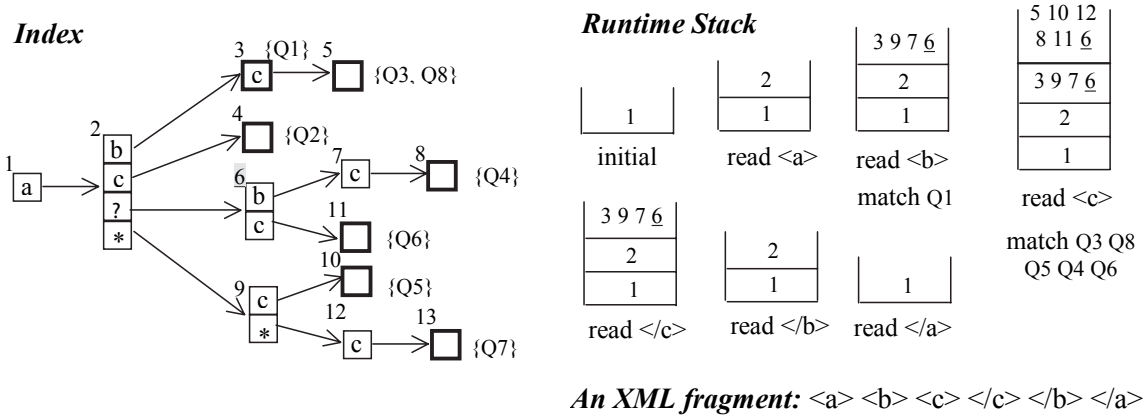
- Αφού προσθέσουμε τις καταστάσεις στόχους από μεταβάσεις, στη συνέχεια θα ελέγξουμε το τύπο της κατάστασης. Όπως προαναφέρθηκε, η κατάσταση μπορεί να έχει βρόγχο (να είναι κατάσταση self-child), κάτι που δεν απεικονίζεται στις μεταβάσεις του. Εάν είναι, τότε προσθέτουμε την παρούσα κατάσταση στο σύνολο καταστάσεων στόχου.

- Τέλος, εάν συναντήσουμε κάποια κενή μετάβαση (εάν δηλαδή υπάρχει κάποια καταχωρημένη στις μεταβάσεις τις κατάστασης), τότε ανακτούμε τη κατάσταση στόχου, την οποία επεξεργαζόμαστε αναδρομικά σύμφωνα με τα τρία ανώτερα βήματα. Σημειώνεται ότι στην ουσία δεν χρειάζεται να ελέγξουμε για αυτό το βήμα στον αναδρομικό έλεγχο, γιατί πολύ απλά η επόμενη κατάσταση αποκλείεται να έχει κάποια κενή μετάβαση, αφού αυτό θα σήμαινε ότι έχουμε δύο συνεχόμενους άξονες απογόνου, κάτι που δεν είναι δυνατό. Με το πέρας αυτής της διαδικασίας, οι καταστάσεις που μαζεύτηκαν σαν καταστάσεις στόχου, τοποθετούνται (push) στη κορυφή της στοίβας runtime, έτσι ώστε να γίνουν οι ¨ενεργές¨ καταστάσεις για το επόμενο τρέξιμο.

Χειρισμός γεγονότος τέλους στοιχείου Το γεγονός αυτό, σηματοδοτεί ότι συναντήσαμε το τέλος κάποιου στοιχείου στο έγγραφό. Αυτό σημαίνει ότι θα πρέπει να εκτελέσουμε οπισθοδρόμηση. Αφαιρούμε τις ενεργές καταστάσεις απο τη κορυφή της στοίβας, και το προηγούμενο σύνολο ενεργών καταστάσεων γίνονται οι ενεργές καταστάσεις. Ουσιαστικά, η διαδικασία αυτή μας φέρνει στο προηγούμενο σύνολο καταστάσεων στο αυτόματο, καθώς ανεβαίνουμε και ένα επίπεδο στη δενδρική αναπαράσταση του εγγράφου XML.

Σημειώνεται μια βασική διάφορα της γνωστής εκτέλεσης ενός μη ντετερμινιστικού πεπερασμένου αυτόματου από την εκτέλεση στο αυτόματο ευρετηριασμένων επερωτήσεων: Στο παραδοσιακό αυτόματο, ψάχνουμε να βρούμε τουλάχιστον ένα μονοπάτι το οποίο μας φέρνει σε τελική κατάσταση. Η εκτέλεση μπορεί να σταματήσει τότε, εφ᾽ όσον έχει καταναλωθεί η είσοδος, και η συμβολοσειρά ανήκει στη γλώσσα του αυτόματου. Στη περίπτωση που εξετάζουμε, η εκτέλεση δεν σταματάει μέχρι να εξαντληθεί το έγγραφο (έτσι ώστε να ανακτήσουμε όλες τις επερωτήσεις που ικανοποιούνται). Επίσης, δεν περιμένουμε να εξαντληθεί η είσοδος (το έγγραφο) για να ελέγξουμε εάν έχουμε φτάσει σε τελική κατάσταση, κάτι που πρέπει να είναι ήδη προφανές από τη περιγραφή.

Στο Σχήμα 3.6, παρουσιάζεται ένα στιγμιότυπο εκτέλεσης του αυτόματου στο YFilter. Στα αριστερά, παρουσιάζεται το ευρετήριο για το αυτόματο το οποίο παρουσιάστηκε στο Σχήμα 3.5. Ο αριθμός στη κορυφή αριστερά κάθε πίνακα, αφορά το αναγνωριστικό της κάθε κατάστασης στην οποία αντιστοιχεί ο πίνακας κατακερματισμού (που όπως φαίνεται είναι η δομή που αποθηκεύει τις μεταβάσεις). Στα δεξιά, παρουσιάζεται η εξέλιξη των περιεχομένων της στοίβας runtime, καθώς διερμηνεύεται το έγγραφο XML του οποίου ένα μέρος παρουσιάζεται. Κάθε υπογραμμισμένη κατάσταση είναι κατάσταση που έχει το βρόχο self-child.

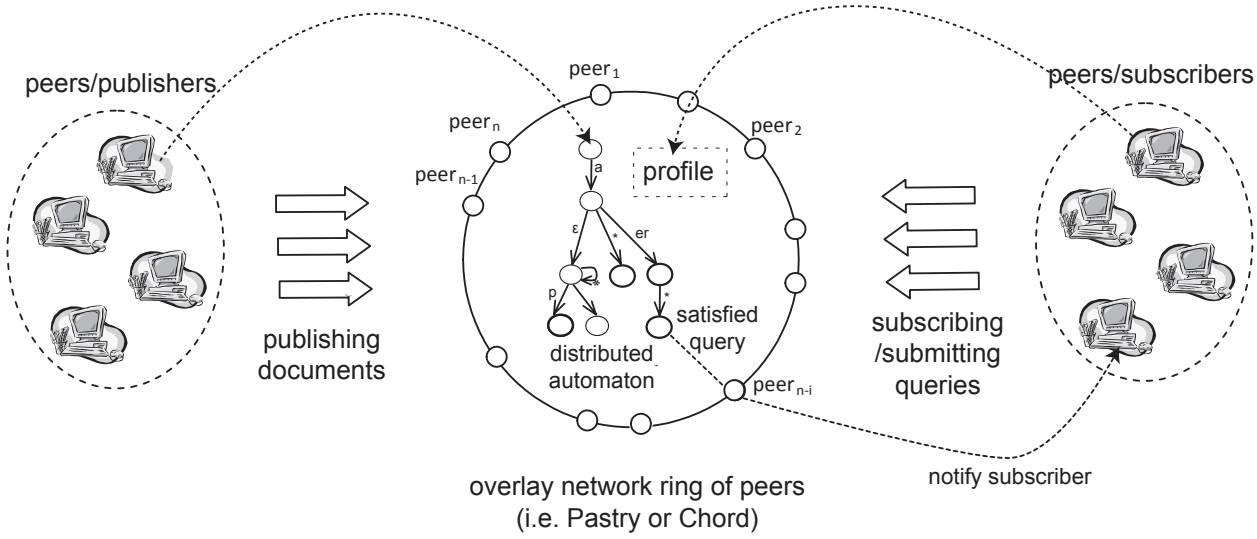


Σχήμα 3.6: Παραδείγματα εκτέλεσης αυτόματου [20]

3.2 Η κατανεμημένη προσέγγιση

Στη συνέχεια, θα περιγράψουμε τη προσέγγιση της οποίας η υλοποίηση είναι και το θέμα της πτυχιακής [41]. Η προσέγγιση αυτή, εκμεταλλεύεται τις δυνατότητες που προσφέρει ένας

κατανεμημένος πίνακας κατακερματισμού, για να επιτύχει το XML φιλτράρισμα. Το σύστημα δεν απαιτεί κεντρικοποιημένο έλεγχο και επιπλέον οι πίνακες δρομολόγησης κάθε κόμβου διατηρούνται σε σταθερό μέγεθος. Παρουσιάζονται αλγόριθμοι, οι οποίοι στην ουσία κατασκευάζουν το αυτόματο που αντιπροσωπεύει τις επερωτήσεις, κατανέμοντας καταστάσεις του αυτόματου στο δίκτυο ομότιμων. Στη συνέχεια παρουσιάζονται δύο μέθοδοι εκτέλεσης του αυτόματου πάνω από το δίκτυο· μία επαναληπτική μέθοδος, η οποία αφορά τον κόμβο όπου γίνεται η δημοσίευση και ο οποίος επαναληπτικά ανακαλεί καταστάσεις από τους υπόλοιπους κόμβους καθώς εκτελεί το αυτόματο, και μία αναδρομική μέθοδος, που εκμεταλλευόμενη την παράλληλη εκτέλεση του αυτόματου (λόγω του μη ντετερμινισμού) εκτελεί παράλληλα διάφορα μονοπάτια του αυτόματου πάνω από το δίκτυο.



Σχήμα 3.7: Σχηματική αναπαράσταση συστήματος υλοποίησης

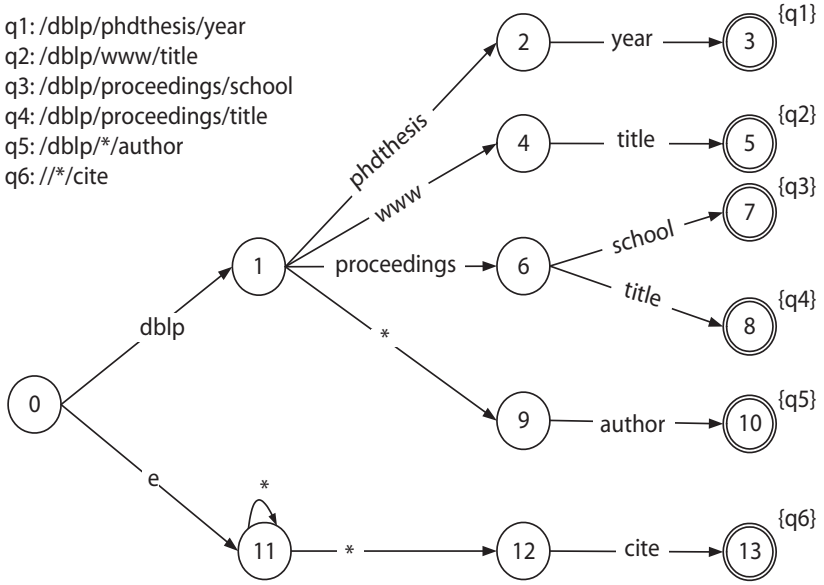
Σημειώνεται ότι η περιγραφή στην ενότητα αυτή θα γίνει με βάση το σύστημα Chord, όπως και στο [41], ενώ στο κεφάλαιο που αφορά την υλοποίηση αυτή καθ' αυτή, η περιγραφή θα αφορά την υλοποίηση στο σύστημα FreePastry. Οι διαφορές στη περιγραφή αφορούν κυρίως κάποιες εσωτερικές λεπτομέρειες των συστημάτων ενώ γενικότερα η μετάβαση από το Chord στο Pastry είναι αρκετά άμεση.

Κατανομή αυτόματου και τοπικές δομές

Στη προσέγγιση του [41], οι καταστάσεις του αυτόματου κατανέμονται πάνω από το δίκτυο, αναθέτοντας σε κάθε κόμβο στο δίκτυο τη κατάσταση q_i, μαζί με οποιαδήποτε άλλη κατάσταση περιλαμβάνεται στην επεκτεταμένη συνάρτηση μετάβασης $\hat{\delta}(q_i, w)$, όπου το $|w| = l$, δηλαδή το w είναι μία συμβολοσειρά μήκους l η οποία περιλαμβάνεται στο $\Sigma \cup \epsilon$. Το l είναι μια παράμετρος του συστήματος, η οποία αφορά το μέγεθος του κομματιού του αυτόματου για το οποίο είναι υπεύθυνος κάθε κόμβος. Για παράδειγμα, εάν $l = 0$, τότε κάθε κατάσταση ανατίθεται μόνο σε ένα κόμβο, με την εξαίρεση των κενών μεταβάσεων, των οποίων η ύπαρξη συνιστά την αποθήκευση της κατάστασης στόχου στην οποία μεταβαίνουμε μέσω αυτής. Για μεγαλύτερες τιμές του l, κάποιες καταστάσεις αποθηκεύονται σε περισσότερους από ένα κόμβους. Αυτό έχει ως αποτέλεσμα την αποθήκευση επικαλυπτόμενων τμημάτων του αυτόματου στους διάφορους κόμβους, των οποίων το μέγεθος καθορίζεται από τη παράμετρο l.

Για τη κατανομή των καταστάσεων σε κόμβους, χρειάζεται κάποιου είδους αναγνωριστικό, το οποίο θα ανατίθεται σε κάθε κατάσταση και θα την χαρακτηρίζει μοναδικά. Ο *υπεύθυνος κόμβος* (responsible peer) για μια κατάσταση η οποία έχει κλειδί key, βρίσκεται ανακτώντας το αναγνωριστικό του κλειδιού, εφαρμόζοντας πάνω του κάποια συνάρτηση κατακερματισμού, $Hash(key)$. Στο Chord, ο υπεύθυνος κόμβος θα είναι ο διάδοχος κόμβος (σύμφωνα με τη λειτουργία του Chord που έχει περιγραφεί), ενώ στο Pastry θα είναι ο αριθμητικά πιο κοντινός κόμβος. Το κλειδί μιας κατάστασης ορίζεται ως η συνένωση των μεταβάσεων από την αρχική κατάσταση (start state), μέχρι τη μετάβαση η οποία οδηγεί στην κατάσταση της οποίας το κλειδί ψάχνουμε, δηλαδή αφορά ένα μονοπάτι του αυτόματου. Για παράδειγμα, στο Σχήμα 3.8, το κλειδί της κατάστασης 2 είναι η συμβολοσειρά *"start"* + *"dblp"* + *"phdthesis"*, όπου ο χαρακτήρας "+" αντιστοιχεί στο χαρακτήρα συνένωσης συμβολοσειράς. Το "start" αναφέρεται στην αρχική κατάσταση, ενώ το σύμβολο "$" χρησιμοποιείται για να αναφερθούμε στη κενή συμβολοσειρά. Έτσι, η κατάσταση 11 έχει κλειδί "start" + "$".

Κάθε κόμβος p, κρατάει κάποιες πληροφορίες στις τοπικές του δομές. Κρατάει μία λίστα καταστάσεων $p.states$, στην οποία ανήκουν οι καταστάσεις για τις οποίες είναι υπεύθυνος, μαζί με πληροφορίες για τη κάθε κατάσταση. Έστω η κατάσταση s, οι πληροφορίες που αποθηκεύονται για αυτή, αφορούν το αναγνωριστικό κλειδί της, τις μεταβάσεις από αυτές, αν η κατάσταση περιέχει βρόγχο στον εαυτό της, και τέλος, αν η κατάσταση είναι τελική, ποιες επερωτήσεις γίνονται αποδεκτές σε αυτή ($s.queries$).



Σχήμα 3.8: Παραδείγματα συνένωσης επερωτήσεων σε αυτόματο [41]

Στο Σχήμα 3.9, παρουσιάζεται μια κατανομή του αυτόματου που παρουσιάζεται στο Σχήμα 3.8 πάνω από ένα δίκτυο 11 κόμβων, όπου κάθε κατάσταση που παρουσιάζεται, εμφανίζεται επίσης στο δίκτυο Chord, με τη παράμετρο $l = 1$. Παρατηρούμε, ότι η κατάσταση 10 περιέχεται στις καταστάσεις του τρίτου κόμβου[2]: $p_3.states$ = {0,1,9,10}, αφού η κενή μετάβαση από την κατάσταση 0 στην 9 δεν επιβαρύνει το l.

Στο Σχήμα 3.2, εμφανίζονται οι επερωτήσεις των οποίων το αυτόματο παρουσιάζεται στο Σχήμα 3.11. Στο Πίνακα 3.1, παρουσιάζουμε τις καταστάσεις οι οποίες κατανέμονται σε κάθε κόμβο, για διαφορετικές τιμές της παραμέτρου l.

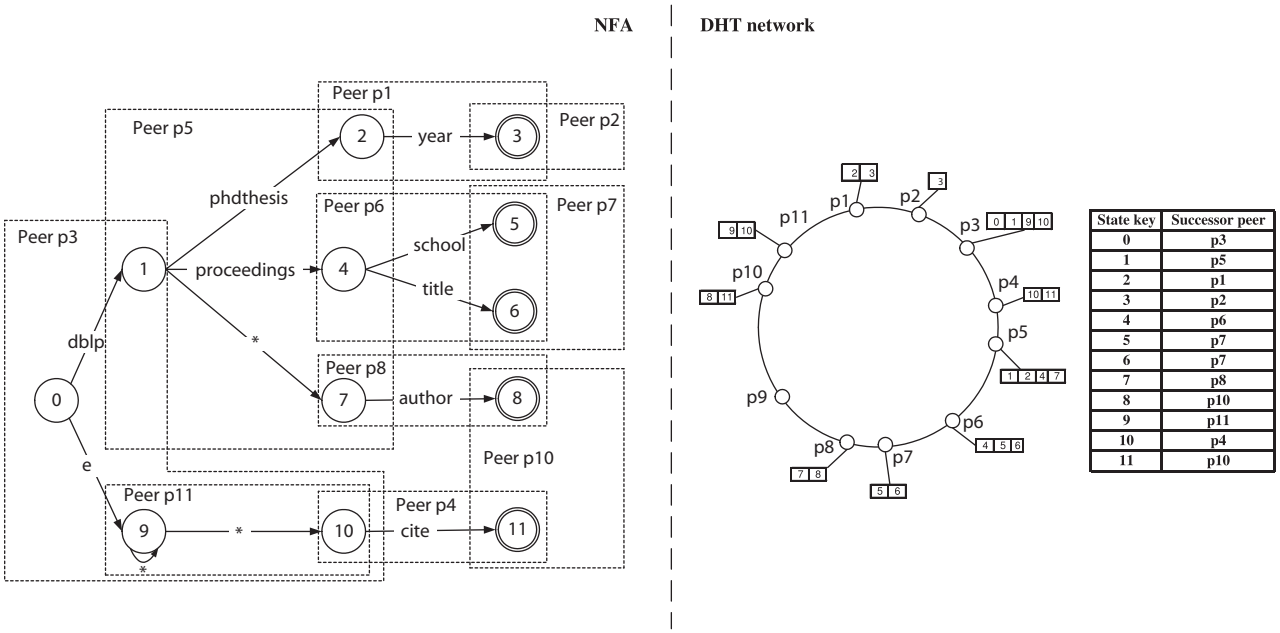
3.2.1 Κατασκευή Αυτόματου

Στη κατανεμημένη προσέγγιση, η κατασκευή αυτόματου γίνεται αυξητικά, καθώς οι επερωτήσεις υποβάλλονται. Θα χρησιμοποιούμε τον όρο *κατανεμημένο αυτόματο* για να αναφερόμαστε στο μη ντετερμινιστικό πεπερασμένο αυτόματο το οποίο κατανέμουμε πάνω από το δίκτυο. Επίσης σημειώνεται ότι σε αυτή την ενότητα, ο συμβολισμός $p.Proc()$ σημαίνει ότι ο κόμβος p λαμβάνει κάποιο μήνυμα και εκτελεί την διαδικασία $Proc()$.

[2]Στις εικόνες χρησιμοποιούνται αριθμοί αντί η συνένωση των συμβολοσειρών όπως ορίστηκε παραπάνω, για συντομία.

Πίνακας 3.1: Η παράμετρος l και πώς επηρεάζει τη κατανομή καταστάσεων στο αυτόματο που παρουσιάζεται στο Σχήμα 3.11

Κόμβος	$l = 0$	$l = 1$	$l = 2$
p_0	$\{0, 2\}$	$\{0, 2, 3, 12, 1\}$	$\{0, 3, 1, 14, 13, 2, 12, 25, 31\}$
p_1	$\{1, 10\}$	$\{1, 4, 5, 6, 7, 8, 9, 10, 11, 15, 22, 23\}$	$\{1, 4, 5, 6, 7, 8, 9, 10, 22, 23, 11, 24, 22, 23, 21, 20, 19, 18, 17, 16, 15, 28\}$
p_2	$\{2\}$	$\{2, 12, 25\}$	$\{2, 12, 25, 31\}$
p_3	$\{3\}$	$\{3, 13, 14\}$	$\{3, 13, 14, 26, 26\}$
p_4	$\{4, 15\}$	$\{4, 15, 28\}$	$\{4, 15, 28\}$
p_5	$\{5\}$	$\{5, 16\}$	$\{5, 16\}$
p_6	$\{6\}$	$\{6, 17, 18\}$	$\{6, 17, 18\}$
p_7	$\{7\}$	$\{7, 19\}$	$\{7, 19, 29\}$
p_8	$\{8\}$	$\{8, 20\}$	$\{8, 20\}$
p_9	$\{9\}$	$\{9, 21\}$	$\{9, 21, 30\}$
p_{10}	$\{10\}$	$\{10, 22, 23\}$	$\{10, 22, 23\}$
p_{11}	$\{11\}$	$\{11, 24\}$	$\{11, 24\}$
p_{12}	$\{12, 25\}$	$\{12, 25, 31\}$	$\{12, 25, 31\}$
p_{13}	$\{13\}$	$\{13, 26\}$	$\{13, 26\}$
p_{14}	$\{14\}$	$\{14, 27\}$	$\{14, 27\}$
p_{15}	$\{15\}$	$\{15, 28\}$	$\{15, 28\}$
p_{16}	$\{16\}$	$\{16\}$	$\{16\}$
p_{17}	$\{17\}$	$\{17\}$	$\{17\}$
p_{18}	$\{18\}$	$\{18\}$	$\{18\}$
p_{19}	$\{19\}$	$\{19, 29\}$	$\{19, 29, 32\}$
p_{20}	$\{20\}$	$\{20\}$	$\{20\}$
p_{21}	$\{21\}$	$\{21, 30\}$	$\{21, 30\}$
p_{22}	$\{22\}$	$\{22\}$	$\{22\}$
p_{23}	$\{23\}$	$\{23\}$	$\{23\}$
p_{24}	$\{24\}$	$\{24\}$	$\{24\}$
p_{25}	$\{25\}$	$\{25, 31\}$	$\{25, 31\}$
p_{26}	$\{26\}$	$\{26\}$	$\{26\}$
p_{27}	$\{27\}$	$\{27\}$	$\{27\}$
p_{28}	$\{28\}$	$\{28\}$	$\{28\}$
p_{29}	$\{29\}$	$\{29, 32\}$	$\{29, 32, 33\}$
p_{30}	$\{30\}$	$\{30\}$	$\{30\}$
p_{31}	$\{31\}$	$\{31\}$	$\{31\}$
p_{32}	$\{32\}$	$\{32, 33\}$	$\{32, 33, 34\}$
p_{33}	$\{33\}$	$\{33, 34\}$	$\{33, 34\}$
p_{34}	$\{34\}$	$\{34\}$	$\{34\}$



Σχήμα 3.9: Παραδείγματα κατανομής καταστάσεων πάνω από ένα δίκτυο Chord [41]

Έστω ο κόμβος s (*subscriber*), ο οποίος υποβάλλει την επερώτηση q στέλνοντας ένα μήνυμα $IndexQuery(q,\ d,\ s)$, όπου q η επερώτηση, d το βάθος του αυτόματου της επερώτησης στο οποίο έχει φτάσει η εκτέλεση και s ο κόμβος συνδρομητής. Η κατασκευή αρχίζει από τη κοινή αρχική κατάσταση, *start state*, η οποία αντιστοιχεί σε βάθος 0 ($d = 0$), όπου ο κόμβος είναι ο υπεύθυνος κόμβος για τη κατάσταση εκκίνησης (στο Chord), ο $succ("start")$ και στο Pastry ο κοντινότερος αριθμητικά κόμβος στο $Hash("start")$.

Κάθε κόμβος p_1 ο οποίος λαμβάνει ένα μήνυμα $IndexQuery$, ανακτά τη κατάσταση st η οποία βρίσκεται σε βάθος d στο αυτόματο της επερώτησης q και προσπαθεί να ανακτήσει τη κατάσταση με κλειδί $st.Key$ από τη τοπική λίστα $p_1.states$. Εάν η κατάσταση είναι τελική κατάσταση, δηλαδή εάν έχουμε φτάσει στο τελικό βάθος του αυτόματου της επερώτησης, τότε προσθέτουμε το αναγνωριστικό της επερώτησης στη λίστα ικανοποιημένων επερωτήσεων στη κατάσταση του αυτόματου ($st.queries$). Διαφορετικά, έστω t η ετικέτα της μετάβασης από τη κατάσταση st σε μία κατάσταση st', όπως αυτή παρουσιάζεται στο αυτόματο της επερώτησης q. Εάν μία τέτοια μετάβαση δεν εμφανίζεται στη κατάσταση st, τότε ο p_1 προσθέτει μία καινούργια μετάβαση στη κατάσταση st με ετικέτα t και με κατάσταση στόχο την st'. Τελικά, ο p_1 στέλνει ένα μήνυμα $IndexQuery(q, d+1, s)$, στον p_2 ο οποίος αντιστοιχεί στο κλειδί της κατάστασης

```
q1: /dblp//cite
q2: /dblp/phdthesis//series
q3: /dblp/article/*
q4: /dblp//chapter
q5: /dblp/*/*
q6: /dblp/book/title/sub/tt/sup/*
q7: /dblp/www/author
q8: /dblp/proceedings/school
q9: /dblp/article/year
q10: /*/inproceedings/cite
q11: /dblp//cite
q12: /dblp/incollection/*/ref
q13: //mastersthesis//crossref
q14: /*/article/month
```

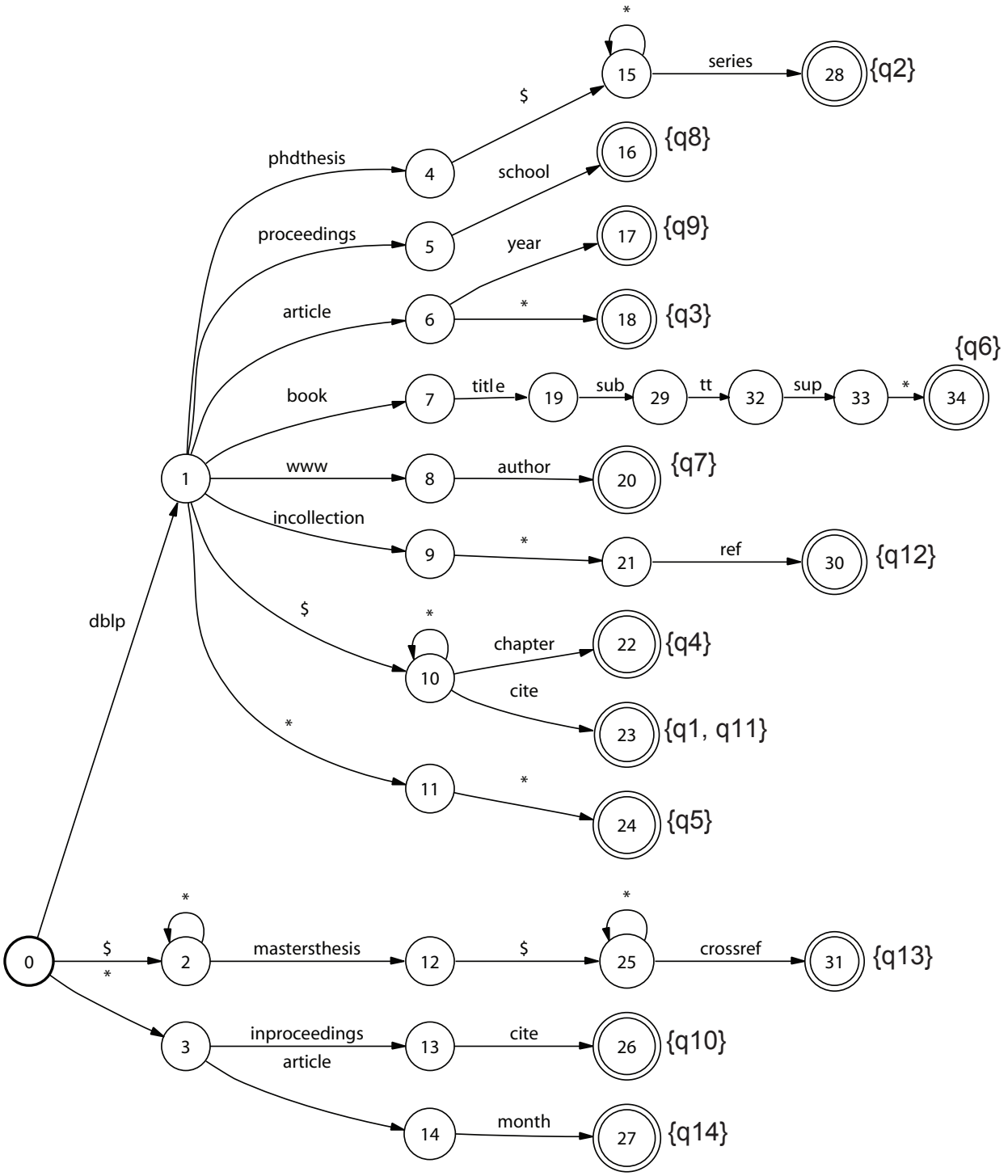
Σχήμα 3.10: Οι επερωτήσεις του αυτόματου που παρουσιάζεται στο Σχήμα 3.11

$st.Key + t$. Σημειώνεται και πάλι ότι το αυτόματο της επερώτησης είναι γραμμικό, δηλαδή δεν έχει διακλαδώσεις.

Στον Αλγόριθμο 3.1 παρουσιάζεται η διαδικασία $IndexQuery()$, η οποία αφορά την κατανεμημένη κατασκευή του αυτόματου για $l = 0$. Για απλότητα, σε αυτό το κεφάλαιο οι αλγόριθμοι παρουσιάζονται με τη παράμετρο l να είναι 0, ενώ στο επόμενο κεφάλαιο που αφορά την υλοποίηση, οι αλγόριθμοι παρουσιάζονται για οποιοδήποτε l. Στον Αλγόριθμο 3.1, σε περίπτωση που έχουμε $l > 0$, τότε θα αποθηκεύεται ένα μεγαλύτερο τμήμα του αυτόματου στο κάθε κόμβο, αντί μια κατάσταση μόνο και οποιεσδήποτε καταστάσεις μπορούμε να μεταβούμε μέσω μίας κενής μετάβασης που έχουμε για $l = 0$. Σημειώνεται, ότι τα μηνύματα που ανταλλάσσονται κατά τη κατασκευή του αυτόματου είναι ανεξάρτητα της τιμής της παραμέτρου l. Αυτό συμβαίνει επειδή ολόκληρο το αυτόματο που αντιστοιχεί στην κάθε επερώτηση είναι διαθέσιμη σε κάθε βήμα της εκτέλεσης του αλγορίθμου κατασκευής.

3.2.2 Εκτέλεση Αυτόματου

Προτείνονται δύο μεθοδολογίες για την εκτέλεση του αυτόματου πάνω από το κατανεμημένο δίκτυο στο οποίο έχουμε κατανέμει τις καταστάσεις του αυτόματου. Όμοια με το YFilter, διατηρούμε κάποιες στοίβες κατά τη διάρκεια της εκτέλεσης. Για κάθε ενεργή κατάσταση (active state), θα πρέπει να ανακτήσουμε τις καταστάσεις στόχους οι οποίες λαμβάνονται μέσω των μεταβάσεων οι οποίες προέρχονται από την αρχική κατάσταση που επεκτείνουμε, και ταιριάζουν με το στοιχείο του XML εγγράφου το οποίο χρησιμοποιούμε.



Σχήμα 3.11: Αυτόματο επερωτήσεων Σχήματος 3.2. Η κατάσταση με αριθμό i αντιστοιχεί στον υπεύθυνο για αυτή κόμβο, p_i

Αλγόριθμος 3.1 n.IndexQuery(q,d,s): Αλγόριθμος ευρετηριασμού μίας επερώτησης

```
1: st is state at depth d of q.NFA
2: if n.states does not contain st then
3:    add st to n.states
4: end if
5: //final state
6: if d = q.NFA.depth() then
7:    add q to st.queries
8: else
9:    t ← transition label at depth d
10:   if there is a transition labelled t from st to a new state st' then
11:      nextPeer ← Lookup(st'.key)
12:      nextPeer.IndexQuery(q, d + 1, s)
13:   else
14:      add a transition labeled t from st to st'
15:      if t = ϵ then
16:         st'.selfchild ← true
17:      end if
18:      st'.key ← st.key + t
19:      nextPeer ← Lookup(st'.key)
20:      nextPeer.IndexQuery(q, d + 1, s)
21:   end if
22: end if
```

3.2.2.1 Επαναληπτική Προσέγγιση

Από τη προηγούμενη περιγραφή προκύπτει άμεσα από την κεντρικοποιημένη προσέγγιση, η επαναληπτική μεθοδολογία (iterative way). Σε αυτή τη προσέγγιση, ο κόμβος ο οποίος δημοσιεύει το XML έγγραφο αναλαμβάνει επίσης να διατηρεί τις στοίβες τοπικά. Η διαφορά βρίσκεται στην επέκταση των στοιχείων του XML εγγράφου, η οποία προωθείται στους κόμβους οι οποίοι, σύμφωνα με τον αλγόριθμο κατανομής καταστάσεων, είναι υπεύθυνοι για καταστάσεις στόχους τις οποίες θέλουμε να ανακτήσουμε, παρατηρώντας τις ενεργές καταστάσεις και τις μεταβάσεις τους. Στους Αλγορίθμους 3.2 και 3.3, παρουσιάζονται αντίστοιχα οι λειτουργίες που εκτελεί ο κόμβος ο οποίος δημοσιεύει το έγγραφο, καθώς και οι ενέργειες στις οποίες προβαίνει ο κόμβος ο οποίος λαμβάνει μια αίτηση επέκτασης κάποιας κατάστασης για την οποία είναι υπεύθυνος.

Πιο συγκεκριμένα, ο κόμβος p ο οποίος δημοσιεύει ένα έγγραφο ακολουθώντας τα βήματα που περιγράφονται στη διαδικασία $PublishDocument(doc)$, όπου doc το υπό δημοσίευση

κείμενο. Για κάθε ενεργή κατάσταση st, ο κόμβος p ο οποίος δημοσιεύει, στέλνει ένα μήνυμα $ExpandState(st, event, publisherId)$, όπου st η ενεργή κατάσταση, $event$ το γεγονός που παράγεται από το συντακτικό αναλυτή του εγγράφου (το οποίο για να αποστέλλεται στον κόμβο για επέκταση είναι σίγουρα ένα γεγονός που ορίζει την έναρξη ενός στοιχείου, *start of element*) καθώς και $publisherId$ το αναγνωριστικό του κόμβου ο οποίος δημοσιεύει. Όπως και στο YFilter, αρχικά η μόνη ενεργή κατάσταση είναι η κατάσταση εκκίνησης (start state).

Στη περίπτωση που ένας κόμβος r λαμβάνει ένα μήνυμα *ExpandState*, το οποίο εμπεριέχει τα δεδομένα που προαναφέρθηκαν ως ορίσματα, υπολογίζει τις καταστάσεις στόχους από τις μεταβάσεις της κατάστασης st οι οποίες ικανοποιούν το στοιχείο που λαμβάνει σαν γεγονός $start\ of\ element$ στις παραμέτρους του. Στη συνέχεια, επιστρέφει με ένα κατάλληλο μήνυμα τις καταστάσεις στο κόμβο που δημοσιεύει, p, και αυτός συνεχίζει με τον ίδιο τρόπο την εκτέλεση μέχρι την πλήρη σάρωση του εγγράφου. Όταν το πλήρες έγγραφο σαρωθεί, αναλαμβάνει να πληροφορήσει όσους συνδρομητές επερωτήσεις στις οποίες έχουν εγγραφεί ικανοποιήθηκαν, ενώ αντιμετωπίζει περιπτώσεις οπισθοχώρησης όπως και η κεντρικοποιημένη προσέγγιση. Στη περίπτωση που το l είναι μεγαλύτερο του μηδενός, η αντιμετώπιση θα πρέπει να προσαρμοστεί κατάλληλα, όπως θα δούμε στην ενότητα που αφορά την υλοποίηση. Η επαναληπτική μέθοδος δεν παρουσιάζει πολύ καλή απόδοση, με δεδομένο το ότι δεν εκμεταλλεύεται πλήρως τις δυνατότητες ενός κατανεμημένου δικτύου, αλλά επιβαρύνει ένα κόμβο για την εκτέλεση, το κόμβο που δημοσιεύει, καθώς και το γεγονός ότι στηρίζεται στην επιστροφή καταστάσεων από άλλους κόμβους, σημείο κατά το οποίο ο κόμβος που δημοσιεύει είναι "απασχολημένος" με τη δημοσίευση, αλλά απλά περιμένει.

3.2.2.2 Αναδρομική Προσέγγιση

Παρατηρώντας τα πολλαπλά μονοπάτια πάνω στο αυτόματο του δικτύου τα οποία είναι αποθηκευμένα στις τοπικές στοίβες της επαναληπτικής μεθοδολογίας και επαναληπτικά διατρέχονται, καθώς και την ανάγκη οπισθοδρόμησης στη περίπτωση που ένα μονοπάτι φτάσει στο τέλος του, στο [41] παρουσιάζεται μια εναλλακτική μεθοδολογία η οποία εκμεταλλεύεται τις δυνατότητες του δικτύου ομότιμων και του αλγορίθμου κατασκευής, εκτελώντας παράλληλα το κάθε μονοπάτι του αυτόματου.

Η αναδρομική αυτή μεθοδολογία έχει ως εξής: Ο κόμβος ο οποίος δημοσιεύει, προωθεί το XML έγγραφο στον κόμβο ο οποίος είναι υπεύθυνος για την κατάσταση εκκίνησης, έτσι ώστε να αρχίσει η εκτέλεση του αυτόματου πάνω από το δίκτυο. Η εκτέλεση προχωράει αναδρομικά, με

Αλγόριθμος 3.2 n.PublishDocument(doc): Δημοσίευση ενός XML εγγράφου με την επαναληπτική μεθοδολογία

```
1: add startState to activeStates
2: runtimeStack.push(activeStates)
3: for each event from parsing doc do
4:   if event is a startElement then
5:     for each st in activeStates do
6:       add st.queries to satisfiedQueries
7:       nextPeer ← Lookup( st.key )
8:       states ← nextPeer.ExpandState(st,event,myId)
9:       add states to targetStates
10:    end for
11:  else
12:    runtimeStack.pop()
13:  end if
14:  activeStates ← targetStates
15:  clear targetStates
16:  runtimeStack.push(activeStates)
17: end for
18: for each q in satisfiedQueries do
19:   notify subscriber of q
20: end for
```

Αλγόριθμος 3.3 n.ExpandState($st, event, publisherId$): Επέκταση κατάστασης ακολουθώντας τις μεταβάσεις - επαναληπτική μεθοδολογία

```
1: e ← element name of event
2: for each element in {e, ∗, ϵ} do
3:   st ← follow transition labeled element
4:   if st is not null then
5:     add st to targetStates
6:     if element is ϵ then
7:       nextPeer → LookUp(st.key)
8:       nextPeer.ExpandState(st,event,myId)
9:     end if
10:  end if
11: end for
12: if st.selfState is true then
13:   add st to targetStates
14: end if
15: return targetStates
```

κάθε κόμβο ο οποίος είναι υπεύθυνος για μια ενεργή κατάσταση να συνεχίζει την εκτέλεση. Σε αυτή τη περίπτωση, η στοίβα runtime δεν διατηρείται ρητά, αλλά υπάρχει έμμεσα στο μονοπάτι εκτέλεσης που ακολουθούμε κατά την αναδρομική εκτέλεση.

Υπάρχουν δύο περιπτώσεις όπου η εκτέλεση γίνεται παράλληλη. Στην ουσία, αν φανταστούμε το αυτόματο πάνω από το δίκτυο, για παράλληλη εκτέλεση μιλάμε στη περίπτωση που υπάρχουνε διακλαδώσεις, και φανερώνονται δύο ή περισσότερα μονοπάτια εκτέλεσης, ή ακόμα εάν η δενδρική μορφή του εγγράφου XML το οποίο δημοσιεύεται, παρουσιάζει μια τέτοια διακλάδωση. Πιο φορμαλιστικά, οι δύο περιπτώσεις που προαναφέρουμε, αφορούν την περίπτωση όπου περισσότερες από μία καταστάσεις στόχοι παρουσιάζονται καθώς επεκτείνουμε μια κατάσταση. Στη περίπτωση αυτή, ο κόμβος ο οποίος ανέλαβε την επέκταση, ειδοποιεί τους κόμβους οι οποίοι είναι υπεύθυνοι ο κάθε ένας για τη κάθε κατάσταση στόχο, οι οποίοι συνεχίζουν παράλληλα την εκτέλεση σε n μονοπάτια, όπου n είναι ο αριθμός των καταστάσεων στόχων. Η δεύτερη περίπτωση που αφορά το XML έγγραφο, αφορά τη περίπτωση όπου ένα στοιχείο του έχει *αδέλφια*, πάντα στη δενδρική αναπαράστασή του. Σε αυτή τη περίπτωση, δημιουργούνται τόσα μονοπάτια εκτέλεσης όσα τα αδέλφια που παρουσιάζονται στο συγκεκριμένο βάθος, στο συγκεκριμένο έγγραφο. Στον Αλγόριθμο 3.4, παρουσιάζονται οι ενέργειες στις οποίες προβαίνει ένας κόμβος, ο οποίος είναι υπεύθυνος για κάποια ενεργή κατάσταση, κατά την εκτέλεση ενός μονοπατιού.

Πιο συγκεκριμένα, έστω ο κόμβος p ο οποίος δημοσιεύει ένα XML έγγραφο, αποστέλλοντας ένα μήνυμα $RecExpandState(st, path, events)$ στον κόμβο r, ο οποίος είναι υπεύθυνος για τη κατάσταση st. Στο Chord είναι ο $succ("start")$, ενώ στο $Pastry$ ο αριθμητικά πλησιέστερος. Η κατάσταση st είναι η κατάσταση εκκίνησης του αυτόματου και $path$ είναι μια στοίβα που αρχικά περιέχει μόνο το ζευγάρι γεγονότος StartOfDocument και τη κατάσταση εκκίνησης, ενώ η λίστα $events$ αφορά τα γεγονότα που παράγονται κατά την εκτέλεση του εγγράφου. Ποιο συγκεκριμένα, τα γεγονότα που δηλώνουν την έναρξη και την λήξη ενός στοιχείου, εμπλουτίζονται με την δομική αναπαράσταση που παρουσιάστηκε στην Ενότητα 2.3.4, έτσι ώστε να μπορεί εύκολα να εξακριβωθεί η δομική σχέση (απόγονος, πρόγονος) μεταξύ στοιχείων. Αυτό χρειάζεται συγκεκριμένα, στην εύρεση των αδελφών στοιχείων, έτσι ώστε να δημιουργηθούν διαφορετικά μονοπάτια εκτέλεσης, όπως περιγράφηκε.

Όταν ο κόμβος r', ο οποίος είναι υπεύθυνος για τη κατάσταση st, λάβει ένα μήνυμα RecExpandState, πρώτα ελέγχει αν το επόμενο στοιχείο του εγγράφου e έχει αδέλφια. Το e, υπάρχει στη κορυφή της στοίβας εκτέλεσης, και έτσι μπορούμε να αποφασίσουμε πιο είναι το επόμενο

στοιχείο. Εάν έχει αδέλφια, τότε υπολογίζονται οι μεταβάσεις για κάθε ένα, χρησιμοποιώντας τις μεταβάσεις της κατάστασης και τη τοπική δομή καταστάσεων $states$. Έστω δηλαδή, st' η κατάσταση στην οποία μεταβαίνουμε από την st ακολουθώντας τη μετάβαση t, ο r' την προσθέτει στο μονοπάτι εκτέλεσης και στη συνέχεια προωθεί το μήνυμα $RecExpandState(st', path, events)$ στο κόμβο r'', ο οποίος είναι υπεύθυνος για το κλειδί της $st' = st.Key + transition$. Έτσι, αν υποθέσουμε ότι $e_1, ..., e_s$ είναι τα στοιχεία - αδέλφια και $TS(e_1), ..., TS(e_s)$ τα σύνολα καταστάσεων στόχων τα οποία λαμβάνονται για κάθε στοιχείο, τότε ο r' θα αποστείλει $\sum_{i=1}^{i=s} |TS(e_i)|$ διαφορετικά μηνύματα, ένα για κάθε διαφορετικό μονοπάτι εκτέλεσης. Προφανώς, το κάθε μονοπάτι εκτέλεσης συνεχίζει μέχρι το τέλος του τμήματος του αυτόματου το οποίο εκτελείται. Ο κάθε κόμβος ο οποίος συμμετέχει στην εκτέλεση, είναι και υπεύθυνος για την ειδοποίηση των συνδρομητών για επερωτήσεις οι οποίες έχουν ικανοποιηθεί.

Πάλι, τα βήματα έχουν περιγραφεί στη περίπτωση όπου $l = 0$. Για μεγαλύτερες τιμές, οι λειτουργίες $ExpandState$ και $RecExpandState$ στην ουσία, αντί να εξομοιώνουν την συνάρτηση μετάβασης δ, εξομοιώνουν την επεκτεταμένη συνάρτηση μετάβασης $\hat{\delta}$. Η αναδρομική εκτέλεση που περιγράφηκε αποστέλλει όλα τα γεγονότα του XML εγγράφου, επομένως υποθέτει ότι το μέγεθος του εγγράφου είναι σχετικά μικρό. Μπορεί να τροποποιηθεί σε διαφορετική περίπτωση, έτσι ώστε να αποστέλλει τμήματα του αυτόματου.

3.3 Συμπεράσματα

Σε αυτό το κεφάλαιο έχουμε παρουσιάσει το σύστημα XML φιλτραρίσματος YFilter, αναλύοντας τις συνιστώσες του συστήματος καθώς και το τρόπο που διαχειρίζεται τις επερωτήσεις των συνδρομητών και τα XML αρχεία που δημοσιεύονται. Στη συνέχεια βασιζόμενοι και στη προηγούμενη περιγραφή, αναλύσαμε τη κατανεμημένη προσέγγιση που ακολουθείται στο [41]. Στη περιγραφή αναφερθήκαμε στον ευρετηριασμό των επερωτήσεων και συγκεκριμένα στο τρόπο κατανομής των καταστάσεων του αυτομάτου πάνω από το δίκτυο, επεκτείνοντας έτσι τη προσέγγιση του YFilter. Επίσης αναφερθήκαμε στην επαναληπτική και αναδρομική μεθοδολογία εκτέλεσης του αυτομάτου κατά τη δημοσίευση ενός εγγράφου καθώς και στους τρόπους εκμετάλλευσης του έμφυτου παραλληλισμού που υφίσταται στα δίκτυα ομότιμων. Η περιγραφή αυτή μας επιτρέπει να αναλύσουμε με αρκετή λεπτομέρεια την υλοποίηση του συστήματος [41] με το σύστημα FreePastry στο επόμενο κεφάλαιο.

Αλγόριθμος 3.4 n.RecExpandState(st,path,events): Αναδρομική επέκταση καταστάσεων σε κάθε μονοπάτι εκτέλεσης - αναδρομική μεθοδολογία

```
1: if st.isAcceptingState then
2:    add st.queries to satisfiedQueries
3: end if
4: event ← events.next()
5: if event.hasSiblings() then
6:    siblingEvents ← event.getSiblings()
7: else
8:    siblingEvents ← {event}
9: end if
10: for each e in siblingEvents do
11:    if e.isStartElement then
12:       compute targetStates from st for input e
13:       for each s in targetStates do
14:          newPath = path.add(e, s)
15:          next = Lookup (s.key )
16:          next.RecExpandState (s, newPath, events)
17:       end for
18:    end if
19:    for each q in satisfiedQueries do
20:       notify subscriber of q
21:    end for
22: end for
```

Κεφάλαιο 4

Υλοποίηση Συστήματος χρησιμοποιώντας το Σύστημα FreePastry

Σε αυτό το κεφάλαιο, θα περιγράψουμε την υλοποίηση του συστήματος που έχει αναλυθεί, περιγράφοντας τις λεπτομέρειες που αφορούν μία τέτοια υλοποίηση, χρησιμοποιώντας την ανοικτού κώδικα (Open Source) υλοποίηση του Pastry, το FreePastry[1]. Η γλώσσα υλοποίησης είναι η Java[2], ενώ η βιβλιοθήκη που εφαρμόζει το FreePastry είναι επίσης υλοποιημένη σε Java. Επίσης, χρησιμοποιούμε τη βιβλιοθήκη του YFilter[3] από την οποία αντλούμε (τροποποιώντας όπου χρειάζεται) τις δομές που αφορούν τις XPath επερωτήσεις και τα XML έγγραφα, ενώ για τη συντακτική ανάλυση των XML εγγράφων και τη δημιουργία των αντίστοιχων γεγονότων χρησιμοποιούμε το συντακτικό αναλυτή Sax parser[4]. Επίσης, είναι σημαντικό να σημειωθεί ότι η υλοποίηση παρέχει ελευθερία στη παράμετρο l, αφού αυτή μπορεί να είναι οποιοσδήποτε ακέραιος.

Επίσης σημειώνεται ότι έχει χρησιμοποιηθεί η βιβλιοθήκη μηχανών πεπερασμένων καταστάσεων της AT&T [5] (AT&T Finite State Machine Library), με τη χρήση της οποίας μπορούν να παραχθούν σε μορφή PDF αυτόματα τα οποία δημιουργούμε πάνω από το δίκτυο. Έτσι μπορεί να παρέχεται ένας έλεγχος ορθότητας στο χρήστη καθώς και η εύκολη και άμεση παρατήρηση του κατανεμημένου αυτόματου το οποίο φτιάχνεται πάνω από το δίκτυο. Παράδειγμα της εξόδου αυτής του συστήματος, αποτελεί το Σχήμα 3.11.

[1] http://freepastry.rice.edu/
[2] http://www.java.com/
[3] http://yfilter.cs.umass.edu/
[4] http://www.saxproject.org/
[5] http://www.research.att.com/~fsmtools/fsm/

Στην πρώτη ενότητα του κεφαλαίου, την Ενότητα 4.1, εισάγουμε τις λεπτομέρειες τις υλοποίησης που αφορούν την κατασκευή του αυτόματου. Παρουσιάζουμε με λεπτομέρεια τον αλγόριθμο ευρετηριασμού που χρησιμοποιούμε, καθώς και τη διαδικασία ευρετηριασμού σε ένα δίκτυο Pastry.

Στην Ενότητα 4.2 εξετάζουμε τις λεπτομέρειες που αφορούν την εκτέλεση του αυτόματου. Πιο συγκεκριμένα, στον Αλγόριθμο 4.2 παρουσιάζουμε το βασικό αλγόριθμο ο οποίος ανακτά καταστάσεις από το δίκτυο (είτε τοπικά, είτε από κάποιο άλλο κόμβο), όπως και το βασικό αλγόριθμο ο οποίος επεκτείνει μια κατάσταση με οδηγό ένα στοιχείο του εγγράφου XML, (Αλγόριθμος 4.3). Επίσης, παρουσιάζουμε αλγορίθμους επέκτασης στοιχείων από κάποιο κόμβο στο δίκτυο, καθώς και τους βασικούς αλγορίθμους επαναληπτικής και αναδρομικής εκτέλεσης του αυτόματου, πάνω από το δίκτυο.

Τέλος, στην Ενότητα 4.3, αναφέρουμε κάποια άλλα θέματα και λεπτομέρειες που αφορούν την υλοποίηση.

4.1 Κατασκευή Αυτόματου

Κατά τη κατασκευή του αυτόματου, οι κόμβοι ανταλλάσσουν μηνύματα[6] της μορφής IndexQuery(Query q, Integer Depth, Id Subscriber, Integer l) , όπου q η επερώτηση XPath, $Depth$ ο ακέραιος που αφορά το βάθος που αναλύεται στην επερώτηση, $subscriber$ το αναγνωριστικό του συνδρομητή σύμφωνα με το Pastry καθώς και l η παράμετρος l, η οποία θυμίζουμε ότι έχει να κάνει με το μέγεθος του αυτόματου το οποίο θα αποθηκευτεί στον εκάστοτε κόμβο. Σημειώνεται, ότι το l δίνεται και ως παράμετρος στην εφαρμογή κατά την εκκίνηση του κάθε κόμβου, γιατί στην ουσία είναι παράμετρος του όλου συστήματος και δεν αφορά μόνο τον ευρετηριασμό των επερωτήσεων. Η επερώτηση, βασίζεται στο τύπο Query, όπως αυτός ορίζεται στις δομές του YFilter, με τη διαφορά ότι έχει τροποποιηθεί έτσι ώστε να είναι σειριοποιήσιμος (serializable)[7].

[6] Σημειώνεται ότι στους αλγορίθμους που παρουσιάζουμε σε αυτό το κεφάλαιο, για τη προώθηση ενός μηνύματος σε κάποιο κόμβο, χρησιμοποιούμε τη συνάρτηση $node.route(Message, Data)$, η οποία παραπέμπει στην αντίστοιχη συνάρτηση του $Pastry$, η οποία αποστέλει το μήνυμα $Message$ στο κόμβο ο οποίος έχει αναγνωριστικό (Id) αριθμητικά πιο κοντά στο $Hash(Data)$.

[7] Ένα αντικείμενο, πρέπει να είναι σειριοποιήσιμο έτσι ώστε να παρέχεται η δυνατότητα να μετατραπεί σε κατάλληλη μορφή έτσι ώστε να είναι δυνατόν να αποθηκευτεί (για παράδειγμα σε κάποιο αρχείο) ή να αποσταλεί πάνω από ένα δίκτυο.

Η επερώτηση, στη μορφή που δίνεται μπορεί να αναλυθεί σε βήματα, τα οποία στην ουσία αντιπροσωπεύουν το τμήμα του αυτόματου το οποίο αντιστοιχεί στην επερώτηση, με δεδομένο το ότι η επερώτηση είναι γραμμική. Διαφορετικά, θα είχαμε πολλά μονοπάτια, τα οποία αντίστοιχα θα αναλύονταν σε βήματα. Επίσης, σημαντικό είναι να σημειωθεί η εσωτερική αναπαράσταση της επερώτησης, στη περίπτωση που έχουμε κενή μετάβαση. Στην ουσία, υπάρχει μόνο μια περίπτωση να έχουμε κάποια κενή μετάβαση, όταν η επερώτηση περιλαμβάνει τον άξονα "//". Σε αυτή τη περίπτωση, το αντίστοιχο βήμα, περιλαμβάνει το χαρακτήρα "$". Με αυτό το χαρακτήρα, συμβολίζουμε ότι έχουμε μία κενή μετάβαση στην επόμενη κατάσταση *και* ένα βρόγχο στην επόμενη κατάσταση, ο οποίος με οποιοδήποτε χαρακτήρα επιστρέφει στην ίδια κατάσταση (Self-Child). Για παράδειγμα, η επερώτηση $/dblp/paper//author$, δίνει τα βήματα $[dblp, paper, \$, author]$, ενώ η επερώτηση $/dblp/paper/author$ τα βήματα $[dblp, paper, author]$.

4.1.1 IndexQuery()

Στον Αλγόριθμο 4.1, βλέπουμε τη διαδικασία κατασκευής του αυτόματου χρησιμοποιώντας το FreePastry. Στην κλήση της συνάρτησης, έχουν προστεθεί η κατάσταση st, καθώς και η δυαδική μεταβλητή $isRecursive$. Η χρησιμότητα της κατάστασης st, είναι εκτός από το να σηματοδοτεί σε ποιο κόμβο του αυτόματου του δικτύου πρέπει να ξεκινήσει η εκτέλεση, να πληροφορεί κάποιο κόμβο στην περίπτωση όπου η κατάσταση από την οποία θα ξεκινήσει να ευρετηριάζει έχει βρόγχο στον εαυτό της. Δηλαδή να πληροφορεί εάν η παράμετρος $selfChild$ είναι αληθής, πάντα σε περίπτωση όπου αυτή η κατάσταση είναι καινούργια στο κατανεμημένο αυτόματο. Εναλλακτικά, θα μπορούσε ο κάθε κόμβος να ανακτά τη κατάλληλη κατάσταση διαβάζοντας και συνθέτοντας τα προηγούμενα βήματα της επερώτησης. Η διαδικασία αυτή μπορεί να παρατηρηθεί στις γραμμές 1-3, ενώ στις γραμμές 4-6, γίνεται έλεγχός εάν έχουμε φτάσει στο τέλος των βημάτων της επερώτησης, κάτι που σημαίνει ότι η παρούσα κατάσταση st είναι και τελική κατάσταση.

Στις γραμμές 7 ως 16, ελέγχουμε αν υπάρχει μετάβαση με ετικέτα t, η οποία ετικέτα αντιστοιχεί στο βάθος $Depth$ στην επερώτηση q. Εάν δεν υπάρχει, τότε δημιουργούμε την αντίστοιχη μετάβαση στη κατάσταση st. Η κατάσταση στην οποία μεταβαίνουμε από την st ακολουθώντας την ετικέτα t αποθηκεύεται στη τοπική μεταβλητή st'.

Στη συνέχεια, θα πρέπει να σκεφτούμε ως εξής: Στη περίπτωση που το t είναι το $, τότε ανεξαρτήτως l, στο κόμβο αυτό θα αποθηκευτεί και η κατάσταση st' τοπικά. Έτσι, θα πρέπει οι

Αλγόριθμος 4.1 IndexQuery(Query q, Int $Depth$, Subscriber Id, State st, Int l, Bool $isRecursive$)

```
1: if st.Key exists in local states list then
2:    st = states.get(st.Key)
3: end if
4: if Depth = q.length then
5:    add q to st.satQueries
6: end if
7: t = transition label of q at depth d
8: if t label does not exist from state st then
9:    add transition labelled t from st to st'
10:   if (t == $) then
11:      st' = st.getTransition(t)
12:      set st'.selfChild = true
13:   end if
14: else
15:   st' = st.getTransition(t)
16: end if
17: // whether transition t existed or not, if current element is $,
18: // then we should fix the transition table of state st'
19: if t == $ then
20:   if Depth < q.length then
21:      t' = transition label of q at depth d+1
22:      if transition t' does not exist for state st' add it //it can not be $ again
23:   end if
24: end if
25: if (Depth < q.length) ∧ (l > 0) then
26:   if t == $ then
27:      IndexQuery(q, Depth+1, Id, st, l, false)
28:   else
29:      IndexQuery(q, Depth+1, Id, st, l-1, false)
30:   end if
31: end if
32: if not isRecursive then
33:   RouteMessage ← IndexQuery(q, Depth+1, Id, st', l, true)
34:   node.route(RouteMessage)
35: end if
```

μεταβάσεις της κατάστασης st' να είναι συνεπείς με την εικόνα του αυτόματου. Επομένως στη περίπτωση που υπάρχουν και άλλα στοιχεία στην επερώτηση (δηλαδή δεν έχουμε φτάσει στο σημείο όπου $Depth = q.length$), προσθέτουμε μετάβαση με ετικέτα t', η οποία αντιστοιχεί στο στοιχείο με $Depth + 1$ βάθος στην επερώτηση, στη κατάσταση st'. Σημειώνεται, ότι δεν υπάρχει περίπτωση το στοιχείο να είναι πάλι το $, γιατί αυτό θα σήμαινε ότι είχαμε δύο άξονες απογόνου συνεχόμενους, κάτι που δεν αποτελεί σωστή σύνταξη μίας επερώτησης. Το κομμάτι αυτό, αφορά μέχρι και τη γραμμή 24.

Στη συνέχεια (γραμμές 25-31), υλοποιείται η αναδρομική κλήση του αλγορίθμου, για να ικανοποιήσουμε το μέγεθος της παραμέτρου l. Η κλήση, γίνεται όσο δεν έχουμε εξαντλήσει το βάθος του αυτόματου της επερώτησης - γιατί διαφορετικά δεν θα είχε νόημα - αλλά και εφ᾽ όσον η παράμετρος l είναι μεγαλύτερη του μηδενός. Σε περίπτωση όπου η ετικέτα t ήταν η κενή μετάβαση, τότε η αναδρομική κλήση γίνεται αυξάνοντας το βάθος ($Depth$) αλλά διατηρώντας σταθερή τη παράμετρο l, αφού η κενή μετάβαση δεν συνεισφέρει σε αυτήν. Διαφορετικά, μειώνουμε κατά ένα το l στη κλήση. Δεν χρειάζεται να ανησυχήσουμε για τη περίπτωση όπου το l γίνει μηδέν, και το t είναι η κενή μετάβαση. Σε αυτή τη περίπτωση, μια αναδρομική κλήση θα ήταν περιττή, αφού η κατάσταση st' έχει προστεθεί στη τοπική λίστα καταστάσεων ήδη.

Τέλος, στη γραμμή 32 έως και 34, γίνεται η προώθηση του μηνύματος στον επόμενο κόμβο, ο οποίος είναι υπεύθυνος για τη κατάσταση st'. Η δυαδική μεταβλητή isRecursive είναι αληθής μόνο στη περίπτωση όπου εξετάζουμε κάποια αναδρομική κλήση, ενώ είναι ψευδής όταν η εκτέλεση βρίσκεται στο πρώτο τρέξιμο. Τότε και εφ᾽ όσον δεν έχουμε εξετάσει ολόκληρη την επερώτηση, γίνεται η προώθηση μηνύματος στον επόμενο κόμβο, με προσαύξηση του βάθους.

4.2 Εκτέλεση Αυτόματου

Σε αυτή την ενότητα, θα περιγράψουμε τους αλγόριθμους για την εκτέλεση του αυτόματου, με δεδομένο κάποιο έγγραφο XML το οποίο δημοσιεύεται. Αρχικά, θα περιγράψουμε κάποιους γενικούς αλγορίθμους, οι οποίοι υποβοηθούν την εκτέλεση, ενώ στη συνέχεια, θα περιγράψουμε την επαναληπτική και αναδρομική μέθοδο εκτέλεσης.

4.2.1 getTargetState()

Στον Αλγόριθμο 4.2, παρουσιάζονται οι ενέργειες στις οποίες προβαίνει ένας κόμβος για να ανακτήσει μία κατάσταση στόχο. Για την ανάκτηση, ο κόμβος λαμβάνει υπ᾽ όψην την παράμετρο l. Πιο συγκεκριμένα, η περίπτωση όπου το $l = 0$ θέλει κάποιο ειδικό χειρισμό. Αν το $l > 0$, τότε η κατάσταση στόχος που αναζητούμε σίγουρα[8] υπάρχει στη τοπική δομή $states$ του κόμβου. Σε διαφορετική περίπτωση, όπου δηλαδή το $l = 0$, μόνο αν το στοιχείο μετάβασης είναι η κενή μετάβαση, η κατάσταση στόχος βρίσκεται τοπικά στο κόμβο. Διαφορετικά, ο κόμβος θα πρέπει να ζητήσει τη κατάσταση από τον κόμβο ο οποίος είναι υπεύθυνος για τη κατάσταση αυτή. Οι προηγούμενες διαπιστώσεις, στηρίζονται φυσικά στο γεγονός ότι κατά την εκτέλεση, από κάθε κόμβο ζητάμε να επεκτείνει το πολύ μήκος l στοιχεία, δηλαδή ένα τμήμα αυτόματου μήκους l. Η ειδική περίπτωση αφορά το $l = 0$, όπου επεκτείνουμε ένα στοιχείο.

Αλγόριθμος 4.2 getTargetState(State st, String label, int l)

```
1: if label = $ then
2:    return states.get(st.Key + label)
3: else if l = 0 then
4:    Message ← Request state st' with name st.Name() + label
5:    stateReceived ← node.route(Message, st'.Name())
6:    return stateReceived
7: else
8:    // case where l > 0
9:    return states.get(st.Key + label)
10: end if
```

4.2.2 ExpandState()

Στην συνέχεια, θα περιγράψουμε τη διαδικασία επέκτασης μιας κατάστασης, με δεδομένα το στοιχείο το οποίο θα επεκταθεί και τις ενεργές καταστάσεις. Με άλλα λόγια, για όλες τις ενεργές καταστάσεις, ψάχνουμε όλες τις πιθανές μεταβάσεις με ετικέτα το στοιχείο $element$, οι οποίες ξεκινούν από κάποια ενεργή κατάσταση.

Στον Αλγόριθμο 4.3, βλέπουμε αυτή τη διαδικασία. Αρχικά, έχουμε ένα βρόγχο ο οποίος διατρέχει όλες τις ενεργές καταστάσεις. Στη συνέχεια, γίνεται έλεγχος εάν η κατάσταση περιέχει βρόγχο. Σε τέτοια περίπτωση, η ίδια η κατάσταση προστίθεται στις $targetStates$. Στη συνέχεια,

[8] Λέμε ότι η κατάσταση βρίσκεται σίγουρα, σε σχέση πάντα με το ποιες καταστάσεις ζητάμε κατά το τρέξιμο των αλγορίθμων - δεν εννοούμε σε καμία περίπτωση ότι οποιαδήποτε κατάσταση περιέχεται τοπικά στο κόμβο.

ένας βρόγχος ελέγχει για την ύπαρξη μετάβασης με ετικέτα κάποιο από τα τρία εξής στοιχεία: για το στοιχείο εισόδου, το χαρακτήρα μπαλαντέρ "*" καθώς και το χαρακτήρα που αντιστοιχεί στη κενή μετάβαση "$". Σε περίπτωση που μία τέτοια μετάβαση υπάρχει, η κατάσταση στόχος προστίθεται στη λίστα $targetStates$. Η ειδική περίπτωση που θέλει προσοχή, αφορά τη περίπτωση που έχουμε κενή μετάβαση. Στη περίπτωση αυτή, θα πρέπει αναδρομικά να ελεγχθεί η κατάσταση st' την οποία ανακτούμε ακολουθώντας τη κενή μετάβαση, σύμφωνα με τους γενικούς κανόνες εκτέλεσης αυτομάτων στη Θεωρία Υπολογισμού. Όμως, αν αναλογιστούμε το υποσύνολο των αυτομάτων που εξετάζουμε, δεν χρειάζεται να γίνει αναδρομικά έλεγχος για τη κενή μετάβαση, γιατί σύμφωνα με τη δομή των αυτομάτων που εξετάζουμε, δεν μπορεί να υπάρξουν συνεχόμενα δύο κενές μεταβάσεις. Επομένως, ελέγχουμε μόνο για την ύπαρξη του στοιχείου ($element$) και του χαρακτήρα "*".

Αλγόριθμος 4.3 ExpandState(States activeStates, String element, int l)

```
1: States targetStates = empty stack of States
2: for st in activeStates do
3:    if st.selfChild then
4:       add st to targetStates
5:    end if
6:    for e in {$, "*", element} do
7:       if st has transition with label t = e then
8:          st' = getTargetState(st, t, l)
9:          add st' to targetStates
10:         if t = $ then
11:            //need to check state st'
12:            for e_2 in {element, "*"} do
13:               if st' has transition with label t' then
14:                  st'' = getTargetState(st', t', l)
15:                  add st'' to targetStates
16:               end if
17:            end for
18:         end if
19:      end if
20:   end for
21: end for
```

4.2.3 Επαναληπτική Μέθοδος

Στην επαναληπτική μέθοδο, όπως έχουμε περιγράψει στο προηγούμενο κεφάλαιο, ο κόμβος ο οποίος δημοσιεύει ένα έγγραφο αναλαμβάνει να το εκτελέσει επίσης, λαμβάνοντας τις καταστάσεις στόχους από την επέκταση ενεργών καταστάσεων με βάση γεγονότα από τον συντακτικό αναλυτή του XML εγγράφου από τους υπεύθυνους κόμβους. Ο αλγόριθμος αναφέρεται σε οποιοδήποτε l με σκοπό να εκμεταλλευτεί ολόκληρο το τμήμα του αυτόματου της επερώτησης, το οποίο βρίσκεται αποθηκευμένο σε κάθε κόμβο. Γι᾽ αυτό, κατά την εκτέλεση, ο κόμβος αποστέλλει $f(l)$ στοιχεία στο κόμβο για επέκταση, με τη συνάρτηση $f(l)$ να ορίζεται ως:

$$f(l) = \begin{cases} l & l > 0 \\ 1 & l = 0 \end{cases}$$

Ο λόγος που ορίζεται η πιο πάνω συνάρτηση, είναι στην ουσία για να συλλάβει την μη γενικεύσιμη περίπτωση όπου $l = 0$. Για $l > 0$, προφανώς το αυτόματο που βρίσκεται αποθηκευμένο στον αντίστοιχο κόμβο έχει μήκος ίσο με l τουλάχιστον, $l + e_c$ για την ακρίβεια, όπου e_c ο αριθμός των κενών μεταβάσεων σε ένα μονοπάτι, με τη προϋπόθεση ότι το ίδιο το μονοπάτι που αφορά την επερώτηση έχει μέγεθος $\geq l + e_c$. Αν όμως το $l = 0$, τότε έχουμε μήκος το πολύ e_c, αφού θεωρητικά, χωρίς κενές μεταβάσεις μια κατάσταση είναι αποθηκευμένη σε ένα μόνο κόμβο. Αναγκαστικά όμως, για τη λειτουργία του αλγορίθμου, θα πρέπει να αποσταλεί τουλάχιστον ένα στοιχείο.

4.2.3.1 requestTargetStates()

Αλγόριθμος 4.4 requestTargetStates(State $activeState$, String[] $elements$)

1: $Message \leftarrow$ Request target states for $elements$ from state $activeState$
2: targetStates $\leftarrow$ $node$.route($Message$, $activeState.Name()$)
3: **return** $targetStates$

Κατά τη διάρκεια της επαναληπτικής εκτέλεσης, ο κόμβος ο οποίος δημοσιεύει ένα έγγραφο θα πρέπει με κάποιο τρόπο να στείλει ένα μήνυμα σε κάποιο άλλο κόμβο, ζητώντας του να επεκτείνει κάποια κατάσταση, με βάση κάποια στοιχεία του XML εγγράφου. Η αίτηση αυτή παρουσιάζεται στον Αλγόριθμο 4.4. Έστω ότι ο κόμβος που δημοσιεύει, p, καλεί αυτή τη συνάρτηση. Θα περάσει ως ορίσματα, τη κατάσταση $activeState$ από την οποία θα αρχίσει η επέκταση. Ο κόμβος ο οποίος θα λάβει το μήνυμα, θα είναι ο κόμβος ο οποίος θα είναι υπεύθυνος για τη

κατάσταση αυτή. Επίσης, ο p θα περάσει $f(l)$ το πολύ στοιχεία στη συνάρτηση, τα οποία ο παραλήπτης κόμβος θα χρησιμοποιήσει για να επεκτείνει τις ενεργές καταστάσεις.

4.2.3.2 returnTargetStates()

Αλγόριθμος 4.5 returnTargetStates(State st, String[] $elements$)

```
1: activeStates.push(st)
2: for element in elements do
3:    nTargetStates.add(ExpandState(activeStates, element, l))
4:    activeStates ← top of nTargetStates
5: end for
6: return nTargetStates
```

Στον Αλγόριθμο 4.5, βλέπουμε τη διαδικασία την οποία ακολουθεί ο κόμβος p', ο οποίος λαμβάνει την αίτηση για επέκταση. Ο κόμβος, θεωρεί σαν $activeStates$ αρχικά την κατάσταση από την οποία αρχίζει την επέκταση. Στη συνέχεια, καλεί την συνάρτηση $ExpandState$ που έχουμε περιγράψει στον Αλγόριθμο 4.3, για κάθε ένα από τα στοιχεία που είναι υπεύθυνος να επεκτείνει. Έτσι, αν έχει λάβει n στοιχεία για επέκταση, στο τέλος της διαδικασίας θα έχουμε n στοίβες, όπου η κάθε μια θα περιέχει τις καταστάσεις στόχους ($targetStates$), που λαμβάνονται μετά την επέκταση του αντίστοιχου στοιχείου. Ο λόγος που πρέπει να αποθηκευτούν και να αποσταλούν οι ενδιάμεσες καταστάσεις στόχοι και όχι μόνο αυτές του τελευταίου (n-οστού) στοιχείου, είναι γιατί στη περίπτωση οπισθοχώρησης, μπορεί να χρειαστεί να μεταβούμε σε κάποιο σύνολο καταστάσεων το οποίο ανήκει σε αυτά τα ενδιάμεσα βήματα.

Για να εξηγήσουμε επακριβώς τη προηγούμενη διαπίστωση, έστω ότι ο κόμβος p' λαμβάνει ένα τέτοιο μήνυμα επέκτασης. Έστω ότι $l = 2$ και ο p λαμβάνει τα στοιχεία el_1, el_2 από τη κατάσταση st. Θα επιστρέψει μια στοίβα ($nTargetStates$), η οποία θα περιέχει στη θέση (0) το σύνολο καταστάσεων στόχων οι οποία προέκυψαν από το el_1, ενώ στη θέση (1) το σύνολο καταστάσεων στόχων οι οποίες προέκυψαν από την επέκταση με βάση το στοιχείο el_2, με ενεργές καταστάσεις τις καταστάσεις στη θέση (0), όπως προκύπτει από τον αλγόριθμο. Έτσι, αν στη συνέχεια το επόμενο γεγονός του XML εγγράφου ήταν ένα $EndOfElement$, το οποίο σηματοδοτούσε το τέλος του στοιχείου el_1, ο κόμβος θα αφαιρούσε από τη κορυφή της στοίβας το σύνολο καταστάσεων (1), εκτελώντας έτσι την οπισθοδρόμηση και συνεχίζοντας την εκτέλεση του εγγράφου. Αυτό δεν θα ήταν δυνατό αν ο κόμβος ο οποίος εκτελούσε την επέκταση επέστρεφε μόνο τις καταστάσεις από το τελικό στοιχείο, δηλαδή το el_2.

4.2.3.3 publishIterative()

Στη συνέχεια, θα αναλύσουμε τον Αλγόριθμο 4.6, όπου παρουσιάζεται αναλυτικά η συνάρτηση επαναληπτικής δημοσίευσης στο υλοποιημένο σύστημα. Αρχικά, υπολογίζουμε τη συνάρτηση $f(l)$, η οποία αποθηκεύεται στη μεταβλητή $pathLength$. Στη συνέχεια προσθέτει την αρχική κατάσταση ($start$ state) στη λίστα ενεργών καταστάσεων, $activeStates$ και μετέπειτα μπαίνει στον βασικό βρόγχο εκτέλεσης.

Ο βρόγχος αυτός, στην ουσία συλλέγει τα στοιχεία τα οποία θα σταλούν σε κάποιο κόμβο για επέκταση, και εκτελείται μέχρι να εξαντληθούν τα γεγονότα που αφορούν το XML έγγραφο. Σε κάθε πέρασμα, αποθηκεύονται μέχρι $f(l)$ στοιχεία στην λίστα $events2Send$. Η αποθήκευση στοιχείων στη λίστα μπορεί να σταματήσει πιο νωρίς, αν συναντήσουμε ένα γεγονός που δηλώνει το τέλος κάποιου στοιχείου, $(EndOfElement)$. Στη συνέχεια, για κάθε ενεργή κατάσταση θα καλούμε τον Αλγόριθμο 4.4, ο οποίος θα επιστρέφει τις επεκτάσεις των στοιχείων αυτών για αυτή τη κατάσταση, επικοινωνώντας με τον υπεύθυνο κόμβο. Σημαντικό είναι να τονίσουμε ότι όλες οι ενεργές καταστάσεις, θα παράξουν n στοίβες, όπου n ο αριθμός των στοιχείων που επεκτείνονται σε αυτό το πέρασμα. Οι n αυτές στοίβες, θα προστεθούν στη στοίβα $runTimeStack$ σε κάθε βήμα, αφού οι καταστάσεις που κρατούν ελεγχθούν για τυχόν ικανοποιημένες επερωτήσεις. Στο τέλος του βήματος, αν το επόμενο στοιχείο στο έγγραφο είναι ένα $EndOfElement$, τότε θα χρειαστεί να αφαιρέσουμε τη στοίβα που βρίσκεται στη κορυφή της $runTimeStack$ στοίβας. Αυτό, ουσιαστικά μας φέρνει στη κατάσταση που ήμασταν πριν ακολουθήσουμε το μονοπάτι το οποίο έχει τελειώσει, έτσι ώστε να εξερευνήσουμε άλλα πιθανά μονοπάτια. Στη συνέχεια, θέτουμε ως ενεργές καταστάσεις τις καταστάσεις στόχους που βρίσκονται στη κορυφή της $runTimeStack$, και ο αλγόριθμος επαναλαμβάνεται.

Για να κατανοηθεί η προηγούμενη διαδικασία, μπορούμε να δώσουμε ένα σύντομο παράδειγμα, το οποίο αφορά το κομμάτι με τη συνάρτηση $merge$. Έστω ότι έχουμε δύο καταστάσεις στη στοίβα $activeStates$, τις st_1 και st_2, και 2 στοιχεία τα οποία θα επεκταθούν για κάθε μια από τις καταστάσεις, τα el_1 και el_2. Αρχικά, η στοίβα $currentTargetStates$ είναι κενή, και σε αυτή προστίθενται οι καταστάσεις που επιστρέφει η κλήση $requestTargetStates(st_1, elements2Send)$. Η συνάρτηση θα επιστρέψει στην ουσία δύο στοίβες, τη στοίβα που αφορά τις ενεργές καταστάσεις για το πρώτο στοιχείο και αντίστοιχα για το δεύτερο. Αυτές, αποθηκεύονται στη στοίβα $currentTargetStates$, όπου τώρα $currentTargetStates \leftarrow \{st_1 el_1 Target, \bar{st_1} el_2 Target\}$, με το σύνολο $st_i el_j Target$ να αντιστοιχεί στη επέκταση με ενεργή κατάσταση την st_i και στο στοιχείο el_j, ενώ ο συμβολισμός $\bar{st_i}$, αντιστοιχεί στις καταστάσεις στόχους που παράχθηκαν

από την επέκταση του προηγούμενου στοιχείου με ενεργή κατάσταση την st_i. Για την κατάσταση st_2, η αντίστοιχη κλήση θα επιστρέψει $\{st_2el_1Target, s\bar{t}_{2_e}l_2Target\}$. Τώρα, η συνάρτηση $merge$ θα ενώσει τις καταστάσεις σύνολα που προκύπτουν από τις δύο επεκτάσεις στις αντίστοιχες ενεργές καταστάσεις, και θα δώσει τη τελική μορφή στη στοίβα $currentTargetStates$: $\{st_1el_1Target \cup st_2el_1Target, s\bar{t}_1el_2Target \cup s\bar{t}_2el_2Target\}$.

4.2.4 Αναδρομική Μέθοδος

4.2.4.1 publishRecursive()

Στον Αλγόριθμο 4.7 βλέπουμε τις ενέργειες στις οποίες προβαίνει ένας κόμβος p, ο οποίος θέλει να δημοσιεύσει ένα XML έγγραφο. Στην ουσία, κατασκευάζουμε το ευρετήριο των στοιχείων του XML εγγράφου, όπως παρουσιάζεται στην Ενότητα 2.3.4. Στη συνέχεια, το ευρετήριο προωθείται στο κόμβο ο οποίος είναι υπεύθυνος για τη κατάσταση εκκίνησης. Η συνάρτηση που εκτελείται στη συνέχεια σε κάθε κόμβο στον οποίο φτάνει μια αίτηση αναδρομικής κλήσης, περιγράφεται στη συνέχεια.

4.2.4.2 RecExpandState()

Στον Αλγόριθμο 4.8, περιγράφεται η αναδρομική επέκταση καταστάσεων σε κάποιο μονοπάτι εκτέλεσης, κατά την αναδρομική εκτέλεση. Η συνάρτηση, λαμβάνει στην είσοδό της το όνομα της παρούσας κατάστασης, $stName$, την οποία θα ανακτήσει από τη τοπική λίστα καταστάσεων, το ευρετήριο των γεγονότων $docNodes$, τη θέση του γεγονότος με βάση το οποίο θα αρχίσουν οι επεκτάσεις καταστάσεων σε αυτό το πέρασμα, $currentIndex$, καθώς και τη θέση του γονιού του στο ευρετήριο, $parentIndex$.

Στη συνέχεια, ο κόμβος υπολογίζει τη συνάρτηση $f(l)$, τον αριθμό δηλαδή των στοιχείων που αυτός ο κόμβος θα χρησιμοποιήσει για τις επεκτάσεις και εισάγει στις ενεργές καταστάσεις τη κατάσταση με όνομα $stName$. Ο βασικός βρόγχος, στον οποίο ανακτούμε καταστάσεις στόχους και εκτελούμε το αυτόματο, συνεχίζεται μέχρι να επεκτείνουμε με βάση τα $f(l)$ στοιχεία, να βρούμε κάποιο στοιχείο το οποίο έχει αδέλφια ($siblings$) ή να συναντήσουμε το τέλος του μονοπατιού το οποίο επεκτείνουμε (βρίσκοντας ένα γεγονός $EndOfElement$). Στο εσωτερικό του βρόγχου, αρχικά ανακτούμε το επόμενο γεγονός στο ευρετήριο ($event$). Αν το στοιχείο έχει αδέλφια, τότε για κάθε ένα από αυτά ανακτούμε τις καταστάσεις στόχους στις οποίες μεταβαίνουμε, από τις ενεργές καταστάσεις ακολουθώντας μεταβάσεις με ετικέτα το όνομα του εκάστοτε

Αλγόριθμος 4.6 publishIterative(XML doc)

```
1: events = parsingEventsOf(doc)
2: if l = 0 then
3:    pathLength ← 1
4: else
5:    pathLength ← l
6: end if
7: request start state from responsible node
8: add start state to activeStates
9: while events list is not empty do
10:    elements2Send is a new list of elements
11:    while events2Send.length < pathLength do
12:       if events.isEmpty then
13:          break
14:       end if
15:       event ← events.getNextItem()
16:       if event is endElement then
17:          break
18:       else
19:          add event to list events2Send
20:       end if
21:    end while
22:    for st in activeStates do
23:       currentTargetStates.merge(requestTargetStates(st, elements2Send))
24:    end for
25:    for state in currentTargetStates do
26:       check for satisfied queries for state
27:       notify all subscribers
28:    end for
29:    add currentTargetStates on top of runTimeStack
30:    if next element in events is an end element then
31:       remove events.getNextItem()
32:       pop top of runTimeStack //backtracking
33:    end if
34:    set top of runTimeStack as the activeStates stack
35: end while
```

Αλγόριθμος 4.7 publishRecursive(XML doc)

```
1: docNodes ← constructIndex(doc.parsingEvents())
2: Message ← RecExpandState("start", docNodes, 0, -1)
3: node.route(Message, "start")
```

Αλγόριθμος 4.8 RecExpandState(String $stName$, Events $docNodes$, $currentIndex$, $parentIndex$)

```
1: st ← states.get(stName)
2: activeStates.add(st)
3: if l = 0 then
4:    f(l) ← 1
5: else
6:    f(l) ← l
7: end if
8: while elements processed < f(l) ∧ not processed all events in docNodes do
9:    event = docNodes.getNextEvent()
10:   if event.isEndElement() then
11:      break
12:   end if
13:   if event has siblings then
14:      for sibling in docNodes.siblingsOf(event) do
15:         targetStates.add(ExpandState(activeStates, sibling, l))
16:         check for satisfied queries in targetStates, notify subscribers
17:      end for
18:      break
19:   else
20:      targetStates ← ExpandState(activeStates, event, l)
21:      check for satisfied queries in targetStates, notify subscribers
22:      activeStates ← targetStates
23:   end if
24: end while
25: for i = 0, i < siblings.size() do
26:   //if no siblings found then siblings contains only event
27:   event ← siblings.get(i)
28:   nextEvent ← siblings.get(i + 1)
29:   if nextEvent.isEndElement() then
30:      continue //to next sibling
31:   end if
32:   for nextState in targetStates.get(i) do
33:      Message ← RecExpandState(nextState.Name(), docNodes, nextEvent, Event)
34:      node.route(Message, nextState.Name())
35:   end for
36: end for
```

στοιχείου. Αυτό γίνεται με κλήση στη συνάρτηση 4.3. Οι καταστάσεις στόχοι που ανακτώνται προσθέτονται στη στοίβα $targetStates$, και στη συνέχεια διακόπτεται ο βρόγχος αφού έχουμε βρει στοιχείο με αδέλφια. Αν το στοιχείο δεν έχει αδέλφια, τότε απλά ανακτούμε τις καταστάσεις στόχους από τις ενεργές καταστάσεις με βάση το στοιχείο ($event$). Αφού γίνει έλεγχος για επερωτήσεις οι οποίες ικανοποιούνται σε αυτές τις καταστάσεις, συνεχίζουμε στην επόμενη επανάληψη, έχοντας θέσει τις νέες ενεργές καταστάσεις.

Όταν ολοκληρωθούν οι επεκτάσεις καταστάσεων, θα πρέπει να προωθηθούν τα επόμενα μηνύματα στο μονοπάτι εκτέλεσης. Ο αλγόριθμος διατρέχει όλα τα αδέλφια κάποιου στοιχείου ($siblings$) τα οποία έχουν βρεθεί κατά την τελευταία επανάληψη (γιατί η επανάληψη διακόπτεται στη περίπτωση που συναντήσουμε αδέλφια κάποιου στοιχείου). Αν δεν συναντήσαμε κάποιο στοιχείο που να έχει αδέλφια σε αυτό το τρέξιμο, τότε απλά συνεχίζουμε με το τελευταίο στοιχείο το οποίο εξετάσαμε. Έστω $event$ το στοιχείο το οποίο εξετάζουμε, και $nextEvent$ το επόμενο στοιχείο στο ευρετήριο γεγονότων το οποίο διατρέχουμε. Αν το $nextEvent$ είναι γεγονός $EndOfElement$, τότε αυτό σημαίνει ότι έχουμε φτάσει στο τέλος του μονοπατιού εκτέλεσης και έτσι δεν θα προωθήσουμε κανένα μήνυμα. Διαφορετικά, για κάθε κατάσταση στις καταστάσεις στόχους (έστω st') που προήλθαν από τη τελευταία εκτέλεση της επανάληψης, δημιουργούμε ένα μήνυμα το οποίο θα προκαλέσει την εκτέλεση της συνάρτησης $RecExpandState$ στο κόμβο στον οποίο θα φτάσει, περνώντας σαν ορίσματα τη κατάσταση st', το ευρετήριο του εγγράφου, το $nextEvent$, το οποίο θα είναι το πρώτο στοιχείο που θα εξεταστεί στο κόμβο που αποτελεί το προορισμό της αποστολής, καθώς και το στοιχείο $Event$, το οποίο στην ουσία αποτελεί το γονιό αυτού του στοιχείου. Η εκτέλεση θα επαναληφθεί αναδρομικά στο κόμβο προορισμό, μέχρι την εξάντληση των μονοπατιών εκτέλεσης.

4.2.5 Παράμετρος l και Replication

Η παράμετρος l, μπορεί να χρησιμοποιηθεί και σαν παράγοντας αντιγραφής (replication factor): Η μέχρι τώρα περιγραφή μας, επικεντρωνόταν στη χρήση του l μόνο ως παράγοντα βελτίωσης της αποδοτικότητας του συστήματος. Όμως, από τη στιγμή που για $l > 0$, αποθηκεύουμε καταστάσεις σε περισσότερους του ενός κόμβους· κάτι τέτοιο σημαίνει ότι η ανάκτηση κάποιας κατάστασης, μπορεί να γίνει - αν αυτό δεν είναι για κάποιο λόγο δυνατό για τον υπεύθυνο κόμβο - από κάποιο άλλο κόμβο του δικτύου, συγκεκριμένα από τους κόμβους οι οποίοι είναι υπεύθυνοι για τις l τελευταίες καταστάσεις πριν από αυτή που ψάχνουμε, χωρίς να υπολογίζουμε

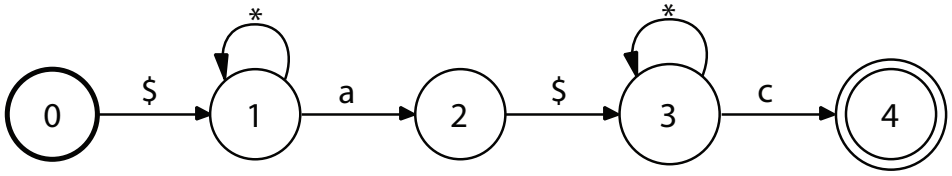
Αλγόριθμος 4.9 requestState$_{lmod}$(String stateName)

```
 1: Message msg ← request state with name stateName
 2: State st ← node.route(msg, stateName)
 3: if st = null then
 4:    //state with stateName should be stored in nodes responsible for l
 5:    //previous transitions (that led us to the state), without counting
 6:    //any ε transitions
 7:    reqFrom = getPreviousStateNameFrom(reqFrom)
 8:    counter ← 0,reqFrom ← stateName
 9:    repeat
10:       t is transition label
11:       if t != "$" then
12:          counter++
13:          if l = 0 then
14:             break;
15:          end if
16:       end if
17:       State st ← node.route(msg, reqFrom)
18:    until counter = l ∨ st != null
19: end if
20: return st
```

κενές μεταβάσεις.

Στον Αλγόριθμο 4.9, βλέπουμε τη διαδικασία χρήσης του l σαν παράγοντα αντιγραφής, έτσι ώστε να μπορούμε να ανακτούμε μια κατάσταση ενώ κάποιος κόμβος έχει αποτύχει να την επιστρέψει. Για να δούμε ένα παράδειγμα, μπορούμε να επικεντρωθούμε στο Σχήμα 4.1. Έστω ότι θέλουμε να ανακτήσουμε τη κατάσταση με όνομα "$start\$a\$$", τη κατάσταση 3 όπως παρουσιάζεται για συντομία στο σχήμα και έστω ότι η παράμετρος $l = 1$ στο σύστημα. Έτσι, σύμφωνα με τα προηγούμενα, θα μπορούμε να ανακτήσουμε τη κατάσταση από το κόμβο υπεύθυνο για τη κατάσταση "$start\$a$". Λόγω της κενής μετάβασης που ακολουθήσαμε (αντίστροφα) όμως για να φτάσουμε στη προηγούμενη κατάσταση, μπορούμε να ανακτήσουμε τη κατάσταση που ζητάμε και από τον κόμβο ο οποίος είναι υπεύθυνος για την "$start\$$".

Τέλος, αναφέρουμε δύο σημειώσεις: Πρώτα, την ιδιομορφία του $l = 0$, όπου σε αυτή τη περίπτωση κατάσταση αποθηκεύεται στο κόμβο που είναι υπεύθυνος για τη προηγούμενη μόνο αν μεταξύ τους μεσολαβεί ϵ μετάβαση. Το δεύτερο, αφορά το ότι σύμφωνα με τη σύνταξη της XPath και τους κανόνες μετατροπής της σε αυτόματο, δεν πρόκειται να συναντήσουμε περισ-



Σχήμα 4.1: Αυτόματο επερώτησης $//a//c$

σότερες της μίας συνεχόμενες κενές μεταβάσεις, αφού κάτι τέτοιο θα σήμαινε ότι έχουμε ένα άξονα "////", κάτι που είναι συντακτικά λανθασμένο.

4.3 Θέματα Υλοποίησης

Σε αυτή την ενότητα, θα αναφέρουμε κάποιες λεπτομέρειές της υλοποίησης που έχει γίνει. Αρχικά, θα θίξουμε το θέμα του τοπικού χώρου αποθήκευσης των κόμβων που συμμετέχουν στο δίκτυο. Όπως είναι φανερό από τη περιγραφή και τους αλγορίθμους, οι αλγόριθμοι είναι συγκεκριμένοι και αυστηροί με τον χειρισμό του τοπικού χώρου αποθήκευσης, έτσι ώστε να εκμεταλλεύονται πλήρως τις δυνατότητες του δικτύου. Όπως έχουμε αναφέρει το PAST προσφέρει τη χρήση των κόμβων που συμμετέχουν σε ένα δίκτυο Pastry σαν αποθηκευτικά μέσα, προσφέροντας τις μεθόδους $insert(data, key)$ και $lookup(key)$ για την αποθήκευση και ανάκτηση δεδομένων. Είναι λογικό, στο σύστημα που παρουσιάστηκε, να υποθέσει κάποιος ότι οι καταστάσεις του αυτόματου μπορούν να αποτελέσουν τα δεδομένα τα οποία θα εισάγουμε στο PAST, χρησιμοποιώντας τη συνάρτηση $insert(state, state.key)$. Όμως, κάτι τέτοιο στην ουσία θα "καταργούσε" το τοπικό χώρο αποθήκευσης, όπως αυτός αναφέρεται στους αλγορίθμους.

Σχήμα 4.2: Τμήμα αυτόματου επερώτησης

Για να δώσουμε ένα παράδειγμα, ας πάρουμε το Σχήμα 4.2. Στο σχήμα αυτό, παρουσιάζονται δύο καταστάσεις, τις οποίες ενώνει μια μετάβαση με ετικέτα dblp. Με τη χρήση του PAST και για $l = 0$, θα είχαμε τις κλήσεις insert(0,"$\$start$") και αντίστοιχα insert(1,"$\$startdblp$"). Η πρώτη κλήση, θα κατέληγε στο κόμβο του οποίου το αναγνωριστικό θα βρισκόταν αριθμητικά

πιο κοντά στο $hash(\text{"}\$start\text{"})$, ενώ η δεύτερη στο κόμβο που βρισκόταν αριθμητικά ποιο κοντά στο $hash(\text{"}\$startdblp\text{"})$. Τι γίνεται όμως αν είχαμε κενή μετάβαση αντί $dblp$ για ετικέτα· Στη περίπτωση αυτή, θα έπρεπε η κατάσταση 1 με κλειδί "$\$start\$$" να αποθηκευτεί στο κόμβο υπεύθυνο για τη κατάσταση 0, με κλειδί "$\$start$". Αυτό όμως δεν είναι[9] δυνατό. Το PAST θα αποθηκεύει πάντα τη κατάσταση στο κόμβο ο οποίος είναι υπεύθυνος για το κλειδί της. Το ίδιο πρόβλημα θα συνέβαινε αν $l > 0$. Στην ουσία, με τη χρήση του PAST με τέτοιες συνθήκες, είναι σαν να "αγνοούμε" τη παράμετρο l. Επομένως, στο σύστημα που υλοποιούμε, χρησιμοποιούμε απευθείας το δίκτυο επικάλυψης που προσφέρει το Pastry και το τοπικό χώρο αποθήκευσης στο κάθε κόμβο, χωρίς να χρησιμοποιούμε το PAST.

Επίσης, θεωρούμε ότι *μόνο* ένα νήμα (thread) σε κάθε κόμβο του δικτύου μπορεί να εκτελεί ευρετηριασμό κάποιας επερώτησης. Αυτό, προκύπτει για τη διατήρηση της συνέπειας και ορθότητας του δικτύου, αφού οι μέθοδοι ευρετηριασμού αλλάζουν τις δομές που αποθηκεύουν καταστάσεις και μεταβάσεις στο κατανεμημένο αυτόματο. Οι διαδικασίες εκτέλεσης του αυτόματου όμως, οι οποίες αφορούν την δημοσίευση, είτε αυτή είναι αναδρομική είτε επαναληπτική, λειτουργούν πολυνηματικά (multi-threading), με απεριόριστο αριθμό εξυπηρετήσεων ταυτόχρονων αιτήσεων δημοσίευσης (πάντα με όρια τις δυνατότητες του υπολογιστικού συστήματος που αντιστοιχεί στο κόμβο). Επίσης, για τις δοκιμές σε κάποιο πραγματικό κατανεμημένο περιβάλλον, έχει υλοποιηθεί εφαρμογή που μπορεί να λειτουργήσει ως πελάτης (client) έτσι ώστε να διευκολυνθεί η λήψη μετρήσεων. Αντίστοιχα, έχει υλοποιηθεί εφαρμογή πολύ-νηματικού εξυπηρετητή (server), ο οποίος μπορεί να εξυπηρετήσει πελάτες για το σκοπό αυτό.

4.4 Συμπεράσματα

Στο κεφάλαιο αυτό έχουμε περιγράψει την υλοποίηση του συστήματος [41] χρησιμοποιώντας το σύστημα FreePastry. Έχουμε περιγράψει τη διαδικασία κατασκευής του κατανεμημένου αυτόματου πάνω από το δίκτυο κάνοντας αναφορά στη παράμετρο l και τις διαδικασίες χειρισμού των τιμών που μπορεί να πάρει. Έχουμε αναφέρει αλγόριθμους που χρησιμοποιούνται γενικότερα κατά τη δημοσίευση, όπως τον αλγόριθμο επέκτασης κάποιων καταστάσεων με βάση κάποιο XML στοιχείο. Στη συνέχεια παρουσιάστηκαν οι αλγόριθμοι που αφορούσαν την επαναληπτική και την αναδρομική μεθοδολογία. Παρουσιάσαμε ένα αλγόριθμο που εκμετ-

[9] Η μόνη περίπτωση που μπορεί να αποθηκευτούν σε αυτή τη περίπτωση οι δύο καταστάσεις στον ίδιο κόμβο είναι αν το $hash(\text{"}\$start\$\text{"})$ είναι αριθμητικά πιο κοντά στο κόμβο ο οποίος είναι αριθμητικά πιο κοντά στο $hash(\text{"}\$start\$\text{"})$, αλλά κάτι τέτοιο δεν γενικεύεται.

αλλεύεται τη παράμετρο l για την ανάκτηση καταστάσεων σε περίπτωση αποτυχίας κάποιου κόμβου, ενώ τελικά παρουσιάστηκαν κάποια γενικά στοιχεία και λεπτομέρειες που αφορούσαν την υλοποίηση.

Κεφάλαιο 5

Επίλογος

Συνοψίζοντας, μέχρι αυτό το σημείο έχουμε παρουσιάσει το θεωρητικό υπόβαθρο της εργασίας, εξετάζοντας στενά ενότητες όπως δίκτυα ομότιμων, κατανεμημένους πίνακες κατακερματισμού (Chord, Pastry), τη γλώσσα XML και τη γλώσσα επερωτήσεων XPath. Επίσης τη θεωρία που σχετίζεται με τα μη ντετερμινιστικά πεπερασμένα αυτόματα καθώς και κάποια θεωρία των συστημάτων δημοσίευσης / συνδρομής. Στη συνέχεια, έχουμε περιγράψει το κεντρικοποιημένο σύστημα YFilter, περιγραφή η οποία μας επέτρεψε στη συνέχεια να περιγράψουμε το κατανεμημένο σύστημα, όπου βρίσκεται και η ουσία της εργασίας, ενώ τελικά, έχουμε περιγράψει με λεπτομέρεια την υλοποίηση καθώς και θέματα που σχετίζονται με αυτή. Σε αυτό το τελευταίο κεφάλαιο, θα παρουσιάσουμε κάποια στοιχεία που αφορούν τρόπους που δοκιμάστηκε η εφαρμογή και τα περιβάλλοντα στα οποία δοκιμάστηκε (Ενότητα 5.1), καθώς και κάποιες πιθανές επεκτάσεις που μπορούν να υλοποιηθούν (Ενότητα 5.2).

5.1 Δοκιμές και Πειράματα

Η υλοποίηση του συστήματος, αρχικά δοκιμάστηκε τοπικά, με τη δημιουργία πολλών κόμβων σε ένα virtual machine της Java, όπως αυτό προσφέρεται από τη βιβλιοθήκη του Pastry[1]. Στη συνέχεια, η υλοποίηση δοκιμάστηκε στο testbed Everlab[2], το οποίο είναι μια έκδοση του Planetlab[3]. Πειραματικές δοκιμές που αφορούν το σύστημα, έχουν ήδη αρχίσει στα testbeds που αναφέρθηκαν, αλλά δυστυχώς τη στιγμή που αυτή η εργασία ολοκληρώνεται, τα αποτελέσματα

[1] Multiple nodes in one virtual machine: `http://freepastry.rice.edu/FreePastry/tutorial/tut_multiple_nodes.html`

[2] `http://www.everlab.org`

[3] `http://www.planet-lab.org/`

δεν είναι έτοιμα ακόμη για παρουσίαση. Ο ενδιαφερόμενος, προς το παρόν μπορεί να καταφύγει στα αποτελέσματα προσομοίωσης του συστήματος στο [41].

5.2 Πιθανές Επεκτάσεις

Στη συνέχεια θα αναφερθούμε σε κάποιες πιθανές επεκτάσεις οι οποίες μπορούν να χτιστούν πάνω στην εφαρμογή η οποία είναι ήδη έτοιμη:

- Επέκταση στην επεξεργασία κατηγορημάτων XPath: Η παρούσα έκδοση της εφαρμογής αγνοεί τα κατηγορήματα των επερωτήσεων XPath. Αλγόριθμοι για την επεξεργασία κατηγορημάτων μπορούν να βρεθούν στο [20], ενώ αυτές οι προσεγγίσεις θα μπορούσαν να τροποποιηθούν με κάποιο τρόπο, έτσι ώστε να εκμεταλλεύονται τις ιδιότητες των κατανεμημένων δικτύων.

- Έχουμε ήδη αναφέρει γιατί δεν έχει χρησιμοποιηθεί το PAST στη παρούσα υλοποίηση (Ενότητα 4.3). Θα μπορούσαν αρχικά να εξερευνηθούν εναλλακτικές προσεγγίσεις, οι οποίες στην ουσία θα θεωρούν διαφορετικές δομές και θα άλλαζαν κάποια κομμάτια του αλγορίθμου, έτσι ώστε να δοκιμαστεί η επίδρασή του PAST στην απόδοση του συστήματος. Σημειώνεται ότι η επίδραση αυτή μπορεί να είναι και αρνητική, αφού το PAST εκτός από τις αυτοματοποιημένες διαδικασίες που προσφέρει, κάτι που είναι σίγουρα θετικό, υπερφορτώνει το σύστημα με πάρα πολλά νήματα. Επίσης, αρνητικό ρόλο μπορεί να διαδραματίσουν συμβιβασμοί που μπορεί να γίνουν στους αλγορίθμους έτσι ώστε να μπορούν να εφαρμοστούν στο PAST.

- Υποστήριξη μιας περισσότερο δυναμικής εικόνας δικτύου: Το Pastry φροντίζει να ειδοποιεί κόμβους κατά την άφιξη ή αποχώρηση κόμβων από το σύστημα. Σε αυτό το σημείο θα μπορούσε να βασιστεί η διατήρηση του αυτόματου κατά τις διαδοχικές δυναμικές αλλαγές του δικτύου, μεταφέροντας καταστάσεις από τις τοπικές δομές κάποιου κόμβου στις τοπικές δομές κάποιου άλλου. Σημειώνεται ότι μια τέτοια προσαρμογή γίνεται αυτόματα από το PAST, το οποίο όμως δεν μπορεί να χρησιμοποιηθεί για τοπικές δομές οι οποίες είναι αποθηκευμένες στο κόμβο, έτσι θα πρέπει να δημιουργηθούν προσαρμοσμένοι στο σύστημα, συγκεκριμένοι αλγόριθμοι.

- Υλοποίηση μεθόδων εξισορρόπησης φόρτου: Το Pastry, από μόνο του φροντίζει η κατανομή των δεδομένων (των καταστάσεων, στη περίπτωσή μας) να είναι ομοιόμορφη στο χώρο αναγνωριστικών του συστήματος, χρησιμοποιώντας τις κατάλληλες συναρτήσεις κατακερμα-

τισμού, κάτι που αφορά αντιμετώπιση στο πρώτο σκέλος του προβλήματος εξισορρόπησης φόρτου. Υπάρχει όμως το πρόβλημα της εξισορρόπησης φόρτου το οποίο αφορά κάποια δεδομένα στα οποία υπάρχει πολύ συχνή πρόσβαση. Ένα απλό παράδειγμα στη περίπτωση του συστήματος το οποίο εξετάζουμε, είναι ο κόμβος ο οποίος είναι υπεύθυνος για τη κατάσταση εκκίνησης (start). Σε κάθε ευρετηριασμό και δημοσίευση, ο κόμβος αυτός καλείται να επιστρέψει τη κατάσταση αυτή, οπότε είναι πολύ πιθανόν να υπερφορτωθεί. Στο Pastry, αυτού του είδους της εξισσορόπηση φόρτου προσφέρει το PAST, αλλά όπως έχουμε αναφέρει στην Ενότητα 4.3, κρίθηκε ότι το PAST δεν είναι κατάλληλο για τους αλγορίθμους του συστήματος. Έτσι, θα πρέπει να αναπτυχθούν ειδικοί αλγόριθμοι για την αντιμετώπιση του προβλήματος αυτού, που να είναι προσαρμοσμένοι πάνω στο σύστημα το ίδιο.

- Υλοποιήσεις του συστήματος με άλλα DHTs, όπως το Bamboo DHT[4]. Κάτι τέτοιο θα αύξανε τη εγκυρότητα της αξιολόγησης του συστήματος, εκμεταλλευόμενο διάφορα προτερήματα τα οποία μπορεί να υπάρχουν στις προσεγγίσεις άλλων συστημάτων. Μία τέτοια υλοποίηση θα μπορούσε να χρησιμοποιεί και κάποια άλλη γλώσσα προγραμματισμού, όπως τη Python[5].

[4] http://bamboo-dht.org/
[5] http://www.python.org/

Ορολογία

Ξενόγλωσσος Όρος	Ελληνικός Όρος
accept states	καταστάσεις αποδοχής
adjacent	παρακείμενα
attribute	γνώρισμα
bit	δυαδικό ψηφίο
blog	ιστολόγιο
broker	μεσίτης
browser	πλοηγητής
cache	κρυφή μνήμη
Centralized Directory Model	Μοντέλο Κεντρικού Καταλόγου
Chord	Σύστημα DHT
client	πελάτης
clockwise	φορά δεικτών ρολογιού
component	συνιστώσα
computer network	δίκτυο επικοινωνιών
concurrent	σύγχρονα
consume	κατανάλωση
content-based	βασισμένο στο περιεχόμενο
continuous query	συνεχής επερώτηση
counter clockwise	αντίστροφη φορά δεικτών ρολογιού
coupled	συζευγμένο
decoupling	αποσύνδεση
dial-up	τρόπος σύνδεσης
dissemination	διάχυση

Συνέχεια στην επόμενη σελίδα

Συνέχεια από τη προηγούμενη σελίδα

Ξενόγλωσσος Όρος	**Ελληνικός Όρος**
distance function	συνάρτηση απόστασης
distributed automaton	κατανεμημένο αυτόματο
Distributed Hash Table	Κατανεμημένοι Πίνακες Κατακερματισμού
Document Type Definition	ορισμός τύπου εγγράφου
domain name system	σύστημα ονομάτων τομέα
e-commerce	ηλεκτρονικό εμπόριο
e-transition	κενή μετάβαση
element	στοιχείο
event	γεγονός
event based parser	συντακτικός αναλυτής βασισμένος σε γεγονότα
Everlab	πλατφόρμα πειραματισμού ερευνητικών έργων
explicit	ρητός
extendable markup language	επεκτάσιμη γλώσσα σήμανσης
fault tolerance	ανοχή σε σφάλματα
finger table	πίνακας δεικτών
Finite State Machines	Μηχανές Πεπερασμένων Καταστάσεων
flooded requests model	μοντέλο πλημμύρας
FLOPS	πράξεις κινητής υποδιαστολής το δευτερόλεπτο
FreePastry	Υλοποίηση συστήματος Pastry
grid	πλέγμα
grid computing	υπολογιστικό πλέγμα
fragment	τμήμα
hierarchical p2p networks	ιεραρχικά δίκτυα ομότιμων
hybrid	υβριδικά
Hypertext Transfer Protocol	Πρωτόκολλο μεταφοράς υπερκειμένου
identifier	αναγνωριστικό
idle time	αδρανής χρόνος
internet	Διαδίκτυο
IP address	διεύθυνση Διαδικτύου

Συνέχεια στην επόμενη σελίδα

Συνέχεια από τη προηγούμενη σελίδα

Ξενόγλωσσος Όρος	**Ελληνικός Όρος**
iterative	επαναληπτικά
Java	γλώσσα προγραμματισμού
key	κλειδί
keywords	λέξεις κλειδιά
large scale	μεγάλης κλίμακας
leaf set	σύνολο φύλλων
load balancing	εξισορρόπηση φόρτου
lookup	αναζήτηση
meta-information	μετά-πληροφορίες
metamarkup languages	γλώσσες μετασήμανσης
neighborhood set	σύνολο γειτονίας
node arrival	άφιξη κόμβου
node departure	αναχώρηση κόμβου
node failure	αποχώρηση κόμβου
node recovery	επανάκαμψη κόμβου
Non Deterministic Finite Automata	Μη Ντετερμινιστικά Πεπερασμένα Αυτόματα
open source	ανοικτού κώδικα
overlay network	δίκτυο επικάλυψης
Pastry	Σύστημα DHT
pattern	μοτίβο DHT
peer exchange	ανταλλαγή ομότιμων κόμβων
peer node	κόμβος σε δίκτυο ομότιμων
peer-to-peer network	Δίκτυο Ομότιμων Κόμβων
Planetlab	πλατφόρμα πειραματισμού ερευνητικών έργων
predecessor	προκάτοχος, προηγούμενος κόμβος
predicate	κατηγόρημα
processing load	φόρτος εργασίας
profile	προφίλ

Συνέχεια στην επόμενη σελίδα

Συνέχεια από τη προηγούμενη σελίδα

Ξενόγλωσσος Όρος	**Ελληνικός Όρος**
proximity formula	φόρμουλα εγγύτητας
proximity metric	μετρική εγγύτητας
publish	δημοσίευση
publish / subscribe	δημοσίευση συνδρομή
publisher	εκδότης
pure	καθαρά
Python	γλώσσα προγραμματισμού
query	επερώτηση
regular	κανονική
replicas	αντίγραφα
ring	δακτύλιος
robust	εύρωστο
routing tables	πίνακες δρομολόγησης
scalability	κλιμάκωση
selection postponed	αναβολή εκτέλεσης
self child	κατάσταση με βρόγχο
serializable	σειριοποιήσιμος
server	εξυπηρετητής
sibling	αδερφό
Single Point of Failure	μοναδικό σημείο αποτυχίας
space decoupling	αποσύνδεση χώρου
state	κατάσταση
stabilization algorithm	αλγόριθμος σταθεροποίησης
start state	κατάσταση εκκίνησης
string	συμβολοσειρά
structured lookup	δομημένη αναζήτηση
subscriber	συνδρομητής
successor node	διάδοχος κόμβος
super node	υπέρ κόμβος
symbol	σύμβολο

Συνέχεια στην επόμενη σελίδα

Συνέχεια από τη προηγούμενη σελίδα

Ξενόγλωσσος Όρος	**Ελληνικός Όρος**
synchronization decoupling	Αποσύνδεση συγχρονισμού
tag	ετικέτα
testbed	πλατφόρμα πειραματισμού ερευνητικών έργων
telnet	πρωτόκολλο δικτύου
Theory of Computation	Θεωρία Υπολογισμού
thread	νήμα
time decoupling	Αποσύνδεση χρόνου
time to live	χρόνος ζωής
tracker	εξυπηρετητής torrent
unstructured routing tables	αδόμητοι πίνακες δρομολόγησης
valid	έγκυρο
well defined	καλά ορισμένο
wildcard	μπαλαντέρ
YFilter	σύστημα XML φιλτραρίσματος

Συντμήσεις-Αρκτικόλεξα

Σύντμηση	Πλήρης Ανάπτυξη
CCW	Counter-Clockwise
CW	Clockwise
DHT	Distributed Hash Table
DNS	Domain Name System
DTD	Document Type Definition
e-commerce	electronic commerce
e-transition	epsilon (ϵ) transition
ΕΚΠΑ	Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
FSM	Finite State Machines
FLOPS	Floating Point Operations per Second
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
IP-Address	Internet Protocol Address
ΚΠΚ	Κατανεμημένοι Πίνακες Κατακερματισμού
p2p	peer-to-peer
pub / sub	publish / subscribe
RSS	Real Simple Syndication
SHA-1	Secure Hash Algorithm-1
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
Telnet	TELecommuncations NETwork
TOC	Theory of Computation

Συνέχεια στην επόμενη σελίδα

Συνέχεια από τη προηγούμενη σελίδα

Σύντμηση	**Πλήρης Ανάπτυξη**
TTL	Time To Live
VoIP	Voice Over IP
WML	Wireless Markup Language
WWW	World Wide Web
XML	Extendable Markup Language
XPath	XML Path Language
XQuery	XML Query Language
XUL	XML User Interface Language

Βιβλιογραφία

[1] K. Aberer and M. Hauswirth. An overview of peer-topeer information systems, *WDAS*, 2002.

[2] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: a self-organizing structured P2P system. *SIGMOD Rec.*, 32(3):29–33, 2003.

[3] D. Aksoy, M. Altinel, R. Bose, U. Çetintemel, M. J. Franklin, J. Wang, and S. B. Zdonik. Research in data broadcast and dissemination. In *AMCP '98: Proceedings of the First International Conference on Advanced Multimedia Content Processing*, pages 194–207, London, UK, 1999. Springer-Verlag.

[4] M. Altinel, D. Aksoy, T. Baby, M. Franklin, W. Shapiro, and S. Zdonik. Dbis toolkit: Adaptable middleware for large scale data delivery. In *Proc. ACM SIGMOD Conf.*, Philadelphia, PA, June 1999.

[5] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 53–64, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[6] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K. Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 53–64. Morgan Kaufmann, 2000.

[7] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in p2p systems. *Commun. ACM*, 46(2):43–48, 2003.

[8] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. *SIGCOMM Comput. Commun. Rev.*, 34(4):353–366, 2004.

[9] N. Bruno, L. Gravano, N. Koudas, and D. Srivastava. Navigation- vs. index-based XML multi-query processing. *ICDE*, 2003.

[10] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Symposium on Principles of Distributed Computing*, pages 219–227, 2000.

[11] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Topologyaware routing in structured peer-to-peer overlay networks. Lecture notes in computer science, Springer Berlin / Heidelberg, 2002.

[12] C. Y. Chan, P. Felber, M.s N. Garofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. In *ICDE*, 2002.

[13] C. Y. Chan and Y. Ni. Efficient XML data dissemination with piggybacking. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 737–748, New York, NY, USA, 2007. ACM.

[14] R. Chand and P. Felber. A scalable protocol for content-based routing in overlay networks *Technical Report RR-03-074, Institut EURECOM*, 2003.

[15] M. P. Consens and T. Milo. Optimizing queries on files. In *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 301–312, New York, NY, USA, 1994. ACM.

[16] World Wide Web Consortium. XML path language (*XPath*), 1999.

[17] World Wide Web Consortium. The XML data model, 2005.

[18] World Wide Web Consortium. Extensible markup language (XML) 1.0 (fourth edition), 2006.

[19] Y. Diao, M. Altinel, M. J. Franklin, H. Zhang, and P. Fischer. Path sharing and predicate evaluation for high-performance XML filtering. *ACM Trans. Database Syst.*, 28(4):467–516, 2003.

[20] Y. Diao and M. Franklin. High-performance XML filtering: An overview of *YFilter*, *ICDE*, 2003.

[21] Y. Diao, S. Rizvi, and M. J. Franklin. Towards an internet-scale XML dissemination service. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 612–623. VLDB Endowment, 2004.

[22] I. Dimov, J. Dongarra, K. Madseni, J. Wasniewski, and Z. Zlatev. *Application of Distributed and Grid Computing*. FGCS Future Generation Computer Systems. Elsevier Science North-Holland Publishing Company, 2007.

[23] P. Th. Eugster, P. A. Felber, R. G., and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[24] P. Felber, C. Chan, M. Garofalakis, and R. Rastogi. Scalable filtering of XML data for web services. *IEEE Internet Computing*, 7(1):49–57, 2003.

[25] E. Freeman, K. Arnold, and S. Hupfer. *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley Longman Ltd., Essex, UK, UK, 1999.

[26] X. Gong, W. Qian, Y. Yan, and A. Zhou. Bloom filter-based XML packets filtering for millions of path queries. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, pages 890–901, Washington, DC, USA, 2005. IEEE Computer Society.

[27] T. J. Green, A. Gupta, G. Miklau, M. Onizuka, and D. Suciu. Processing XML streams with deterministic automata and stream indexes. *ACM Trans. Database Syst.*, 29(4):752–788, 2004.

[28] Χ. Δουλιγιέρης, Ρ. Μαυροπόδη, Ε. Κοπανάκη. *Τεχνολογίες Διαδικτύου, Αρχές Λειτουργίας και Προγραμματισμός Εφαρμογών στο Διαδίκτυο*. Εκδόσεις Νηρηίδες, 2006.

[29] S. Guha, N. Daswani, and R. Jain. An experimental study of the skype peer-to-peer voip system, in IPTPS, 2006.

[30] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi. Meghdoot: content-based publish/subscribe over p2p networks. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 254–273, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[31] A. K. Gupta and D. Suciu. Stream processing of xpath queries with predicates. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 419–430, New York, NY, USA, 2003. ACM.

[32] E. R. Harold and W. S. Means. *XML in a Nutshell, Third Edition*. O'Reilly Media, Inc., 2004.

[33] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed object location in a dynamic network. In *Proceedings of the Fourteenth ACM Symposium on Parallel Algorithms and Architectures*, 2002.

[34] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*. Addison Wesley, November 2000.

[35] S. Hou and H. Jacobsen. Predicate-based filtering of xpath expressions. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 53, Washington, DC, USA, 2006. IEEE Computer Society.

[36] Y. Huang and H. Garcia-Molina. Publish/subscribe in a mobile environment. *Wirel. Netw.*, 10(6):643–652, 2004.

[37] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, May 1997.

[38] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st annual ACM symposium on Principles of distributed computing*. ACM Press, 2002.

[39] G. S. Manku. *Dipsea: A Modular Distributed Hash Table*. PhD dissertation, Stanford University, Department of Computer Science, August 2004.

[40] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric, in proceedings of IPTPS02, 2002.

[41] I. Miliaraki, Z. Kaoudi, and M. Koubarakis. XML Data Dissemination using Automata on Top of Structured Overlay Networks. In *Proceedings of the 17th International World Wide Web Conference (WWW2008)*, Beijing, China, April 2008.

[42] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing.

[43] R. Monson-Haefel and D. Chappell. *Java Message Service*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2000.

[44] M. Hedlund et al. N. Minar. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. pub-ORA, pub-ORA, March 2001. Edited by Andrew Oram.

[45] Moni Naor and Udi Wieder. Novel architectures for p2p applications: The continuous-discrete approach. *ACM Trans. Algorithms*, 3(3):34, 2007.

[46] CORPORATE National Institute of Standards and Technology. Federal information processing standards publication 180 (1993 may 11): specifications for the secure hash standard (shs). pages 87–92, 1995.

[47] B. Ozen, O. Kilic, M. Altinel, and A. Dogac. Highly personalized information delivery to mobile clients. *Wirel. Netw.*, 10(6):665–683, 2004.

[48] F. Peng and S. S. Chawathe. *XPath* queries on streaming data. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 431–442, New York, NY, USA, 2003. ACM.

[49] D. Powell. Group communication. *Commun. ACM*, 39(4):50–53, 1996.

[50] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, volume 31, pages 161–172. ACM Press, October 2001.

[51] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. OpenDHT: a public DHT service and its uses. *SIGCOMM Comput. Commun. Rev.*, 35(4):73–84, 2005.

[52] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–??, 2001.

[53] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. *SIGOPS Oper. Syst. Rev.*, 35(5):188–201, 2001.

[54] A. C. Snoeren, K. Conley, and D. K. Gifford. Mesh-based content routing using XML. *SIGOPS Oper. Syst. Rev.*, 35(5):160–173, 2001.

[55] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[56] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *IEEE Transactions on Networking*, 11, February 2003.

[57] A. Tanenbaum. *Computer Networks*. Prentice Hall, 2002.

[58] A. S. Tanenbaum and M. V. Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2001.

[59] C. Tryfonopoulos, S. Idreos, and M. Koubarakis. Publish/subscribe functionality in ir environments using structured overlay networks. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 322–329, New York, NY, USA, 2005. ACM.

[60] H. Uchiyama, M. Onizuka, and T. Honishi. Distributed XML stream filtering system with high scalability. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, pages 968–977, Washington, DC, USA, 2005. IEEE Computer Society.