Πανεπιστήμιο Κρήτης
Σχολή Θετικών Επιστημών
Τμήμα Επιστήμης Υπολογιστών

# Σχεδίαση & Υλοποίηση Μεθόδων για τη Βελτίωση της Απόδοσης των WWW Proxies

**Μιχαήλ Δ. Φλουρής**

Μεταπτυχιακή Εργασία

Ηράκλειο, Φεβρουάριος 2000

Πανεπιστήμιο Κρήτης
Σχολή Θετικών Επιστημών
Τμήμα Επιστήμης Υπολογιστών

# Σχεδίαση & Υλοποίηση Μεθόδων για τη Βελτίωση της Απόδοσης των WWW Proxies

Εργασία που υποβλήθηκε από τον
Μιχαήλ Δ. Φλουρή
ως μερική εκπλήρωση των απαιτήσεων για την απόκτηση
ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ

Συγγραφέας:

Μιχαήλ Δ. Φλουρής
Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

Εισηγητική Επιτροπή:

Ευάγγελος Π. Μαρκάτος,
Επίκουρος Καθηγητής, Επόπτης

Μανόλης Κατεβαίνης,
Καθηγητής, Μέλος

Απόστολος Τραγανίτης,
Αναπληρωτής Καθηγητής, Μέλος

Δεκτή:

Πάνος Κωνσταντόπουλος, Καθηγητής
Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών

Φεβρουάριος 2000

*Στην οικογένειά μου*

# Σχεδίαση & Υλοποίηση Μεθόδων για τη Βελτίωση της Απόδοσης των WWW Proxies

Μιχαήλ Δ. Φλουρής

Μεταπτυχιακή Εργασία

Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

## Περίληψη

Το πρόβλημα ποιότητας υπηρεσίας (QoS) του Παγκόσμιου Ιστού, αλλά και η προστασία των τοπικών δικτύων πίσω από συστήματα firewall, οδηγούν όλο και περισσότερο προς τη χρήση WWW Proxies (ή WWW Caches), που αποτελεί ουσιαστικά τη μόνη μακροπρόθεσμη λύση. Ωστόσο, ο εκθετικός ρυθμός ανάπτυξης του Παγκόσμιου Ιστού δημιουργεί την ανάγκη για WWW Proxies με αυξανόμενα υψηλές επιδόσεις και ολοένα μεγαλύτερα συστήματα αποθήκευσης αντικειμένων WWW, τόσο σε κύρια (δυναμική) μνήμη όσο και σε συστήματα μαγνητικών δίσκων. Σε αντίθεση με τις απαιτήσεις αυτές, πρόσφατες ερευνητικές εργασίες έχουν δείξει ότι υπάρχουν σημαντικά προβλήματα στην απόδοση των WWW Proxies, τα οποία οφείλονται σε μεγάλο βαθμό στην αποθήκευση αντικειμένων σε συστήματα μαγνητικών δίσκων.

Στην εργασία αυτή μελετάται αρχικά το πρόβλημα της αποθήκευσης αντικειμένων στους δίσκους ενός WWW Proxy και αναλύονται αλγόριθμοι που έχουν προταθεί για τη λύση του. Εντοπίζοντας σε πρώτη φάση τα προβλήματα από τη δημιουργία και τη διαγραφή πολλών μικρών αρχείων και σε δεύτερη φάση από τη μη σειριακή προσπέλαση δεδομένων στο δίσκο, οι αλγόριθμοι αυτοί καταφέρνουν να βελτιώσουν αρκετές φορές την ταχύτητα αποθήκευσης αντικειμένων στο σύστημα δίσκων.

Η εργασία αυτή εστιάζει κυρίως στην υλοποίηση των αλγορίθμων αποθήκευσης αντικειμένων σε ένα πραγματικό WWW Proxy με σκοπό την πειραματική αποτίμησή τους. Μέσα στα πλαίσια αυτά, αναπτύχθηκε από μηδενική βάση ένα σύστημα WWW Proxy με βασικό στόχο τις υψηλές επιδόσεις σε ένα παραδοσιακό λειτουργικό σύστημα Unix και το γενικού σκοπού σύστημα αρχείων που το συνοδεύει. Η αρχιτεκτονική της υλοποίησής μας χρησιμοποιεί τις ασύγχρονες λειτουργίες των TCP/IP sockets και μία μηχανή πεπερασμένων καταστάσεων (FSM), για την παράλληλη εξυπηρέτηση πολλών αιτήσεων πελατών WWW.

Η αποτίμηση του WWW Proxy που υλοποιήθηκε έγινε σε πραγματικές συνθήκες με τη βοήθεια μετροπρογραμμάτων όπως το Web Polygraph που αποτελούν εκτων-πραγμάτων πρότυπο (defacto standard) για μετρήσεις σύγχρονων εμπορικών συστημάτων WWW Proxies. Οι μετρήσεις αυτές απέδειξαν ότι η υλοποίησή μας έχει περισσότερο από τρεις φορές υψηλότερο μέσο ρυθμό εξυπηρέτησης αιτήσεων WWW, συγκρινόμενη με το Squid που αποτελεί το δημοφιλέστερο και πιο εξελιγμένο Web Proxy ανοιχτού κώδικα σήμερα.

Επόπτης:

Ευάγγελος Π. Μαρκάτος
Επίκουρος Καθηγητής
Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

# Design & Implementation of Methods Enchancing the Performance of WWW Proxies

Michail D. Flouris

Master of Science Thesis

Computer Science Department
University of Crete

**Abstract**

The quality of service (QoS) problem of the World Wide Web and the protection of intranets behind firewalls, lead to the increasing use of WWW Proxies (or WWW Caches) as the only long-term solution. However, due to the exponential growth of the Web, there is an increasing demand for higher performance WWW Proxies with larger WWW object caches, not only in main (dynamic) memory but also on magnetic disks. Contrary to this demand, several research studies have proved, not only that modern WWW proxy systems suffer severe performance problems, but also that the main cause of these performance problems is the magnetic disk I/O induced by the WWW object storage.

In this Master's Thesis, we initially study the problem of WWW object storage in a WWW proxy's disks and analyse algorithms proposed for its solution. Resolving at first the problems caused by the creation and deletion of many small files, and secondly by the non-sequential disk accesses, these algorithms manage to enhance the throughput of storing WWW objects on disks by several times.

The main focus of this Thesis is the implementation of these algorithms for WWW object disk storage on a real WWW proxy, in order to evaluate them in a real-world situation. Accordingly, a WWW proxy system was developed from scratch, implemented with respect to the highest possible performance on a traditional Unix Operating System and the general-purpose Filesystem that accompanies it. The architecture of our prototype uses the asynchronous operations of TCP/IP sockets and a finite state machine (FSM), in order to process several WWW client requests in parallel.

The evaluation of our WWW Proxy prototype was conducted in a real-world setting using sophisticated WWW proxy benchmarks, like the Web Polygraph which is considered a de-facto standard for measuring the performance of commercial WWW proxies. Our experiments prove that our prototype has an average request service rate (throughput) more than three times higher than Squid, which is considered the most popular and advanced open source WWW proxy today.

Supervisor:

Evangelos P. Markatos
Associate Professor
Computer Science Department
University of Crete

# Ευχαριστίες

Στο σημείο αυτό, θα ήθελα κατ' αρχάς να ευχαριστήσω τον επόπτη καθηγητή μου κ. Ευάγγελο Μαρκάτο, για την πολύτιμη καθοδήγηση που μου παρείχε κατά τη διάρκεια των μεταπτυχιακών μου σπουδών και την καθοριστική του συμμετοχή στην επιστημονική μου πρόοδο. Τον ευχαριστώ επίσης για τις διαφωτιστικές υποδείξεις και την εύστοχη κριτική του κατά τη διάρκεια των συζητήσεών μας, κάτι που με έμαθε να σκέφτομαι πιο δημιουργικά και να βελτιωθώ σε ερευνητικό επίπεδο.

Επίσης, ευχαριστώ τα μέλη της εισηγητικής επιτροπής κ. Μανόλη Κατεβαίνη και κ. Απόστολο Τραγανίτη για τις εποικοδομητικές τους παρατηρήσεις και διορθώσεις στην εργασία αυτή καθώς και κατά τη διάρκεια της παρουσίασής μου.

Ευχαριστώ επίσης το Τμήμα Επιστήμης Υπολογιστών του Πανεπιστημίου Κρήτης για το περιβάλλον εργασίας που μου παρείχε και τις γνώσεις που μου έδωσε την ευκαιρία να αποκτήσω, καθώς και το Ινστιτούτο Πληροφορικής του Ιδρύματος Τεχνολογίας και Έρευνας, και ειδικότερα την Ομάδα Αρχιτεκτονικής και Ψηφιακών Συστημάτων, για την υλικοτεχνική υποδομή που μου παρείχε κατά τη διάρκεια των σπουδών μου και της ερευνητικής μου προσπάθειας.

Θα ήταν μεγάλη παράλειψη να μην ευχαριστήσω όλους ανεξαιρέτως τους φίλους και συμφοιτητές μου τα χρόνια των σπουδών μου. Στην πορεία μου μέχρι σήμερα μοιράστηκα ποικίλες εμπειρίες μαζί τους, που έκαναν το ταξίδι μου όχι μόνο ενδιαφέρον αλλά και εφικτό. Οι εμπειρίες αυτές, ευχάριστες αλλά και δυσάρεστες, έχουν διαμορφώσει κατά ένα μεγάλο μέρος το χαρακτήρα και τις απόψεις μου και θα μείνουν για πάντα χαραγμένες στην ψυχή μου.

Το μεγαλύτερο ευχαριστώ όμως, θέλω να το πω στην οικογένειά μου που με στήριξε συναισθηματικά και υλικά σε όλη την πορεία μου μέχρι σήμερα. Είμαι σίγουρος ότι χωρίς την υποστήριξη και την αγάπη τους δεν θα είχα φτάσει ως εδώ. Ως ελάχιστη λοιπόν ένδειξη σεβασμού για την αγάπη, την προσφορά και τις θυσίες της οικογένειάς μου, τους αφιερώνω την εργασία αυτή.

Μιχαήλ Δ. Φλουρής

# Περιεχόμενα

# Κατάλογος Σχημάτων

# Κατάλογος Πινάκων

# Εισαγωγή

There's no place like http://www.home.com
—Anon.

## 1.1 Ο ΠΑΓΚΟΣΜΙΟΣ ΙΣΤΟΣ (WWW)

Η υπηρεσία του Παγκόσμιου Ιστού (πιο γνωστή σε όλους μας ως World Wide Web ή WWW)[1] είναι σήμερα η πιο δημοφιλής υπηρεσία του Διαδικτύου (Internet). Είναι μάλιστα τόσο δημοφιλής στους χρήστες του Διαδικτύου, ώστε για πολλούς από τους αρχάριους χρήστες ο όρος Διαδίκτυο είναι συνώνυμος του Παγκόσμιου Ιστού.

Η ιδέα για την υπηρεσία του Παγκόσμιου Ιστού ξεκίνησε στα μέσα του 20ού αιώνα. Από το 1945 ο Vannevar Bush, επιστημονικός υπεύθυνος του προέδρου Roosevelt, γράφει στο περιοδικό Atlantic Monthly για το Memex, μία συσκευή βασισμένη σε μικροφίλμ για την αποθήκευση μεγάλου αριθμού εγγράφων στο χώρο ενός μόνο γραφείου, με μηχανική βοήθεια για την αναζήτηση, οργάνωση και προσθήκη αυτού του αρχείου [14]. Μέσα στη δεκαετία του 1960 ο Douglas Engelbart κατασκευάζει το πρώτο σύστημα *hypertext* [2], το *"oNLine System" (NLS)*, το οποίο τρέχει στις πολύ ακριβές και τεραστίου μεγέθους μηχανές της δεκαετίας αυτής [3] [37], [97].

Ο όρος "hypertext" αναφέρεται πρώτη φορά το 1968 από τον Ted Nelson [37]. Το hypertext είναι ένα κομμάτι κειμένου στο οποίο μερικές λέξεις ή φράσεις είναι *"σύνδεσμοι" ("hyperlinks")* προς άλλα κείμενα ή άλλες μορφές πληροφορίας όπως εικόνες, ήχοι κτλ. με σχετικό περιεχόμενο. Ένα έγγραφο σε μορφή hypertext ονομάζεται hypertext document. Έτσι ένα σύστημα hypertext είναι ουσιαστικά ένα δίκτυο από hypertext documents εννοιολογικά συνδεδεμένα. Μπορούμε να πούμε ότι ο Παγκόσμιος Ιστός είναι ένας τεράστιος αριθμός από *hypertext documents* (γνωστά και ως

[1] Κατά τη διάρκεια αυτής της εργασίας θα χρησιμοποιούμε τόσο τον όρο Παγκόσμιος Ιστός, όσο και τους όρους World Wide Web ή WWW ή απλά Web.

[2] Η Ελληνική απόδοση του όρου είναι "υπερκείμενο".

[3] Ο Engelbart είναι επίσης και ο εφευρέτης της συσκευής του ποντικιού το οποίο ανακάλυψε για χρήση στο σύστημα hypertext.

*WWW documents, WWW pages ή ιστοσελίδες*).

Το 1989 ο Tim Berners-Lee γράφει στο ερευνητικό κέντρο CERN στην Ελβετία την πρόταση “Information Management: A Proposal” η οποία περιγράφει ένα σύστημα διαχείρισης πληροφοριών το οποίο βασίζεται σε hypertext. Έτσι ένα χρόνο αργότερα, το 1990, ο ίδιος αρχίζει την κατασκευή ενός παγκόσμιου συστήματος hypertext και του πρώτου προγράμματος για την ανάγνωση hypertext το οποίο ονόμασε “WorldWideWeb”, όπως δηλαδή ονομάζουμε σήμερα τον *Παγκόσμιο Ιστό* [37], [97]. Η υπηρεσία του Παγκόσμιου Ιστού ξεκίνησε να λειτουργεί το 1991 [108].

Σχεδόν 10 χρόνια μετά η μορφή του Παγκόσμιου Ιστού έχει αλλάξει εντελώς. Η αρχική μορφή του έχει εμπλουτιστεί και επεκταθεί ώστε να μπορέσει να αποτελέσει μια κοινή πλατφόρμα ανταλλαγής και ανάκτησης κάθε είδους πληροφοριών. Η γλώσσα *HTML (HyperText Markup Language)* [100] που χρησιμοποιείται για την παράσταση των πληροφοριών στον Παγκόσμιο Ιστό, αφού έχει υποστεί πολλές διορθώσεις και επεκτάσεις μέχρι την σημερινή έκδοση 4.0, σύντομα θα αντικατασταθεί από τη διάδοχό της, την *XML (eXtended Markup Language)* [98], η οποία θα δώσει στο WWW ακόμη περισσότερες δυνατότητες στην εννοιολογική οργάνωση και παρουσίαση των πληροφοριών.

Σήμερα, εκτός από το κείμενο, χρησιμοποιούμε το WWW για να μεταφέρουμε κάθε είδους ψηφιακή πληροφορία. Η ιδέα του hypertext έχει μεταφερθεί σχεδόν σε όλα τα έγγραφα που γράφουμε σήμερα, αλλά ακόμα και στην οργάνωση της πληροφορίας μέσα στον υπολογιστή μας. Τα σημερινά λειτουργικά συστήματα παραθέτουν ανάμεσα στα προτερήματά τους το γεγονός ότι μπορούν να μας παρουσιάζουν κάθε είδους πληροφορία για τον υπολογιστή μας σε μορφή hypertext η οποία είναι απλή και κατανοητή ακόμα και στους πιο αρχάριους χρήστες.

Ο Παγκόσμιος Ιστός αποτελεί επίσης την ιδανική βάση για το *Ηλεκτρονικό Εμπόριο (Electronic Commerce)* το οποίο όπως είναι κατανοητό σχετίζεται άμεσα με την παγκόσμια οικονομική ανάπτυξη και την παγκοσμιοποίηση του εμπορίου και του ανταγωνισμού μεταξύ ατόμων, εταιρειών και κρατών. Είναι γνωστό ότι ο Παγκόσμιος Ιστός είχε το μεγαλύτερο βαθμό διείσδυσης στους καταναλωτές των ΗΠΑ από οποιαδήποτε άλλη τεχνολογία μέχρι σήμερα συμπεριλαμβανομένου του τηλεφώνου, της τηλεόρασης και του VCR [42]. Είναι λοιπόν κατανοητό ότι για κάθε είδους υπηρεσία του Διαδικτύου, υπάρχουσα ή μελλοντική, ένας κρίσιμος συντελεστής για την επιτυχία της είναι αν μπορεί να λειτουργήσει μέσω του Παγκόσμιου Ιστού.

Όλα αυτά δείχνουν τη σημασία του Παγκόσμιου Ιστού στη σημερινή παγκόσμια “ηλεκτρονική κοινότητα”. Στις επόμενες παραγράφους θα δούμε το ρυθμό αύξησης της Διαδικτυακής κίνησης WWW και τα προβλήματα που αυτός ο ρυθμός έχει εναποθέσει στους ώμους των επιστημόνων πληροφορικής. Επίσης θα κάνουμε μια εισαγωγή στα προβλήματα που εστιάζει η παρούσα μεταπτυχιακή εργασία και θα αναφέρουμε τη συνεισφορά μας στη μελέτη και λύση τους.

### 1.1.1 Ο ΡΥΘΜΟΣ ΑΥΞΗΣΗΣ ΤΟΥ WWW

Όλες οι ερευνητικές ομάδες που έχουν μελετήσει την κίνηση δεδομένων (data traffic ή απλώς traffic) του Παγκόσμιου Ιστού αναφέρουν ότι παρουσιάζει εκθετική αύξηση [28], [39], [40], [50], [53] από την έναρξη της λειτουργία της υπηρεσίας αυτής [54], [56], [11], [74].

Μερικά εντυπωσιακά στοιχεία του μεγάλου ρυθμού αύξησης του Παγκόσμιου Ιστού αναφέρουμε παρακάτω:

◇ Μόνο το 1993 όταν δημιουργήθηκε ο πρώτος εξελιγμένος WWW browser, ο NCSA Mosaic ο ετήσιος ρυθμός αύξησης της υπηρεσίας WWW ήταν 341.634% το χρόνο [108].

◇ Πιο πρόσφατα στατιστικά δείχνουν ότι η κίνηση του Παγκόσμιου Ιστού αποτελούσε το 40% της συνολικής Διαδικτυακής κίνησης το 1996, ενώ το 1994 ήταν μόλις το 19% [15].

◇ Υπολογίζεται ότι μέχρι το 2001 ο αριθμός των χρηστών του Παγκόσμιου Ιστού θα είναι μεγαλύτερος των 175 εκατομμυρίων παγκοσμίως [42].

◇ Ένας μεγάλος *Παροχέας Υπηρεσιών Διαδικτύου (Internet Service Provider ή ISP)* υπολογίζει ότι η δικτυακή κίνηση διπλασιάζεται κάθε 90 μέρες καθώς αυξάνεται το πλήθος των γραφικών, του ήχου και της κινούμενης εικόνας (animation ή video) [42].

◇ Σήμερα υπολογίζεται ότι υπάρχουν πάνω από ένα δισεκατομμύριο αντικειμένων στον Παγκόσμιο Ιστό και πάνω από 6.5 εκατομμύρια Εξυπηρέτες WWW στο Διαδίκτυο [25], [46].

◇ Σύμφωνα με πηγές [74] που όπως υποστηρίζουν παρακολουθούν δυναμικά το μέγεθος του Παγκόσμιου Ιστού, στις 7 Φεβρουαρίου 2000 υπήρχαν σε αυτόν 1.6 δισεκατομμύρια αντικείμενα, 361 εκατομμύρια στατικές εικόνες, 30.1 TerraBytes ($10^{12}$ bytes) κειμένου και 6 TerraBytes στατικές εικόνες. Μόνο το 24ωρο από τις 6 στις 7 Φεβρουαρίου 2000 προστέθηκαν στον Παγκόσμιο Ιστό 3.26 εκατομμύρια νέα αντικείμενα, 734 χιλιάδες νέες στατικές εικόνες, 61.2 GigaBytes ($10^9$ bytes) κείμενο και 12.2 GigaBytes στατικές εικόνες[4].

[4]Όπως παρατηρούμε ο ρυθμός που αυξάνεται ο όγκος νέων στατικών εικόνων είναι μεγαλύτερος από αυτόν του νέου κειμένου [74]

Περισσότερες πληροφορίες για το μέγεθος και το ρυθμό αύξησης του Παγκόσμιου Ιστού υπάρχουν στα [87], [74].

## 1.2 ΟΡΙΣΜΟΣ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ

Με την εκθετική ανάπτυξη του Παγκοσμίου Ιστού όπως αναφέραμε στην παράγραφο 1.1.1 ο φόρτος που επωμίζονται οι διαδικτυακοί πόροι είναι τεράστιος και συνεχώς αυξάνει. Αυτό έχει ως συνέπεια την αύξηση του χρόνου που χρειάζεται για να δούμε μία ιστοσελίδα. Ειδικά οι δημοφιλείς ιστοσελίδες χρειάζονται δεκάδες δευτερόλεπτα για να φτάσουν στην οθόνη του χρήστη. Αυτός ο χρόνος είναι πολύ μακριά από το στόχο που έχει θέσει η μελέτη της επικοινωνίας ανθρώπου - μηχανής και είναι τα 200 χιλιοστά του δευτερολέπτου (msec). Σύμφωνα με αυτές τις μελέτες αν ένα πρόγραμμα δεν αντιδρά μέσα σε 200 msec φαίνεται αργό στον χρήστη. Πολύ παραπάνω από αυτό το χρόνο είναι εκνευριστικό [43].

Σύμφωνα με μελέτες του Graphics, Visualization and Usability (GVU) Center που γίνονται από το 1994, το πρώτο πρόβλημα του Παγκόσμιου Ιστού σύμφωνα με τους χρήστες του είναι η "ταχύτητα" ή πιο σωστά *"η ποιότητα υπηρεσίας" (Quality of Service ή QoS)* [96]. Συνεπώς, το σημαντικό πρόβλημα που επωμίζονται οι μηχανικοί και οι επιστήμονες των συστημάτων πληροφορικής είναι φανερό :

**Πώς θα βελτιώσουμε την ποιότητα υπηρεσίας του Παγκόσμιου Ιστού.**

Το πρόβλημα αυτό είναι πιο σημαντικό από ότι αρχικά φαίνεται. Όπως έχουν δείξει μελέτες [81] καθυστέρηση μεγαλύτερη των 4 δευτερολέπτων μπορεί να προκαλέσει την αφαίρεση της προσοχής του ανθρώπου, ενώ ένα πείραμα που έγινε τελευταία έδειξε ότι χωρίς την ύπαρξη μπάρας προόδου (progress bar) ή άλλων οπτικών μέσων η πλειοψηφία των χρηστών δεν περιμένουν την ολοκλήρωση μιας ενέργειας πάνω από 8.5 δευτερόλεπτα. Είναι λοιπόν φανερό ότι η μεγάλη καθυστέρηση του Παγκόσμιου Ιστού μειώνει την παραγωγικότητα των χρηστών.

Παράλληλα, είναι γεγονός ότι ο Παγκόσμιος Ιστός έχει γίνει μια "κρίσιμη" εφαρμογή για ένα συνεχώς αυξανόμενο πλήθος ανθρώπων, με τη λογική ότι είναι πλέον απόλυτα αναγκαίος για τη δουλειά τους και άρα των εταιρειών στις οποίες εργάζονται. Εφόσον λοιπόν η χαμηλή ποιότητα υπηρεσίας του Παγκόσμιου Ιστού μειώνει την παραγωγικότητα των χρηστών μια εταιρείας, μειώνει την αποδοτικότητα της εταιρείας. Είναι άρα ένα πρόβλημα που μεταφράζεται σε απώλεια χρημάτων.

Δεν είναι όμως μόνο η μείωση της παραγωγικότητας από την καθυστέρηση του Παγκόσμιου Ιστού που συνεπάγεται την απώλεια χρημάτων για τις εταιρείες, αλλά και η απώλεια πελατών που πλέον χρησιμοποιούν τον Παγκόσμιο Ιστό για τις ηλεκτρονικές αγορές τους. Σύμφωνα με στοιχεία της International Data Corporation, περισσότεροι από 10 εκατομμύρια άνθρωποι σήμερα αγοράζουν αγαθά και υπηρεσίες ηλεκτρονικά [35]. Μέχρι το 2002 υπολογίζεται ότι ο αριθμός τους θα φτάσει τα 47 εκατομμύρια. Σύμφωνα με τις προβλέψεις της Forrester Research Inc. την ίδια χρονιά ο τζίρος από το ηλεκτρονικό εμπόριο στις ΗΠΑ θα φτάσει το ύψος των 327 δισεκατομμυρίων δολαρίων [35].

Με τέτοια αύξηση του τζίρου από το ηλεκτρονικό εμπόριο είναι αναμενόμενο να αυξηθεί ο ανταγωνισμός μεταξύ των εταιρειών και ένα από τα κρίσιμα ζητήματα για κάθε τέτοια εταιρεία θα είναι η ποιότητα ηλεκτρονικής πρόσβασης μέσω του WWW. Καθυστέρηση θα σημαίνει πιθανή απώλεια πελατών προς ανταγωνιστές των οποίων οι ιστοσελίδες φτάνουν στο χρήστη γρηγορότερα και κατά συνέπεια απώλεια χρημάτων.

Όλα τα παραπάνω δείχνουν πόσο σοβαρό είναι το πρόβλημα της απόδοσης του Παγκόσμιου Ιστού.

### 1.2.1 ΛΑΝΘΑΣΜΕΝΕΣ ΠΡΟΣΕΓΓΙΣΕΙΣ

Μία λανθασμένη αντίληψη που έχουν πολλοί χρήστες του Διαδικτύου είναι ότι η αιτία της χαμηλής ποιότητας υπηρεσίας WWW είναι η δικτυακή τεχνολογία που χρησιμοποιείται σήμερα, γιατί έχει χαμηλό *εύρος ζώνης (bandwidth)* και υψηλό *χρόνο απόκρισης (latency)*. Συνεπώς, κατά την άποψή τους, το πρόβλημα λύνεται αντικαθιστώντας το σημερινό Διαδικτυακό κορμό (Internet backbone) με ταχύτερα εξαρτήματα όπως fast routers, ATM switches [94] και Gigabit Ethernet Switches [95]. Δυστυχώς, αυξάνοντας το εύρος ζώνης και μειώνοντας το χρόνο απόκρισης θα έχουμε μικρή βελτίωση στην απόδοση του Παγκόσμιου Ιστού.

Παρακάτω θα προσπαθήσουμε να εξηγήσουμε το γιατί, με τα εξής τρία επιχειρήματα:

- ◇ Ακόμα κι αν βελτιώσουμε σημαντικά το Διαδικτυακό κορμό με τα πιο εξελιγμένα δικτυακά εξαρτήματα, αυξάνοντας το δικτυακό εύρος ζώνης και μειώνοντας την τιμή του ανά μονάδα, το πιθανότερο είναι η ταχύτητα του Παγκοσμίου Ιστού να αυξηθεί μόνο προσωρινά γιατί υπάρχει αυξανόμενη απαίτηση για πλουσιότερο περιεχόμενο στις ιστοσελίδες. Καθώς λοιπόν το περιεχόμενο των ιστοσελίδων γίνεται πιο πλούσιο, θα αυξάνει το μέγεθος τους, συνεπώς η κίνηση WWW για τις ίδιες σελίδες θα αυξάνει.

  Εκτός όμως από το πλουσιότερο περιεχόμενο, όπως έχουμε δει στην παράγραφο 1.1.1 η κίνηση στον Παγκόσμιο Ιστό αυξάνεται εκθετικά, άρα αν δεν αναβαθμίζουμε συνεχώς το δίκτυο με τον ίδιο εκθετικό ρυθμό, η βελτίωση που θα δούμε θα είναι προσωρινή [3]. Τέλος, πρέπει να λάβουμε υπόψη μας το κόστος που έχουν οι βελτιώσεις του Διαδικτυακού κορμού για τις οποίες μιλάμε τόσο εύκολα. Μία τέτοια επένδυση κατά μήκος όλου του Διαδικτύου είναι τεράστια, καθώς π.χ. η τιμή για κίνηση εύρους ζώνης ενός MBit το δευτερόλεπτο υπολογίζεται από τους Παροχείς Διαδικτυακών Υπηρεσιών (ISPs) σε περίπου $800 το μήνα [22].

◇ Είναι γεγονός ότι η ζήτηση και η προσφορά στον Παγκόσμιο Ιστό δεν είναι ισορροπημένη, αλλά εξαρτάται από το πόσο δημοφιλείς γίνονται κατά διαστήματα κάποιες ιστοσελίδες. Αυτό σημαίνει ότι ανά πάσα στιγμή σε κάποια σημεία του δικτύου θα υπάρχει πολύ μεγάλη κίνηση, ενώ σε άλλα θα είναι πολύ μικρή. Δημιουργούνται λοιπόν προβλήματα σε δύο κατηγορίες χρηστών:

  ▷ Στους χρήστες που προσπαθούν να δουν κάποιες πολύ δημοφιλείς ιστοσελίδες (flash crowds), γιατί λόγω του ότι προσπαθούν πάρα πολλοί ταυτόχρονα προκαλούν υπερβολικό φόρτο στους Εξυπηρέτες που έχουν τις ιστοσελίδες, με αποτέλεσμα οι τελευταίοι να μην αντέχουν το φορτίο και να διακόπτουν τη λειτουργία τους. Έτσι σε πολλές περιπτώσεις κανένας χρήστης δεν μπορεί να δει τις πολύ δημοφιλείς ιστοσελίδες.

  ▷ Παράλληλα, δημιουργείται πρόβλημα και σε άλλους χρήστες που δρομολογούνται μέσα από τα φορτωμένα τμήματα του δικτύου, αλλά δεν ενδιαφέρονται για τις συγκεκριμένες ιστοσελίδες. Αυτοί οι χρήστες απορούν γιατί υπάρχει καθυστέρηση, ενώ η δική τους σύνδεση με το Διαδίκτυο είναι γρήγορη.

Συμπεραίνουμε λοιπόν ότι αν βελτιώσουμε συνολικά το Διαδικτυακό κορμό με πιο εξελιγμένα δικτυακά εξαρτήματα, η μη ισοκατανομή της κίνησης θα συνεχίζει να προκαλεί προβλήματα γιατί θα παρουσιάζεται παράλογη κίνηση σε κάποια τμήματά του.

◇ Για να δούμε ότι πραγματικά το πρόβλημα δεν είναι στην δικτυακή υποδομή, ας έρθουμε τώρα σε μια υποθετική κατάσταση και υποθέτοντας ότι αγνοούμε το κόστος που αναφέραμε παραπάνω. Έστω ότι όλοι οι Διαδικτυακοί κόμβοι απαντούσαν στιγμιαία σε όλους τους χρήστες, δηλαδή είχαν μηδενική καθυστέρηση απόκρισης και έστω ότι υπήρχε άπειρο εύρος ζώνης σε ολόκληρο το Διαδίκτυο. Ας υποθέσουμε ακόμα ότι όλοι οι *Εξυπηρέτες WWW (WWW servers)* απαντούσαν με μηδενική καθυστέρηση και ότι είχαμε την καλύτερη δυνατή δρομολόγηση κατά μήκος ολόκληρου του Διαδικτύου.

Ακόμα και σε αυτή την βέλτιστη θεωρητική κατάσταση θα είμαστε δεσμευμένοι από την ταχύτητα του φωτός, ένα φυσικό εμπόδιο το οποίο σε πολλές περιπτώσεις είναι αρκετό για να προκαλέσει από μόνο του αρκετή καθυστέρηση. Όταν μιλάμε για την ταχύτητα του φωτός, εννοούμε το χρόνο που χρειάζονται τα ψηφιακά δεδομένα για να περάσουν μέσα από μια οπτική ίνα. Ο χρόνος αυτός έχει ένα ελάχιστο όριο που εξαρτάται αποκλειστικά από την ταχύτητα του φωτός.

Αν, για παράδειγμα, υπήρχε ένας Πελάτης WWW στην Δυτική Ακτή των ΗΠΑ ο οποίος θέλει να διαβάσει μια ιστοσελίδα με 10 ενσωματωμένα αντικείμενα[5] (embedded objects) η οποία βρίσκεται σε ένα Εξυπηρέτη WWW στην Ανατολική Ακτή των ΗΠΑ, θα έχει καθυστέρηση περίπου ενός δευτερολέπτου η οποία θα προκαλείται αποκλειστικά από την ταχύτητα του φωτός[6]. Αυτός ο χρόνος οφείλεται στο γεγονός ότι για να διαδοθεί ένα οπτικό σήμα σε απόσταση ίση με αυτήν από τη μία Ακτή των ΗΠΑ στην άλλη χρειάζεται περίπου 27 χιλιοστά του δευτερολέπτου (ms) [43].

Αν ένας χρήστης βρισκόταν στην Ιαπωνία και ήθελε να διαβάσει μια σελίδα σε κάποιο Εξυπηρέτη των ΗΠΑ με πλούσιο περιεχόμενο η οποία περιείχε 40 ενσωματωμένα αντικείμενα θα είχε καθυστέρηση περίπου δέκα δευτερολέπτων,[43] μόνο εξαιτίας της ταχύτητας του φωτός. Αυτό μπορεί να μην είναι τόσο προφανές σήμερα όπου οι περισσότεροι χρήστες του Παγκόσμιου Ιστού βρίσκονται στις ΗΠΑ.Όσο όμως το Διαδίκτυο επεκτείνεται παγκοσμίως, εκατοντάδες εκατομμύρια χρήστες θα προσπελαύνουν εκατομμύρια Εξυπηρέτες WWW που συχνά θα βρίσκονται σε αποστάσεις 10 ή 15 χιλιάδων χιλιομέτρων μακριά. Τότε η ταχύτητα του φωτός θα γίνει καθοριστικός παράγοντας καθυστέρησης για όλες τις υπηρεσίες του Διαδικτύου.

## 1.3 Η ΛΥΣΗ ΤΗΣ ΕΝΔΙΑΜΕΣΗΣ ΜΝΗΜΗΣ (WWW CACHING)

Η καλύτερη βραχυπρόθεσμη αλλά και μακροπρόθεσμη λύση στο πρόβλημα της ποιότητας υπηρεσίας του Παγκόσμιου Ιστού είναι να μεταφέρουμε την πλειοψηφία των δημοφιλών αντικειμένων πιο κοντά στους χρήστες, μειώνοντας την απόσταση και τους ενδιάμεσους κόμβους ανάμεσα στο χρήστη και στα αντικείμενα αυτά [3]. Έτσι επιτυγχάνουμε τρεις πρωταρχικούς στόχους :

◇ Μειώνουμε την καθυστέρηση γιατί πλέον ανάμεσα στο χρήστη και τα δημοφιλή αντικείμενα υπάρχουν πολύ λίγοι ενδιάμεσοι δρομολογητές ή μεταγωγείς για να

[5] Με τον όρο ενσωματωμένα αντικείμενα εννοούμε τα αντικείμενα που είναι συνδεμένα με μια ιστοσελίδα, όπως μικρά εικονίδια, κινούμενα γραφικά, κομμάτια hypertext κτλ.

[6] Ο χρόνος αυτός αφορά την έκδοση 1.0 του πρωτοκόλλου HTTP, όπου για κάθε αντικείμενο WWW της ιστοσελίδας ανοίγεται νέα TCP σύνδεση. Στην έκδοση 1.1 του πρωτοχόλλου HTTP, όπως θα δούμε στην παράγραφο 2.3, η κατάσταση βελτιώνεται αισθητά.

προκαλέσουν καθυστέρηση.

◇ Μειώνουμε την κίνηση στα "μακρινά" τμήματα του Διαδικτύου.

◇ Μειώνουμε δραστικά το πρόβλημα που δημιουργεί μακροχρόνια η ταχύτητα του φωτός, γιατί μειώνουμε την απόσταση που πρέπει να διανύσει το οπτικό ή ηλεκτρικό σήμα στο φυσικό μέσο (οπτική ίνα ή μεταλλικό καλώδιο).

Όπως θα δούμε παρακάτω τα πλεονεκτήματα από αυτή τη λύση είναι πολύ περισσότερα.

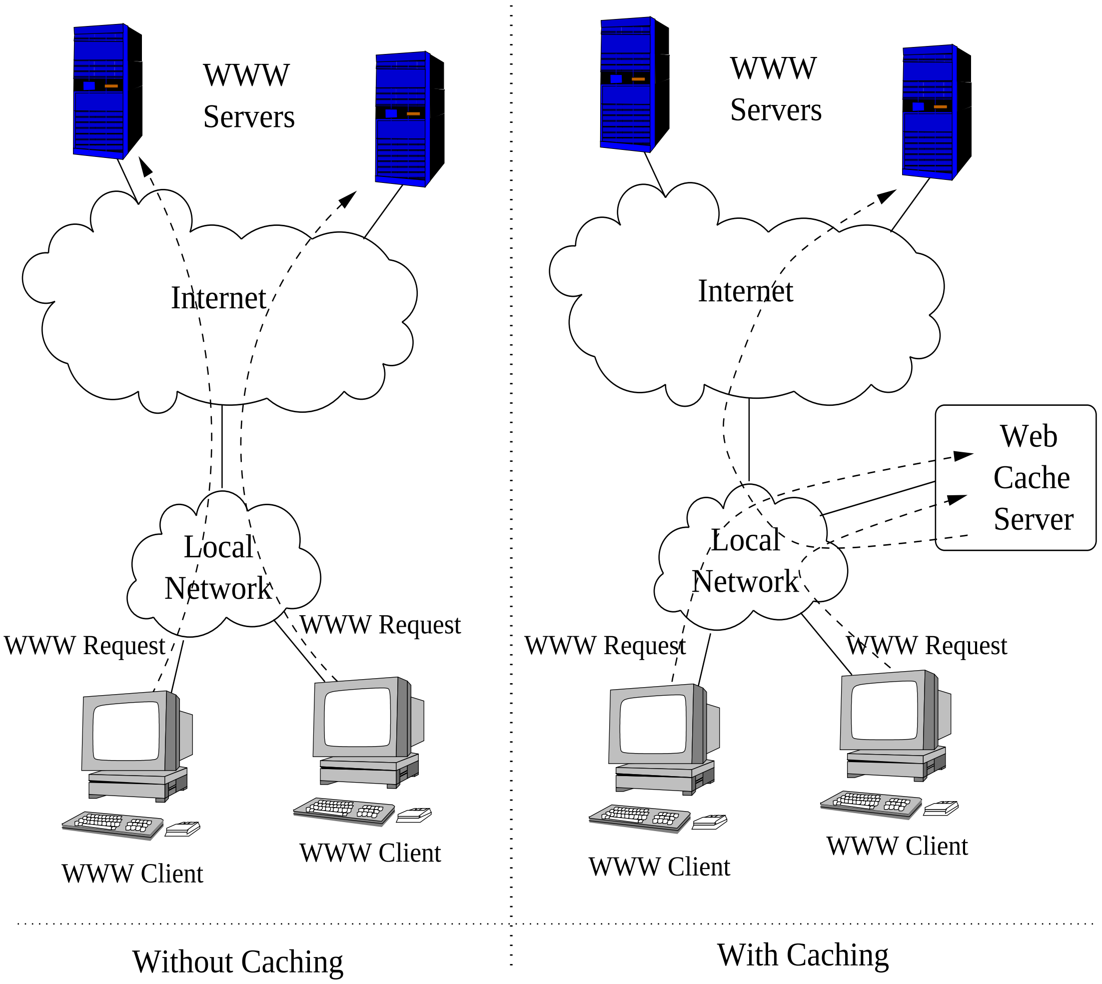
Η τεχνική της μεταφοράς των δημοφιλέστερων αντικειμένων πλησιέστερα στον χρήστη του Παγκόσμιου Ιστού επιτυγχάνεται με τη χρήση της τεχνολογίας *ενδιάμεσης μνήμης του Παγκόσμιου Ιστού (Web Caching)*. Στην ορολογία του Παγκόσμιου Ιστού, *η ενδιάμεση μνήμη (cache)* είναι μία τοποθεσία όπου αποθηκεύονται προσωρινά αντίγραφα των αντικειμένων του.

Η χρήση ενδιάμεσης μνήμης δεν είναι κάτι καινούριο στον κόσμο των υπολογιστών. Όλα τα σύγχρονα υπολογιστικά συστήματα χρησιμοποιούν αυτή την τεχνολογία σε αρκετά σημεία της αρχιτεκτονικής τους για να βελτιώσουν τις επιδόσεις της ιεραρχίας μνήμης, του επεξεργαστή, των δίσκων κτλ. Η ιδέα της ενδιάμεσης μνήμης δεν έχει χρησιμοποιηθεί όμως μόνο σε συστήματα υλικού (hardware), αλλά και σε λογισμικό. Μερικές υπηρεσίες του Διαδικτύου, όπως για παράδειγμα η Domain Name Service (DNS) χρησιμοποιεί την αρχή αυτή για να βελτιώσει τις επιδόσεις της. Η λύση αυτή άργησε να χρησιμοποιηθεί στον Παγκόσμιο Ιστό και ο βασικός λόγος είναι ότι δεν έγινε πρόβλεψη για τέτοια λειτουργία στον αρχικό σχεδιασμό του Παγκόσμιου Ιστού. Σήμερα γίνεται μια ευρεία προσπάθεια ώστε να εισαχθεί εκ των υστέρων η τεχνολογία της ενδιάμεσης μνήμης στον Παγκόσμιο Ιστό με σκοπό την αποδοτικότερη λειτουργία του.

Η βασική λειτουργία της τεχνολογίας της ενδιάμεσης μνήμης φαίνεται στο σχήμα 1.1 παρακάτω. Βέβαια όπως θα δούμε σε επόμενα κεφάλαια η αποδοτική λειτουργία της τεχνολογίας αυτής δεν είναι τόσο απλή όσο φαίνεται στο σχήμα.

Όπως φαίνεται στην αριστερή εικόνα του σχήματος αυτού, χωρίς την χρήση ενδιάμεσης μνήμης οι αιτήσεις των χρηστών για αντικείμενα του παγκόσμιου ιστού περνούσαν από το τοπικό δίκτυο του χρήστη και μέσω του Διαδικτύου εξυπηρετούνταν κατευθείαν από τους μακρινούς Εξυπηρέτες Παγκόσμιου Ιστού (WWW servers). Αυτό έχει ως αποτέλεσμα αν γίνουν δύο αιτήσεις από χρήστες στο ίδιο τοπικό δίκτυο για το ίδιο αντικείμενο να εξυπηρετηθούν από τους μακρινούς Εξυπηρέτες σπαταλώντας έτσι Διαδικτυακούς πόρους.

Αντίθετα στη δεξιά πλευρά του σχήματος φαίνεται η λειτουργία του Παγκόσμιου Ιστού με χρήση ενδιάμεσης μνήμης. Όπως παρατηρεί ο αναγνώστης υπάρχει ένα επι-




**Σχήμα 1.1 Διαφορές στη λειτουργία την υπηρεσίας WWW με χρήση ενδιάμεσης μνήμης και χωρίς αυτήν.**

πλέον δικτυακό υποσύστημα που ονομάζεται *Εξυπηρέτης Ενδιάμεσης Μνήμης (Web Cache Server ή Web Proxy)* [7]. Πρόκειται για ένα αφοσιωμένο (dedicated) υπολογιστικό σύστημα το οποίο συνήθως διαθέτει μεγάλο αποθηκευτικό χώρο σε μνήμη (κύρια μνήμη ή συστήματα δίσκων), παρακολουθεί όλες τις αιτήσεις που γίνονται για αντικείμενα του Παγκόσμιου Ιστού στο τοπικό δίκτυο (ή γενικότερα σε ένα δίκτυο) και αναλαμβάνει να τις εξυπηρετήσει. Η εξυπηρέτηση των αιτήσεων διαφέρει ανάλογα με το αντικείμενο κάθε μιας και διακρίνεται σε δύο περιπτώσεις:

◇ Αν το αντικείμενο της αίτησης δεν υπάρχει στη μνήμη του Web Proxy, ο τελευταίος συνδέεται με τον μακρινό Εξυπηρέτη Παγκόσμιου Ιστού που έχει το αντι-

[7] Ο Εξυπηρέτης Ενδιάμεσης Μνήμης διαφέρει από τον Εξυπηρέτη Παγκόσμιου Ιστού. Στη συνέχεια της εργασίας θα χρησιμοποιούμε τους αγγλικούς όρους Web Proxy ή Web Cache για τον Εξυπηρέτη Ενδιάμεσης Μνήμης έτσι ώστε να αποφευχθούν τυχόν ασάφειες που θα μπερδέψουν τον αναγνώστη.

κείμενο, το φέρνει και το επιστρέφει στον χρήστη που το ζήτησε. Παράλληλα, αν ο Web Proxy κρίνει ότι το αντικείμενο αυτό είναι δημοφιλές και θα ξαναζητηθεί σύντομα από κάποιο χρήστη στο δίκτυο που παρακολουθεί, το αποθηκεύει στην μνήμη του.

◇ Στη δεύτερη περίπτωση, αν το αντικείμενο της αίτησης υπάρχει στη μνήμη του Web Proxy, ο τελευταίος το επιστρέφει κατευθείαν στον χρήστη που το ζήτησε χωρίς να συνδεθεί με τον μακρινό Εξυπηρέτη. Έτσι η αίτηση εξυπηρετείται ταχύτατα και γίνεται μεγάλη οικονομία στο εύρος ζώνης του Διαδικτύου.

Η χρήση ενδιάμεσης μνήμης έχει πλεονεκτήματα, αλλά η υλοποίησή της παρουσιάζει αρκετές προκλήσεις γιατί καλείται να λύσει μερικά πολύπλοκα αλλά και ενδιαφέροντα προβλήματα. Στις επόμενες παραγράφους θα ασχοληθούμε με τα πλεονεκτήματα της ενδιάμεσης μνήμης και τα προβλήματα που παρουσιάζει.

### 1.3.1 ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΤΩΝ WWW CACHES

Η χρήση ενδιάμεσης μνήμης έχει αρκετά πλεονεκτήματα [17], [91], [42]:

◇ **Καλύτερη ποιότητα υπηρεσίας στο Διαδίκτυο.**
Τα αντικείμενα Web που έρχονται από τους μακρινούς Εξυπηρέτες μεταφέρονται από μακρινά δίκτυα και περνάνε από πολλούς ενδιάμεσους δρομολογητές και μεταγωγείς μέχρι να φτάσουν στον browser του χρήστη. Η χρήση ενδιάμεσης μνήμης ελαττώνει πολύ τον χρόνο απόκρισης γιατί ένα μεγάλο ποσοστό αντικειμένων έρχεται από το τοπικό δίκτυο. Συνεπώς βελτιώνεται η ποιότητα υπηρεσίας στον Παγκόσμιο Ιστό.

◇ **Μείωση της συνολικής κίνησης στο Διαδίκτυο.**
Όσοι περισσότεροι χρήστες χρησιμοποιούν την τεχνολογία ενδιάμεσης μνήμης, τόσο πιο πιθανό είναι τα αντικείμενα που ζητούν να τα έχει ζητήσει και κάποιος άλλος χρήστης, και άρα να υπάρχουν στη μνήμη κάποιου Web proxy. Τότε λοιπόν θα έχουμε τη βέλτιστη λειτουργία του Web proxy και άρα τη μεγαλύτερη μείωση της κίνησης του Διαδικτύου.

Έτσι, αν ένας Web proxy έχει ποσοστό επιτυχημένων αιτήσεων (document hit rate) 35%, δηλαδή το 35% των αιτήσεων που του γίνονται είναι για αντικείμενα που υπάρχουν στην μνήμη του και δεν χρειάζεται να τα φέρει από τους μακρινούς Εξυπηρέτες, τότε αντίστοιχα 35% θα είναι και η μείωση της κίνησης από το δίκτυο που βρίσκεται ο Web proxy προς το υπόλοιπο Διαδίκτυο.

◊ **Προστασία των Εξυπηρετών Web από μεγάλο φόρτο.**
Η ενδιάμεση μνήμη προστατεύει τους Εξυπηρέτες από το μεγάλο φορτίο που μπορεί να δημιουργηθεί από πολύ δημοφιλή αντικείμενα του Web. Για παράδειγμα το 1998 ήταν τόσοι οι χρήστες που ζητούσαν την έκθεση Starr[8] που δημιουργήθηκε ένα πολύ μεγάλο "μποτιλιάρισμα" στο Διαδίκτυο και η εξομάλυνση της κίνησης έγινε αρκετές ώρες αργότερα [17].

Οι Web Proxies είναι ιδανικοί για τέτοιες καταστάσεις, γιατί όσοι περισσότεροι χρήστες ζητάνε ένα αντικείμενο WWW, τόσο πιθανότερο είναι το αντικείμενο να υπάρχει στη μνήμη του Web proxy και άρα τόσο πιο αποδοτικός θα είναι ο Web proxy στη μείωση της κίνησης.

◊ **Μείωση του κόστους του εύρους ζώνης για τους Παροχείς Υπηρεσιών Διαδικτύου και άρα για τους χρήστες.**
Πολλές εταιρείες και Παροχείς Υπηρεσιών Διαδικτύου χρησιμοποιούν σήμερα λύσεις ενδιάμεσης μνήμης για να μειώσουν την δικτυακή τους κίνηση προς τα έξω και να κερδίσουν έτσι πολλά χρήματα που θα έδιναν σε υψηλότερου εύρους ζώνης γραμμές. Παράλληλα παρέχουν υψηλότερη ποιότητα υπηρεσίας στους χρήστες τους.

Για παράδειγμα η HomeNet Communications, Παροχέας Υπηρεσιών Διαδικτύου στο Warner Robins της Georgia των ΗΠΑ χρησιμοποιεί δύο προϊόντα ενδιάμεσης μνήμης, το CacheFlow 1000 και το SkyCache των ομώνυμων εταιρειών για να ελαττώσει κατά 55% την κίνηση προς το υπόλοιπο Διαδίκτυο [18]. Έτσι κερδίζει αρκετά χρήματα που θα έδινε για παραπάνω συνδέσεις με τον κορμό του Διαδικτύου, παράλληλα προσφέρει υψηλότερη ποιότητα υπηρεσίας στους πελάτες της. Αξίζει να παρατηρήσουμε ότι η ενδιάμεση μνήμη παρέχει χαμηλότερη καθυστέρηση στους χρήστες της HomeNet από την περίπτωση όπου η εταιρεία αυτή θα είχε γραμμές διπλάσιου εύρους ζώνης προς το υπόλοιπο Διαδίκτυο [18].

Αλλες έρευνες αγοράς επισημαίνουν ότι χρησιμοποιώντας συστήματα ενδιάμεσης μνήμης ένας Παροχέας Υπηρεσιών Διαδικτύου μπορεί να εξοικονομήσει μέχρι και $86.000 το χρόνο ανά γραμμή T3 (δηλαδή εύρους ζώνης μέχρι 45MBps) [42].

### 1.3.2 Η ΙΔΑΝΙΚΗ WWW CACHE ΚΑΙ ΠΟΣΟ ΕΦΙΚΤΗ ΕΙΝΑΙ

Σε ένα τέλειο κόσμο, για να χαρακτηρίσουμε ένα Web proxy ιδανικό, θα έπρεπε να έχει τα οκτώ χαρακτηριστικά που περιγράφουμε παρακάτω [91]. Επειδή όμως κάτι τέτοιο

[8] Η έκθεση αφορούσε τις ανακρίσεις του επιθεωρητή Starr σχετικά με την υπόθεση του σκανδάλου Lewinsky στο οποίο είχε αναμιχθεί ο Bill Clinton [51].

δεν συμβαίνει, μαζί με τα χαρακτηριστικά αυτά που αποτελούν τους επιδιωκόμενους στόχους μας αναφέρουμε και το πόσο τα πλησιάζει σήμερα ένα σύστημα καθώς και τα προβλήματα που παρουσιάζονται σε κάθε περίπτωση. Το ιδανικό Web proxy:

1. **Έχει ποσοστό επιτυχημένων αιτήσεων (document hit rate) 100%.**
   Το ιδανικό θα ήταν όλες οι αιτήσεις των χρηστών να εξυπηρετούνταν από τη μνήμη του Web proxy. Όπως είναι κατανοητό, για να γίνει αυτό εφικτό πρέπει η μνήμη του Web proxy να χωράει όλα τα αντικείμενα του Παγκόσμιου Ιστού, κάτι πρακτικά αδύνατο. Αλλά ακόμα κι αν συνέβαινε κάτι τέτοιο δεν θα μπορούσαμε να αποθηκεύσουμε τα δυναμικά αντικείμενα ή αυτά που μεταβάλλονται καθημερινά στον Παγκόσμιο Ιστό.

   Τυπικά ποσοστά επιτυχημένων αιτήσεων σήμερα κυμαίνονται από 30% μέχρι 50% [3], [34], [85], [93]. Το ποσοστό αυτό εξαρτάται από την πολιτική επιλογής των αντικειμένων που πρέπει να αποθηκευτούν στην μνήμη του Web proxy. Πρόκειται στην ουσία για μία πολιτική αντικατάστασης αντικειμένων γιατί ο Web proxy συνήθως γεμίζει πρώτα τη μνήμη του με αντικείμενα και έπειτα επιλέγει ποια από αυτά θα κρατήσει και ποια θα αντικαταστήσει με καινούρια από τους μακρινούς Εξυπηρέτες.

   Το ερευνητικό πρόβλημα της πολιτικής αντικατάστασης αντικειμένων είναι αρκετά ενδιαφέρον γιατί χρειάζεται μελέτη της συμπεριφοράς μεγάλου πλήθους χρηστών και της ακολουθίας των αιτήσεων τους [3], [1], [2], [34]. Στον τομέα αυτό έχει γίνει αρκετή έρευνα και έχει αποδειχτεί ότι μία αρκετά καλή πολιτική για την αντικατάσταση είναι αυτή της "Ελάχιστα Πρόσφατα Χρησιμοποιημένου" αντικειμένου, ο γνωστός από τα λειτουργικά συστήματα αλγόριθμος αντικατάστασης σελίδων "Least Recently Used" (LRU), με κάποιο όμως άνω όριο στο μέγεθος των αντικειμένων (δηλαδή δεν αποθηκεύουμε τα πολύ μεγάλα αντικείμενα στην μνήμη γιατί σπαταλάμε πολύτιμο χώρο) [3]. Ο αλγόριθμος αυτός ονομάζεται LRU-TH (από το LRU-Threshold) [15]. Βέβαια όταν μιλάμε για μια "πολύ καλή πολιτική", εννοούμε ότι είναι μια γενική πολιτική και ότι έχει μικρό σφάλμα (2 - 3% χαμηλότερο ποσοστό επιτυχημένων αιτήσεων) από μία βέλτιστη πολιτική στις περισσότερες περιπτώσεις. Ο αναγνώστης μπορεί να βρει περισσότερες πληροφορίες για τις πολιτικές αντικατάστασης στις παρακάτω παραπομπές [15], [48], [34], [52], [76], [86] [29], [3], [104], [8], [9].

2. **Έχει άμεσες αποκρίσεις για αντικείμενα που βρίσκονται εντός της ενδιάμεσης μνήμης, ενώ δεν προκαλεί επιπλέον καθυστέρηση στις αποτυχημένες αιτήσεις.**
   Ένας σημαντικός στόχος που έχει σχέση με την απόδοση του Web proxy σαν

σύστημα είναι οι γρήγορες αποκρίσεις προς τους χρήστες, έτσι ώστε να εξυπηρετεί όσους περισσότερους χρήστες ανά μονάδα χρόνου είναι δυνατόν. Αποτελεί ένα δύσκολο στόχο, γιατί ένας Web proxy μπορεί να δέχεται από δεκάδες μέχρι δεκάδες χιλιάδες αιτήσεις ανά δευτερόλεπτο ανάλογα με τους χρήστες που εξυπηρετεί, κάτι που προκαλεί υπερβολικό φόρτο στο υπολογιστικό σύστημα που τον υποστηρίζει [19], [73].

Σήμερα τα περισσότερα Web proxies δουλεύουν σε μεγάλα συστήματα με πολλούς και ισχυρούς επεξεργαστές, άφθονη κεντρική μνήμη (RAM), ισχυρό υποσύστημα παράλληλων δίσκων (RAID) [71], [21], [20] και πολλές φορές ειδικά λειτουργικά συστήματα για να μπορέσουν να υποστηρίξουν ένα ικανοποιητικό αριθμό αιτήσεων από τους χρήστες [19], [73].

Με τα προβλήματα απόδοσης των Web proxies έχουν ασχοληθεί αρκετοί ερευνητές και θα ασχοληθούμε κι εμείς στην παρούσα γραπτή εργασία. Στα επόμενα κεφάλαια θα αναλύσουμε το πρόβλημα σε βάθος και θα επεκταθούμε σε λύσεις που χρησιμοποιούνται σήμερα σε υπάρχοντα συστήματα καθώς θα δείξουμε τη δική μας συνεισφορά στη λύση αυτών των προβλημάτων [62].

3. **Δεν δημιουργεί άσκοπη δικτυακή κίνηση προς το έξω Διαδίκτυο.**
   Η δημιουργία άσκοπης δικτυακής κίνησης προς τα έξω εξαρτάται κυρίως από την πολιτική αντικατάστασης που είδαμε στον πρώτο στόχο και από το πώς χειρίζεται ένα Web proxy τα δυναμικά αντικείμενα του Παγκόσμιου Ιστού. Σκεπτόμενοι την αρχική ιδέα των Web proxies σαν χώροι ενδιάμεσης αποθήκευσης αντικειμένων, των οποίων οι ενέργειες καθοδηγούνται αυστηρά από τις αιτήσεις των χρηστών μπορούμε να πούμε ότι κάθε Web proxy θα ήταν "αρκετά καλό" σε αυτό το σημείο. Όμως σήμερα μελετώνται στρατηγικές *prefetching* (δηλαδή ανάκτησης σε ώρες μη αιχμής αντικειμένων που προβλέπονται να ζητηθούν από τους χρήστες σε ώρες αιχμής)[9] και έχουν υλοποιηθεί πρωτόκολλα όπως το Internet Cache Protocol (ICP) [101], σύμφωνα με το οποίο ανταλλάσσονται πληροφορίες (cache-digests) ανάμεσα σε Web proxies για τα αντικείμενα που καθένα έχει αποθηκεύσει [93]. Μπορούμε να πούμε λοιπόν ότι ένα Web proxy που χρησιμοποιεί τέτοιες τεχνικές είναι επικίνδυνο να δημιουργήσει άσκοπη Διαδικτυακή κίνηση.

4. **Κάνει βέλτιστη χρήση του χώρου της ενδιάμεσης μνήμης.**
   Ένας Web proxy πρέπει να κάνει βέλτιστη χρήση του χώρου μνήμης που διαθέτει, χωρίς για παράδειγμα να αποθηκεύει περισσότερα από ένα αντίγραφα του ίδιου αντικειμένου. Όσο περισσότερα αντικείμενα χωράνε στη μνήμη του Web proxy

[9] Περισσότερες πληροφορίες για τα θέματα του prefetching στα [36], [61], [59].

τόσο υψηλότερο θα είναι το ποσοστό επιτυχημένων αιτήσεων και άρα η μείωση της κίνησης του Web proxy προς το έξω Διαδίκτυο.

5. **Λειτουργεί πάντα (έχει 100% uptime).**
   Ανάλογα με το είδος εξάρτησης των χρηστών σε ένα δίκτυο με το Web proxy μπορεί η διακοπή της λειτουργίας του να σημαίνει από πλήρη διακοπή της υπηρεσίας του Παγκόσμιου Ιστού (όταν π.χ. οι χρήστες βρίσκονται πίσω από δίκτυο που προστατεύεται με firewall) μέχρι απλά αλλαγή κάποιων παραμέτρων στον Web browser κάθε χρήστη. Σε κάθε περίπτωση πάντως, η βέλτιστη κατάσταση είναι ο Web proxy να είναι διαθέσιμος κάθε στιγμή και να δουλεύει σωστά. Το θέμα αυτό εξαρτάται κυρίως από την υλοποίηση του Web proxy και άρα ένα σωστά σχεδιασμένο Web proxy δεν θα έχει πρόβλημα με αυτό το θέμα.

6. **Δεν έχει αρνητική επίδραση σε Διαδικτυακή κίνηση εκτός Παγκοσμίου Ιστού.**
   Είναι κατανοητό ότι η λειτουργία του Web proxy δεν πρέπει να επηρεάζει τη λειτουργία των υπολοίπων δικτυακών υπηρεσιών, όπως π.χ. η υπηρεσία telnet, η υπηρεσία email κτλ. Η απαίτηση αυτή καλύπτεται σήμερα από όλα τα Web proxies.

7. **Έχει μηδενικό κόστος αγοράς.**
   Το θέμα του κόστους αφορά σίγουρα το πόσο θα εξαπλωθεί η τεχνολογία αυτή και άρα το πόσο θα ωφεληθούν συνολικά οι χρήστες του Διαδικτύου. Έτσι ενώ το θέμα του κόστους καθαυτό, δεν αποτελεί απαραίτητα ερευνητικό θέμα, το πως μπορούμε να έχουμε υψηλή απόδοση σε μηχανές χαμηλού κόστους ώστε η τεχνολογία των Web proxies να εξαπλωθεί ευκολότερα είναι ένα σοβαρό ερευνητικό πρόβλημα.

   Σήμερα υπάρχουν κυρίως δύο ειδών Web proxies από άποψη κόστους. Υπάρχουν από τη μία κάποια που η χρήση του λογισμικού τους δεν κοστίζει τίποτα και τρέχουν σε κοινό υπολογιστικό υλικό, δηλαδή σε ευρέως διαδεδομένα υπολογιστικά συστήματα χαμηλού ή μέσου κόστους. Τα πιο σημαντικά από αυτά είναι το Squid [102], ο Apache [7] και ο CERN httpd [57] που τρέχουν στα λειτουργικά συστήματα Unix, το DeleGate που τρέχει και σε Unix και σε Windows NT, ενώ υπάρχουν και τα MOWS και W3C's Jigsaw που έχουν υλοποιηθεί σε Java. Το πιο δημοφιλές και σημαντικό από αυτά σε επιδόσεις, σε λειτουργικότητα αλλά και μελετημένο ερευνητικά είναι το Squid, για το οποίο υπάρχει διαθέσιμος και ο πηγαίος κώδικάς του [80], [19].

   Από την άλλη υπάρχουν και πολλά εμπορικά Web proxies με κόστος που φτάνει τις πολλές δεκάδες εκατομμύρια δραχμές, τα οποία χρησιμοποιούν ειδικό υλικό (hardware), λειτουργικό σύστημα και ειδικά βελτιστοποιημένο λογισμικό για να ανεβάσουν τις επιδόσεις τους. Τα γνωστότερα [80], [19] τέτοια συστήματα Web Cache είναι το CacheFlow της ομώνυμης εταιρείας [13], το Cache Engine της Cisco [23], το NetCache της Network Appliance [67], ο BorderManager από τη Novell [68], το Proxy Server της Netscape [66], το Web Cache από την IBM [24], το DynaCache της InfoLibria [45] και ο Traffic Server της Inktomi [47].

8. **Δεν χρειάζεται μεγάλες αλλαγές για να εγκατασταθεί σε ένα δίκτυο, ούτε σημαντικό χρόνο για τη διαχείρισή του.**
Αυτός ο στόχος έχει σχέση με τον τρόπο που ένα Web proxy παρακολουθεί ένα δίκτυο δηλαδή με το σημείο εγκατάστασης του. Το βέλτιστο θα ήταν να μην χρειάζεται καμμία παρέμβαση από το χρήστη ώστε ο Web browser του να χρησιμοποιεί το Web proxy, αλλά να παρακολουθούνται όλες οι αιτήσεις Παγκόσμιου Ιστού και αυτόματα να εξυπηρετούνται από το Web proxy. Η λύση αυτή υλοποιείται από αρκετά συστήματα, όπου είναι όμως αναγκαία η εγκατάσταση του Web proxy σε ένα κεντρικό δρομολογητή ή μεταγωγέα δικτύου ώστε να ελέγχονται όλες οι αιτήσεις Παγκοσμίου Ιστού προς τα έξω. Σε πολλά όμως δίκτυα που αυτού του είδους η εγκατάσταση δεν είναι εφικτή, είναι αναγκαίο όλοι οι χρήστες να αλλάξουν κάποιες παραμέτρους στον browser τους. Περισσότερες λεπτομέρειες για το θέμα αυτό αναφέρουμε στην παράγραφο 2.2 όπου αναλύουμε τα είδη εγκατάστασης ενός Web proxy.

Συνοψίζοντας λοιπόν τους παραπάνω στόχους για ένα τέλειο Web proxy βρίσκουμε ότι τα κύρια ερευνητικά προβλήματα εστιάζονται σε δύο κυρίως τομείς :

- ◇ Στην πολιτική εισαγωγής και αντικατάστασης ιστοσελίδων στην μνήμη του Web proxy, ώστε να επιτύχουμε τον πρώτο, τρίτο και τέταρτο στόχο από αυτούς που αναφέραμε παραπάνω [15].
- ◇ Στην αποδοτική λειτουργία του Web proxy ώστε να επιτύχουμε τον δεύτερο στόχο της άμεσης απόκρισης στις αιτήσεις των χρηστών.

## 1.4 Η ΔΙΚΗ ΜΑΣ ΣΥΝΕΙΣΦΟΡΑ

Η εξάπλωση των Web proxies σήμερα δεν είναι η επιθυμητή - ούτε η αναμενόμενη αν σκεφτεί κανείς το κέρδος που μπορεί να προσφέρει - παρ' ότι σε αρκετά σημεία η λειτουργία τους πλησιάζει τους στόχους που θέσαμε στην προηγούμενη παράγραφο. Πιστεύουμε ότι η μικρότερη της επιθυμητής εξάπλωση των Web proxies οφείλεται κατά κύριο λόγο στην ανυπαρξία ενός Web proxy μικρού κόστους (το οποίο συνεπάγεται τη χρήση διαδεδομένου υλικού (hardware) και λειτουργικού συστήματος) που παράλληλα θα παρέχει και υψηλές επιδόσεις.

Στην παρούσα γραπτή εργασία θα ασχοληθούμε με το ερευνητικό θέμα της απόδοσης των Web proxies και ειδικότερα θα εστιάσουμε στο σημείο της αποθήκευσης των αντικειμένων WWW στους δίσκους που αποτελεί σύμφωνα με μελέτες [79], [77], [6] το μεγαλύτερο παράγοντα μείωσης των επιδόσεων. Όπως θα δούμε στη συνέχεια της παρούσας μεταπτυχιακής εργασίας η δική μας συνεισφορά αφορά:

- ◇ Την περιληπτική παρουσίαση μεθόδων για τη βελτίωση του υποσυστήματος δίσκων ενός Web proxy. Μέσα από μελέτη της χρήσης των δίσκων και των συστημάτων αρχείων παρουσιάζουμε αποτελεσματικούς αλγορίθμους για την αύξηση της ταχύτητας εγγραφής και ανάγνωσης των δεδομένων από τους δίσκους. Οι αλγόριθμοι αυτοί αναπτύχθηκαν από μια ομάδα ανθρώπων στην οποία συμμετείχε και ο συγγραφέας και παρουσιάστηκαν εκτενώς στο [62].

- ◇ Την ανάπτυξη ενός πρότυπου Web proxy (το οποίο ονομάσαμε FOXY) σχεδιασμένου ώστε να προσφέρει υψηλές επιδόσεις υλοποιώντας τους παραπάνω μηχανισμούς για τη βελτίωση της ταχύτητας εγγραφής και ανάγνωσης των δεδομένων από τους δίσκους. Το Web proxy που αναπτύξαμε, χρησιμοποιεί διαδεδομένο υλικό (hardware) και γενικού σκοπού λειτουργικά συστήματα.

- ◇ Τον πειραματισμό και τη σύγκριση της υλοποίησής μας με το Squid [102], που αποτελεί το πιο δημοφιλές και προηγμένο Web proxy ανοιχτού κώδικα που τρέχει σε γενικού σκοπού λειτουργικά συστήματα σήμερα. Όπως θα αποδείξουν τα πειράματά μας, η υλοποίησή μας έχει απόδοση τουλάχιστον τρεις φορές υψηλότερη από το Squid.

Πιστεύουμε ότι η συνεισφορά μας αυτή προάγει την τεχνολογία των Web proxies και τις ευεργετικές συνέπειες της στην καθημερινή χρήση του Παγκόσμιου Ιστού.

## 1.5 ΟΡΓΑΝΩΣΗ ΤΗΣ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Στο επόμενο κεφάλαιο 2 θα κάνουμε μια εισαγωγή περισσότερο τεχνική στα θέματα της λειτουργίας του Παγκόσμιου Ιστού και θα εξερευνήσουμε τα επιμέρους ζητήματα που αφορούν την απόδοση ενός Web proxy. Όπως θα αναλύσουμε στο κεφάλαιο αυτό, τα προβλήματα αυτά αφορούν κυρίως το υποσύστημα μαγνητικών δίσκων το οποίο θέτει το άνω όριο της ταχύτητας σε ένα σύστημα Web proxy.

Στο κεφάλαιο 3 θα ασχοληθούμε διεξοδικά με το ερευνητικό πρόβλημα της εισόδου - εξόδου των δεδομένων ενός Web proxy στους μαγνητικούς δίσκους και θα μελετήσουμε τις προσεγγίσεις και τις λύσεις που έχουν προταθεί από άλλους ερευνητές.

Στο κεφάλαιο 4 θα παρουσιάσουμε το θεωρητικό υπόβαθρο της παρούσας μεταπτυχιακής εργασίας, αναφέροντας τους μηχανισμούς που αναπτύχθηκαν [62] για την αύξηση της ταχύτητας εισόδου - εξόδου δεδομένων στους δίσκους ενός Web proxy.

Στο κεφάλαιο 5 περιγράφουμε την αρχιτεκτονική και την υλοποίηση του δικού μας Web proxy, τον οποίο ονομάσαμε "FOXY". Παράλληλα αναφερόμαστε στα προβλήματα που αντιμετωπίσαμε στην προσπάθειά μας αυτή καθώς και τις προτεινόμενες λύσεις. Τέλος στο κεφάλαιο αυτό δίνουμε λεπτομερείς πληροφορίες για τη χρήση του "FOXY" από ερευνητές που θα ήθελαν να το δοκιμάσουν.

Στο κεφάλαιο 6 παρουσιάζουμε τα αποτελέσματα των πειραμάτων μας με το "FOXY" και τη σύγκρισή του με το Squid.

Τέλος, στο κεφάλαιο 7 παρουσιάζουμε τα συμπεράσματα της μεταπτυχιακής εργασίας και προτείνουμε κατευθύνσεις για μελλοντική έρευνα στον τομέα της απόδοσης του Παγκόσμιου Ιστού.

# Συστήματα WWW Caching

Never underestimate the bandwidth of a station wagon full of tapes hurtling down the highway.

—Andrew Tannenbaum

Στο κεφάλαιο αυτό παρουσιάζουμε το θέμα του WWW Caching αρχίζοντας από τη θεμελιώδη λειτουργία του WWW και μια περιληπτική παρουσίαση του πρωτοκόλλου HTTP. Στη συνέχεια αναλύουμε τα προβλήματα που παρουσιάζει η λειτουργία του Παγκόσμιου Ιστού και δείχνουμε γιατί η χρήση ενδιάμεσης μνήμης είναι η σωστή λύση στα προβλήματα αυτά.

## 2.1 ΤΟ ΠΡΩΤΟΚΟΛΛΟ HTTP

Η λειτουργία του Παγκόσμιου Ιστού βασίζεται στο μοντέλο Πελάτη - Εξυπηρέτη (Client - Server model). Στο μοντέλο αυτό ο Πελάτης και ο Εξυπηρέτης είναι προγράμματα τα οποία τρέχουν σε δύο διαφορετικά μηχανήματα τα οποία συνδέονται μέσω του Διαδικτυακού πρωτοκόλλου TCP/IP (Transfer Control Protocol / Internet Protocol). Στην υπηρεσία WWW το πρόγραμμα του πελάτη λέγεται WWW browser και είναι το πρόγραμμα που μπορεί να διαβάζει δεδομένα hypertext σε μορφή HTML και να τα δείχνει στην οθόνη του χρήστη. Αντίθετα το πρόγραμμα του Εξυπηρέτη λέγεται WWW server και η σκοπός του είναι να εξυπηρετεί τους WWW browsers που συνδέονται με αυτόν δίνοντάς τους τα hypertext documents που του ζητάνε.

Οι WWW browsers (που αναφέρονται πολλές φορές και σαν WWW clients) και οι WWW servers επικοινωνούν μεταξύ τους χρησιμοποιώντας το πρωτόκολλο HTTP (HyperText Transfer Protocol) [99], [88]. Πρόκειται για ένα απλό πρωτόκολλο που καθορίζει τον τρόπο που ο WWW browser ζητάει τα documents από τον WWW server και τον τρόπο που ο τελευταίος του τα παραδίδει ή του γνωστοποιεί ότι η αίτηση που έγινε ήταν λάθος. Στην παράγραφο αυτή θα περιγράφουμε περιληπτικά τη βασική ορολογία που χρησιμοποιούμε σχετικά με τον Παγκόσμιο Ιστό και τη λειτουργία του πρωτοκόλλου HTTP.

Παρακάτω παραθέτουμε κάποια ορολογία που θα χρησιμοποιήσουμε για την περιγραφή του πρωτοκόλλου HTTP [88]:

◇ **URI :** Ακρώνυμο του όρου Uniform Resource Identifier. Το URI είναι το γενικό σύνολο από όλα τα ονόματα και τις διευθύνσεις που είναι ακολουθίες χαρακτήρων και αφορούν διαδικτυακά αντικείμενα (Internet objects). Οι πιο συνηθισμένες μορφές URI είναι τα URLs τα οποία περιγράφουμε αμέσως παρακάτω.

◇ **URL :** Ακρώνυμο του όρου Uniform Resource Locator. Το URL είναι η διεύθυνση για ένα αντικείμενο (object) στον Παγκόσμιο Ιστό, ή πιο απλά μία ακολουθία χαρακτήρων (string) που περιγράφει επαρκώς και ακριβώς σε ένα πρόγραμμα Web browser πως θα εντοπίσει ένα αντικείμενο (το οποίο μπορεί να είναι ένα έγγραφο, αρχείο, υπηρεσία κτλ.) στο WWW. Κάθε αντικείμενο στον Παγκόσμιο Ιστό πρέπει να περιγράφεται από ένα URL, έτσι ώστε να μπορεί να εντοπιστεί από έναν Web browser. Επίσης χρησιμοποιούνται μέσα στα έγγραφα HTML για να προσδιορίσουν τους συνδέσμους (hyperlinks) μεταξύ δύο εγγράφων HTML.

Μερικά παραδείγματα από URLs φαίνονται παρακάτω :

▷ http://www.w3.org/default.html

▷ http://foldoc.doc.ic.ac.uk/?Uniform+Resource+Locator

▷ http://www.w3.org/default.html#Introduction

▷ ftp://spy:secret@ftp.acme.com/pub/topsecret/weapon.tgz

▷ mailto:dbh@doc.ic.ac.uk

▷ telnet://dra.com

Το πρώτο μέρος του URL βρίσκεται πριν από το χαρακτήρα ":" και προσδιορίζει τον τρόπο πρόσβασης ή το πρωτόκολλο που πρέπει να χρησιμοποιηθεί ώστε να προσπελάσουμε το αντικείμενο. Τα πιο συχνά πρωτόκολλα είναι: ftp, http (World Wide Web), gopher, WAIS ή file. Το τελευταίο χρησιμοποιείται όταν θέλουμε να αναφερθούμε σε ένα αρχείο που βρίσκεται στον ίδιο κόμβο (host). Μερικά λιγότερο συχνά πρωτόκολλα είναι τα : news, telnet or mailto (E-mail).

Η σημασία του τμήματος μετά το χαρακτήρα ":" προσδιορίζεται ανάλογα με το πρωτόκολλο. Γενικά δύο χαρακτήρες "/" προσδιορίζουν το όνομα του κόμβου (hostname), ενώ χρησιμοποιούνται και οι μορφές "hostname:port" ή για το πρωτόκολλο FTP τα user:passwd@hostname:port or user@hostname:port κτλ. Το επόμενο τμήμα ενός HTTP ή FTP URL αντιστοιχεί στη διαδρομή ενός αρχείου μέσα στο σύστημα αρχείων του κόμβου. Τέλος, το τελευταίο (μη υποχρεωτικό) μέρος του URL μπορεί να είναι μία ακολουθία χαρακτήρων αναζήτησης η οποία αρχίζει με το χαρακτήρα "?" ή μία ετικέτα ενός συγκεκριμένου κομματιού ενός εγγράφου HTML η οποία θα ξεκινάει με το χαρακτήρα "#".

◇ **Σύνδεση (connection) :** Μία δίοδος επικοινωνίας μέσω του διαδικτυακού πρωτοκόλλου TCP/IP (Transfer Control Protocol / Internet Protocol) ανάμεσα σε δύο προγράμματα.

◇ **Μήνυμα (message) :** Η βασική δομή της επικοινωνίας του πρωτοκόλλου HTTP που αποτελείται από μια ακολουθία από bytes δομημένη όπως περιγράφει το πρωτόκολλο που μεταδίδεται μέσω της σύνδεσης.

◇ **Αίτηση HTTP (request) :** Ένα μήνυμα αίτησης HTTP (HTTP request message). Η δομή των μηνυμάτων αίτησης HTTP περιγράφεται παρακάτω.

◇ **Απόκριση HTTP (response) :** Ένα μήνυμα απόκρισης HTTP (HTTP response message). Η δομή των μηνυμάτων απόκρισης HTTP περιγράφεται παρακάτω.

◇ **Πελάτης WWW (WWW client) :** Ο Πελάτης WWW είναι ένα πρόγραμμα που δημιουργεί συνδέσεις με σκοπό την αποστολή αιτήσεων. Στο εξής όταν μιλάμε για τον Πελάτη θα εννοούμε Πελάτης WWW εκτός αν αναφέρεται διαφορετικά.

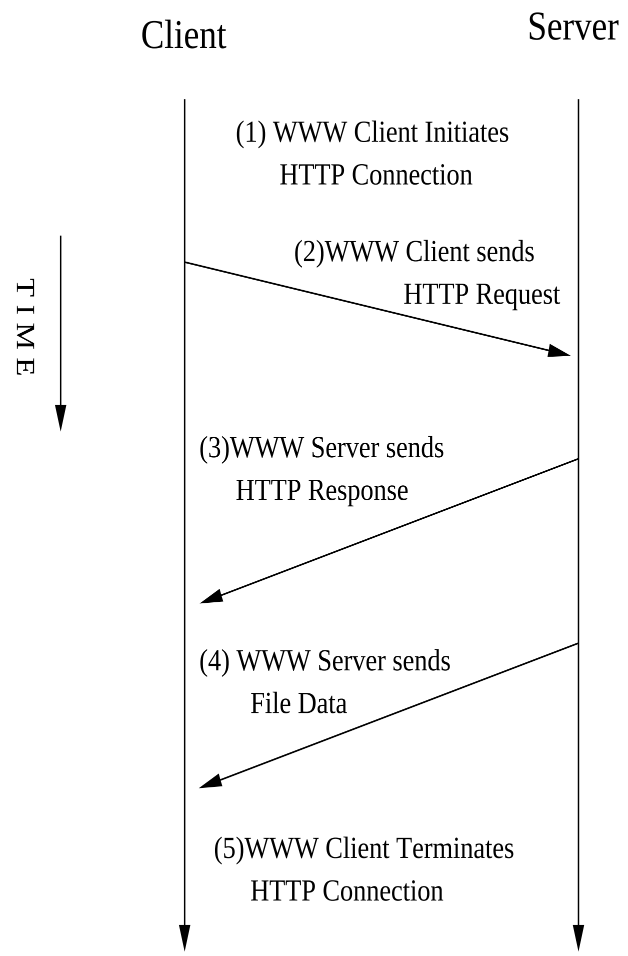
◇ **Πράκτορας Χρήστη (user agent) :** Ο Πελάτης που ξεκινάει μία αίτηση. Οι πράκτορες χρήστη είναι τις περισσότερες φορές οι WWW browsers (Netscape, Internet Explorer, Opera, Mosaic, Lynx κτλ.) γι' αυτό και δεν θα χρησιμοποιούμε τον όρο αυτό πολύ συχνά. Πολλές φορές όμως οι πράκτορες χρήστη μπορεί να είναι και άλλα προγράμματα όπως κειμενογράφοι, αυτόματα προγράμματα αρχειοθέτησης εγγράφων Web, κτλ.

◇ **Εξυπηρέτης WWW (WWW server) :** Ο Εξυπηρέτης WWW είναι ένα πρόγραμμα που δέχεται συνδέσεις με σκοπό να τις εξυπηρετήσει στέλνοντας πίσω αποκρίσεις. Στο εξής όταν μιλάμε για τον Εξυπηρέτη θα εννοούμε Εξυπηρέτη WWW εκτός αν αναφέρεται διαφορετικά. Οι πιο δημοφιλείς Εξυπηρέτες WWW σήμερα είναι ο Apache [7], ο MS IIS, ο Netscape WWW Server, κ.ά.

Ένα πρόγραμμα μπορεί να είναι ταυτόχρονα Πελάτης και Εξυπηρέτης. Η χρήση της ορολογία αυτής του Πελάτη και του Εξυπηρέτη έχει σχέση με την περιγραφή του ρόλου του προγράμματος σε μία συγκεκριμένη σύνδεση και όχι με τις δυνατότητες του προγράμματος γενικότερα [88].

Οι συνδέσεις του πρωτοκόλλου HTTP γίνονται συνήθως στο port 80 του Εξυπηρέτη, αλλά δεν αποκλείεται να γίνουν και σε άλλα ports όπως μπορεί να καθοριστεί στο URI.

Το πρωτόκολλο HTTP είναι ένα πρωτόκολλο αίτησης - απόκρισης. Στην έκδοση 1.0 του πρωτοκόλλου ανοίγεται μία καινούργια σύνδεση για κάθε αίτηση του πελάτη και απόκριση του Εξυπηρέτη. Αν χρειαστεί να ξαναγίνει αίτηση στον ίδιο Εξυπηρέτη ανοίγεται νέα σύνδεση. Στην έκδοση 1.1 του πρωτοκόλλου, μία σύνδεση μπορεί να χρησιμοποιηθεί για μία ή περισσότερες αιτήσεις και αποκρίσεις ανάμεσα στον ίδιο πελάτη και Εξυπηρέτη. Αυτό, όπως θα δούμε παρακάτω μπορεί να βελτιώσει κατά πολύ την ταχύτητα της εξυπηρέτησης πελατών από την Εξυπηρέτη.




**Σχήμα 2.1** Ακολουθία ανταλλαγής μηνυμάτων ανάμεσα στον Εξυπηρέτη και τον Πελάτη σύμφωνα με το πρωτόκολλο HTTP 1.0 για τη μεταφορά ενός αντικειμένου WWW.

Η λειτουργία του πρωτοκόλλου HTTP φαίνεται στο σχήμα 2.1 όπου γίνονται τα εξής :

1. Ο Πελάτης ανοίγει μία νέα σύνδεση με τον Εξυπηρέτη, ή αν υποστηρίζει την έκδοση 1.1 του πρωτοκόλλου HTTP και υπάρχει ήδη ανοικτή σύνδεση μπορεί να την χρησιμοποιήσει.

2. Ο Πελάτης στέλνει μέσω της σύνδεσης μία αίτηση στον Εξυπηρέτη σε μια μορφή που περιγράφει : την μέθοδο αίτησης, το URI του εγγράφου και την έκδοση του πρωτοκόλλου HTTP καθώς και άλλες μη-υποχρεωτικές πληροφορίες του πελάτη (όπως την έκδοση του browser ή σε ποια γλώσσα προτιμάει το έγγραφο κτλ.)[88].

   Πιο συγκεκριμένα η πρώτη γραμμή της αίτησης που είναι υποχρεωτική έχει τη μορφή [88] :

   ◇ Request-Line = Method SP Request-URI SP HTTP-Version CRLF [1]

   όπου τα μέρη Method, Request-URI και HTTP-Version έχουν τη παρακάτω μορφή :

   ◇ Method = "OPTIONS" | "GET" | "HEAD" | "POST" | "PUT" | "DELETE" | "TRACE" | "CONNECT" | extension-method

   ◇ Request-URI = absolute-URI | relative-URI
   (π.χ. absolute-URI = http://www.w3.org/WWW/TheProject.html, relative-URI = /WWW/TheProject.html).

   ◇ HTTP-Version = HTTP / <version number>
   (π.χ. HTTP/1.1 ή HTTP/1.0)

   Η Μέθοδος (Method) που χρησιμοποιείται συχνότερα από όλες είναι η "GET" που ζητάει ένα WWW έγγραφο από τον Εξυπηρέτη. Για τους σκοπούς της εργασίας αυτής δεν κρίνουμε σκόπιμο να περιγράψουμε όλες τις μεθόδους, αρκεί ο αναγνώστης να καταλάβει τη μορφή των αιτήσεων. Για παράδειγμα δύο τέτοιες γραμμές αίτησης που ζητάνε από τον Εξυπηρέτη που τρέχει στον κόμβο www.w3.org το έγγραφο που βρίσκεται στο αρχείο με τη διαδρομή /WWW/TheProject.html σύμφωνα με το πρωτόκολλο HTTP έκδοσης 1.1, είναι οι ακόλουθες :

   ◇ GET http://www.w3.org/WWW/TheProject.html HTTP/1.1 (Absolute-URI)

   ◇ GET /WWW/TheProject.html HTTP/1.1 (Relative-URI)

[1] Το σύμβολο SP συμβολίζει τον κενό χαρακτήρα (space), ενώ το σύμβολο CRLF συμβολίζει τον χαρακτήρα αλλαγής γραμμής.

Η γραμμή αίτησης μπορεί να ακολουθείται από άλλες γραμμές που περιέχουν κάποιον από τους παρακάτω headers οι οποίοι έχουν ειδική σημασία (π.χ. Cache-Control, User-Agent, If-Modified-Since, If-Unmodified-Since, Pragma, Host, Transfer-Encoding, Proxy-Authorization, Via, Accept-Encoding, If-Match κτλ.). Μία ολοκληρωμένη λοιπόν αίτηση HTTP είναι για παράδειγμα αυτή που φαίνεται στον πίνακα 2.1.

| |
|---|
| GET /WWW/TheProject.html HTTP/1.1 |
| Host: www.w3.org |
| Proxy-Connection: Keep-Alive |
| User-Agent: Mozilla/4.7 [en] (X11; I; Linux 2.2.12-20 i686) |
| Pragma: no-cache |
| Accept: image/gif, image/x-xbitmap, image/jpeg, image/png, */* |
| Accept-Encoding: gzip |
| Accept-Language: en |
| Accept-Charset: iso-8859-1,*,utf-8 |
| X-Xact: 02249457.53ee0d5c:00000032 |
| X-Oid-Shortage: 16 |

**Πίνακας 2.1** Αίτηση HTTP.

3. Ο Εξυπηρέτης, αφού λάβει και αποκωδικοποιήσει την αίτηση HTTP, απαντάει με μία γραμμή απόκρισης που έχει τη μορφή :

   ◇ Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

   Τη μορφή του HTTP-Version την είδαμε παραπάνω, ενώ το Status-Code (Κωδικός Κατάστασης) είναι ένας τριψήφιος κωδικός που αντιστοιχεί στην κατάσταση της απόκρισης του Εξυπηρέτη που είναι και η αντίστοιχη Reason-Phrase. Στον παρακάτω πίνακα φαίνονται οι συχνότεροι τριψήφιοι κωδικοί μαζί με τις αντίστοιχες καταστάσεις τους.

   Σύμφωνα λοιπόν με τα παραπάνω σωστές γραμμές απόκρισης του Εξυπηρέτη μπορεί να είναι για παράδειγμα οι :

   ◇ HTTP/1.1 100 OK (Η αίτηση ήταν έγκυρη και ο Εξυπηρέτης θα στείλει στη συνέχεια το ζητούμενο έγγραφο).

   ◇ HTTP/1.1 400 Bad Request (Η αίτηση δεν ήταν σωστή και άρα απέτυχε).

   ◇ HTTP/1.1 404 Not Found (Ο Εξυπηρέτης δεν μπορεί να βρει το ζητούμενο έγγραφο και άρα η αίτηση απέτυχε).

| Κωδικός | Κατάσταση |
|---|---|
| "100" | Continue |
| "101" | Switching Protocols |
| "200" | OK |
| "201" | Created |
| "202" | Accepted |
| "300" | Multiple Choices |
| "301" | Moved Permanently |
| "302" | Found |
| "304" | Not Modified |
| "305" | Use Proxy |
| "400" | Bad Request |
| "401" | Unauthorized |
| "403" | Forbidden |
| "404" | Not Found |

**Πίνακας 2.2** Τριψήφιοι κωδικοί κατάστασης Εξυπηρέτη και οι αντίστοιχες καταστάσεις τους.

Μαζί με τη γραμμή απόκρισης ο Εξυπηρέτης μπορεί να στείλει και άλλες πληροφορίες σχετικά με το ζητούμενο έγγραφο (π.χ. πεδία όπως Cache-Control, Date, Content-Length, Last-Modified κτλ.) Μία ολοκληρωμένη δηλαδή απόκριση του Εξυπηρέτη μπορεί να είναι για παράδειγμα αυτή που φαίνεται στον πίνακα 2.3.

| |
|---|
| HTTP/1.0 200 OK |
| Cache-Control: public |
| Date: Wed, 19 Jan 2000 15:42:32 GMT |
| Expires: Fri, 18 Jan 2002 15:42:32 GMT |
| Last-Modified: Tue, 19 Jan 1999 15:19:06 GMT |
| Content-Length: 5593 |
| X-Xact: 02249457.53ee0d5c:7fffffcf |
| X-Srv-Wid: 0224944d.42e10d5a:00000002 |
| X-Srv-Oids: 6:177.16 |

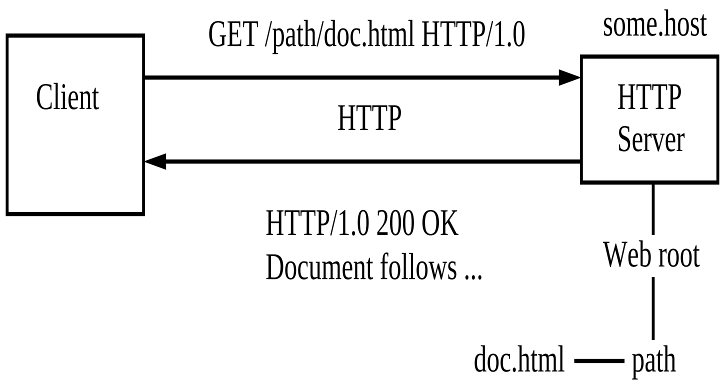
**Πίνακας 2.3** Απόκριση HTTP.

4. Στη συνέχεια μετά την απόκριση, και αν η αίτηση είναι επιτυχής ο Εξυπηρέτης στέλνει το ζητούμενο έγγραφο στον Πελάτη και η σύνδεση τερματίζεται (εκτός

αν ο Πελάτης ζητήσει και άλλο έγγραφο οπότε ή θα ανοίξει καινούργια σύνδεση ή αν υποστηρίζει την έκδοση 1.1 του πρωτοκόλλου HTTP θα χρησιμοποιήσει μια υπάρχουσα σύνδεση).

## 2.2 Η ΛΕΙΤΟΥΡΓΙΑ ΤΩΝ ΣΥΣΤΗΜΑΤΩΝ WWW CACHING

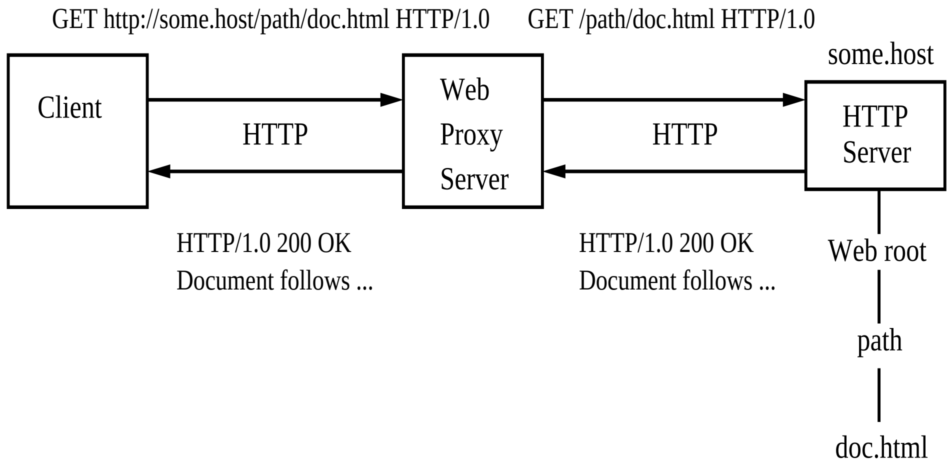
Τα συστήματα ενδιάμεσης μνήμης του Παγκόσμιου Ιστού (Web proxies) συμπεριφέρονται σαν μεσάζοντες στις συνομιλίες ανάμεσα στα συστήματα Πελατών και Εξυπηρετών που γίνονται μέσω του πρωτοκόλλου HTTP, τη λειτουργία του οποίου είδαμε παραπάνω. Στο σχήμα 2.2 φαίνεται μία συνομιλία HTTP όπως θα γινόταν χωρίς τη χρήση ενός Web proxy. Παρατηρούμε ότι ο Πελάτης συνδέεται κατευθείαν με τον Εξυπηρέτη, από όπου ανακτά τη ζητούμενη ιστοσελίδα χρησιμοποιώντας το πρωτόκολλο HTTP.



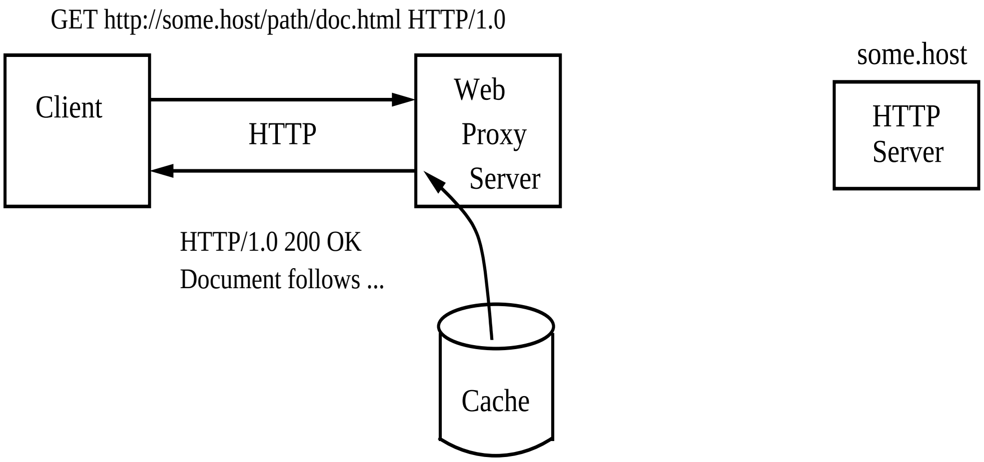

**Σχήμα 2.2 Μία άμεση συνομιλία HTTP ανάμεσα στον Πελάτη και τον Εξυπηρέτη.**

Αντίθετα στο σχήμα 2.3 φαίνεται μια συνομιλία HTTP ανάμεσα στον Πελάτη και τον Εξυπηρέτη με χρήση συστήματος ενδιάμεσης μνήμης. Όπως μπορεί κανείς να παρατηρήσει, ο Πελάτης στέλνει την αίτηση στον Web proxy δίνοντας ολόκληρο το URL της ιστοσελίδας που αναζητά. Το Web proxy με τη σειρά του συνδέεται στον Εξυπηρέτη και στέλνει μια αίτηση HTTP όπως θα την έκανε ο Πελάτης σε μία απευθείας συνομιλία του με τον Εξυπηρέτη. Η απάντηση του Εξυπηρέτη επιστρέφεται στον Πελάτη και, αν κριθεί σκόπιμο, αποθηκεύεται στην μνήμη του Web proxy.

Μελλοντικές αιτήσεις για την ίδια ιστοσελίδα θα εξυπηρετηθούν γρήγορα από τη μνήμη του Web proxy όπως φαίνεται στο σχήμα 2.4. Όπως παρατηρούμε στο σχήμα αυτό, το Web proxy δεν συνδέεται καθόλου με τον Εξυπηρέτη της ιστοσελίδας κι έτσι μειώνει την κίνηση προς το έξω δίκτυο.




**Σχήμα 2.3 Μία συνομιλία HTTP ανάμεσα στον Πελάτη και τον Εξυπηρέτη με χρήση συστήματος ενδιάμεσης μνήμης.**




**Σχήμα 2.4 Μία αίτηση HTTP του Πελάτη που εξυπηρετείται γρήγορα από τη μνήμη του Web proxy.**

### 2.2.1 Η ΘΕΣΗ ΤΟΥ WEB PROXY

Ένα ερώτημα που ίσως γεννηθεί στον αναγνώστη από τα σχήματα 2.3 και 2.4 είναι πως ξέρει ο Πελάτης (συνήθως ο Web browser ενός χρήστη) ότι υπάρχει ένα Web proxy διαθέσιμο και πως μπορεί να το χρησιμοποιήσει. Υπάρχουν συνήθως τρία είδη εγκαταστάσεων ενός Web proxy σε ένα δίκτυο [91], [17]:

- ◇ **Απλό Proxy Caching**

  Σ' αυτού του είδους τις εγκαταστάσεις κάθε χρήστης πρέπει να γνωρίζει την ύπαρξη του Web proxy, τη διεύθυνση IP του και το port στο οποίο δέχεται αιτήσεις και να ορίσει αυτές τις παραμέτρους στον Web browser του. Όπως είναι κατανοητό, αυτό αποτελεί ένα μειονέκτημα αυτού του είδους εγκατάστασης, μαζί με το γεγονός ότι πολλοί χρήστες δεν χρησιμοποιούν το Web proxy είτε γιατί δεν γνωρίζουν την ύπαρξη και το ρόλο του, είτε γιατί δεν γνωρίζουν πώς πρέπει να ορίσουν τις παραμέτρους στον Web browser τους.

◇ **Διαφανές (Transparent) Proxy Caching**

Στις εγκαταστάσεις αυτές ο χρήστης δεν χρειάζεται να γνωρίζει τίποτα για την ύπαρξη και τη λειτουργία του Web proxy. Ο διαφανής proxy είναι εγκατεστημένος σε κάποιο κεντρικό σημείο του δικτύου (δρομολογητή, πύλη ή μεταγωγέα δικτύου) από όπου περνάνε όλες οι αιτήσεις των χρηστών για ιστοσελίδες. Με αυτόν τον τρόπο μπορεί να τις παρακολουθεί και να τις εξυπηρετεί όλες χωρίς να το γνωρίζουν οι χρήστες.

Τα κύρια μειονεκτήματα του διαφανούς proxy είναι τρία :

- ▷ Η εγκατάσταση διαφανούς proxy προϋποθέτει την ύπαρξη ενός μόνο σημείου από όπου περνάει η κίνηση του Παγκόσμιου Ιστού. Συνεπώς σε πολλά δίκτυα χρειάζονται αλλαγές στην τοπολογία του δικτύου προκειμένου να εγκατασταθεί ένα διαφανές proxy [17].
- ▷ Το διαφανές proxy δημιουργεί πρόβλημα διαθεσιμότητας, όταν για κάποιο λόγο διακοπεί η λειτουργία του. Τότε όλοι οι χρήστες του εσωτερικού δικτύου χάνουν την πρόσβαση την υπηρεσία του Παγκόσμιου Ιστού [91].
- ▷ Η λειτουργία του διαφανούς proxy θα έχει κατά πάσα πιθανότητα αρνητική επίδραση στην κίνηση του δικτύου εκτός της υπηρεσίας WWW γιατί πρέπει να φιλτράρει όλη την κίνηση με σκοπό να ανιχνεύει την κίνηση WWW. Επίσης το γεγονός ότι πρέπει να έχει μεγάλη υπολογιστική ισχύ για να εξετάζει όλα τα πακέτα δεδομένων που περνάνε προς το έξω δίκτυο, σημαίνει ότι το διαφανές proxy θα έχει μεγάλο κόστος [91].
- ▷ Η ανίχνευση των αιτήσεων HTTP λειτουργεί μόνο για το port 80 στο οποίο δουλεύουν συνήθως οι Εξυπηρέτες WWW.

◇ **Διαφανές Proxy Caching με Ανακατεύθυνση Αιτήσεων**

Στις περιπτώσεις αυτές, δεν χρειάζεται να εγκατασταθεί ο Web proxy σε κάποιο κεντρικό σύστημα για να ελέγχει όλη την κίνηση. Εγκαθίσταται σε κάποιο άλλο σημείο του δικτύου και σε ένα ή περισσότερα κεντρικά δικτυακά σημεία (δρομολογητές, πύλες ή μεταγωγείς δικτύου) εγκαθίστανται συστήματα που ανιχνεύουν την κίνηση WWW και την ανακατευθύνουν προς το Web proxy. Με αυτόν τον τρόπο έχουμε όλα τα πλεονεκτήματα της περίπτωσης του διαφανούς Web proxy και εξουδετερώνουμε τα περισσότερα από τα αρνητικά του σημεία.

Έτσι δεν ισχύει το μειονέκτημα της διαθεσιμότητας, αφού το σύστημα ανακατεύθυνσης αιτήσεων αν καταλάβει διακοπή στη λειτουργία του Web proxy μπορεί να στέλνει τις αιτήσεις WWW προς το έξω δίκτυο. Επίσης εφόσον το Web proxy δε λειτουργεί πάνω στο κεντρικό κόμβο του δικτύου, η επεξεργασία των πακέτων από τον κεντρικό αυτό κόμβο είναι πολύ ταχύτερη [91]. Από την άλλη μεριά υπάρχει πάλι το μειονέκτημα ότι η ανίχνευση των αιτήσεων HTTP λειτουργεί μόνο για το port 80.

Πρόσφατα δημιουργήθηκε ένα νέο πρωτόκολλο, το *Web Proxy Autodiscovery Protocol (WPAD)* [26] από τις εταιρείες Inktomi, Real Networks, Sun Microsystems και Microsoft και υποβλήθηκε στην *Internet Engineering Task Force IETF* [42], [26]. Το πρωτόκολλο αυτό, που πρόκειται να ενσωματωθεί στους Web browsers και στα υπόλοιπα προγράμματα Πελατών του Διαδικτύου, είναι σχεδιασμένο να ανακαλύπτει αυτόματα τις τυχόν Web Caches που υπάρχουν σε ένα δίκτυο και να τις χρησιμοποιεί χωρίς παρέμβαση από τους χρήστες. Τότε θα προστεθεί μία καλύτερη εναλλακτική λύση σε αυτές που περιγράψαμε παραπάνω και οι οποίες προϋποθέτουν πάντα τη βοήθεια εξαρτημάτων του δικτύου όπως δρομολογητές, μεταγωγοί, firewalls κτλ. [42]

### 2.2.2 ΕΝΗΜΕΡΩΣΗ ΤΟΥ ΠΕΡΙΕΧΟΜΕΝΟΥ ΕΝΟΣ WEB PROXY

Ένα δεύτερο ερώτημα που ίσως γεννηθεί στον αναγνώστη από στο σχήμα 2.4 είναι πώς γνωρίζει ο Web proxy ότι το WWW αντικείμενο στον Εξυπηρέτη δεν έχει αλλάξει, εφόσον δεν συνδέεται με αυτόν για να ενημερωθεί. Το ερώτημα είναι λογικό γιατί η πλειοψηφία των αντικείμενων του Παγκόσμιου Ιστού αλλάζει με το χρόνο, άλλα πιο γρήγορα κι άλλα πιο αργά [32], [31]. Συνεπώς το Web proxy πρέπει να ενημερώνεται για το πόσο πρόσφατα είναι τα αντικείμενα που βρίσκονται στη μνήμη του και να αντικαθιστά όσα έχουν αλλάξει [65], [31]. Μερικοί ερευνητές μάλιστα υποστηρίζουν ότι ο χρόνος ενημέρωσης των αντικειμένων πρέπει να λαμβάνεται υπόψη στην πολιτική αντικατάστασης των αντικειμένων στη μνήμη του Web proxy και έχουν κατασκευάσει τέτοιες πολιτικές [86], [75].

Τα περισσότερα Web proxies σήμερα χρησιμοποιούν έναν από τους παρακάτω τρεις τρόπους για την ενημέρωση τους [17]:

◇ Στέλνουν μία αίτηση “get if modified” (δηλαδή “φέρε αν έχει αλλάξει”) στον μακρινό Εξυπηρέτη για κάθε αίτηση από τους Πελάτες με την ημερομηνία που έχει το αντικείμενο στη μνήμη του Web proxy. Αν το αντικείμενο στον μακρινό Εξυπηρέτη έχει αλλάξει από τότε, γίνεται ενημέρωση με το καινούριο, διαφορετικά ο Εξυπηρέτης απαντάει ότι το αντικείμενο είναι ακόμα έγκυρο και το Web proxy το επιστρέφει στον Πελάτη από τη μνήμη του. Όπως είναι κατανοητό αυτή η μέθοδος είναι η πιο ασφαλής για να είναι τα αντικείμενα πάντα ενημερωμένα, αλλά έχει το μειονέκτημα της καθυστέρησης γιατί κάθε φορά γίνεται σύνδεση με το μακρινό Εξυπηρέτη.

◇ Χρησιμοποιούν δεδομένα "εγκυρότητας" όπως για το πόσο έγκυρα είναι τα αντικείμενα που έχουν στη μνήμη τους. Αυτά τα δεδομένα περιέχονται συνήθως στις αποκρίσεις HTTP του Εξυπηρέτη όταν δίνει ένα αντικείμενο και αναφέρουν στο Web proxy για πόσο χρόνο είναι έγκυρο το αντικείμενο αυτό.

◇ Χρησιμοποιούν ευριστικές μεθόδους (heuristics) για να κρίνουν τη "διάρκεια εγκυρότητας" των αντικειμένων στη μνήμη τους βασισμένες συνήθως στο χρόνο που έχει περάσει από την τελευταία μεταβολή του αντικειμένου [10]. Έτσι, όταν το Web proxy φέρνει ένα αντικείμενο από τον Εξυπηρέτη, παρατηρεί το χρόνο τελευταίας μεταβολής του αντικειμένου και υποθέτει ότι το αντικείμενο έχει ένα μέγιστο χρόνο εγκυρότητας ίσο με ένα σταθερό ποσοστό (π.χ. 10%) του χρόνου που έχει περάσει από την τελευταία αλλαγή του [17], [10].

Ο τελευταίος τρόπος με τις ευριστικές μεθόδους (heuristics) είναι ο πιο δημοφιλής σήμερα γιατί μειώνει την καθυστέρηση των αιτήσεων, αφού το Web proxy δεν χρειάζεται να συνδεθεί με το μακρινό Εξυπηρέτη [17], [44], [10]. Βέβαια μερικές φορές η υπόθεση που γίνεται με τις ευριστικές μεθόδους δεν ισχύει και το Web proxy δίνει μη έγκυρα αντικείμενα στους πελάτες. Γι' αυτό πολλές εταιρείες σήμερα προωθούν την μέθοδο του "active caching". Η βασική ιδέα της μεθόδου αυτής είναι ένας συνδυασμός των ευριστικών μεθόδων και της ενημέρωσης των αντικειμένων από τον Εξυπηρέτη όταν το Web proxy δεν έχει φόρτο, π.χ. τις νυχτερινές ώρες [44].

Τέλος ένα θέμα που απασχολεί έντονα την ερευνητική κοινότητα που ασχολείται με την απόδοση του Παγκόσμιου Ιστού είναι το θέμα των "δυναμικών αντικειμένων", δηλαδή των αντικειμένων του Παγκόσμιου Ιστού που δημιουργούνται δυναμικά την ώρα της αίτησης του χρήστη και είναι έγκυρα για τη συγκεκριμένη αίτηση μόνο [16]. Τέτοια αντικείμενα αφορούν για παράδειγμα ιστοσελίδες που δημιουργούνται από Εξυπηρέτες Ηλεκτρονικού Εμπορίου και περιέχουν π.χ. αντικείμενα που έχει αγοράσει ένας συγκεκριμένος χρήστης, ή ιστοσελίδες που αφορούν αναζητήσεις που έχει κάνει ένας χρήστης με κάποιο κλειδί. Το ποσοστό τέτοιου είδους αντικειμένων παρουσιάζει αύξηση σε σχέση με τα στατικά αντικείμενα που μπορούν να αποθηκευτούν στη μνήμη ενός Web proxy. Η αποθήκευση των δυναμικών αντικειμένων στη μνήμη αποφεύγεται στα σημερινά Web proxies χρησιμοποιώντας πληροφορίες που βρίσκονται στην απόκριση HTTP κάθε Εξυπηρέτη που μεταδίδει ένα τέτοιο αντικείμενο. Μελετώνται πάντως τεχνικές για τον πιο αποτελεσματικό χειρισμό αυτών των αντικειμένων [16], [32], [31], [33].

## 2.3 ΒΕΛΤΙΩΣΗ ΤΟΥ ΠΡΩΤΟΚΟΛΛΟΥ HTTP

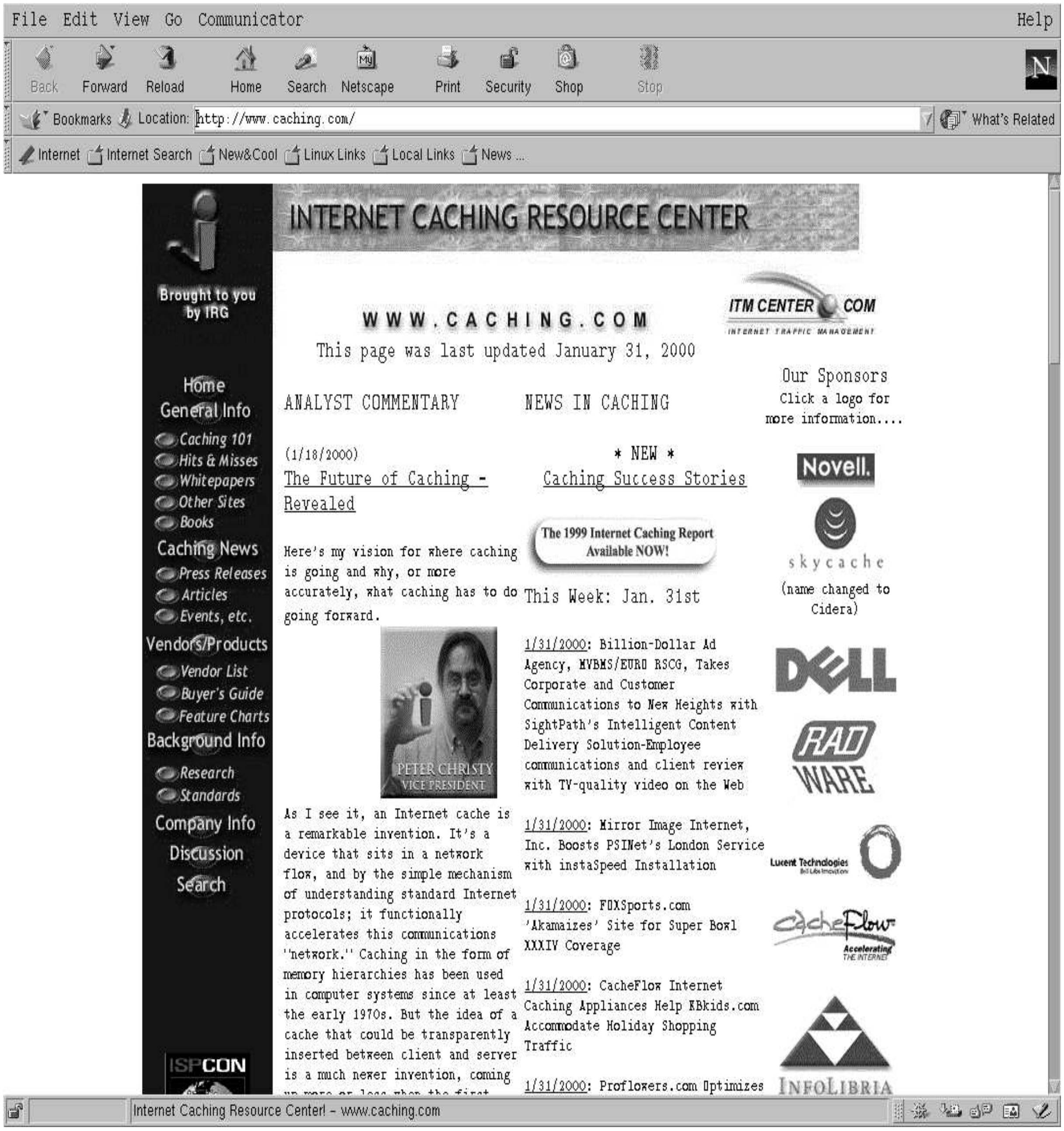
Στην παράγραφο αυτή θα δούμε την ανατομία μιας ιστοσελίδας και αργότερα θα εξετάσουμε συνολικά τις μεμονωμένες πράξεις που πρέπει να γίνουν για να μεταφερθεί ολόκληρη αυτή η ιστοσελίδα στον εκάστοτε χρήστη.

Όπως ίσως γνωρίζει ο αναγνώστης, μία ιστοσελίδα δεν αποτελείται από ένα μόνο αντικείμενο, π.χ. κείμενο σε μορφή HTML. Αποτελείται συνήθως από την αρχική σελίδα που είναι κείμενο σε μορφή HTML και αρκετά μικρά εικονίδια ή μεγαλύτερες εικόνες, applets, ήχους και διαφημίσεις. Καθένα από αυτά τα ενσωματωμένα αντικείμενα έχουν διαφορετικούς χρόνος ζωής αν αποθηκευτούν στη μνήμη ενός Web proxy. Για παράδειγμα τα εικονίδια που συνδέουν το περιεχόμενο μιας ιστοσελίδας με άλλες δεν αλλάζουν συχνά, ενώ οι διαφημίσεις μπορεί να αλλάζουν κάθε ώρα και άρα δεν έχει κανένα νόημα να τις αποθηκεύουμε στην μνήμη του Web proxy.

Ας δούμε για παράδειγμα την ιστοσελίδα που βρίσκεται στη διεύθυνση http://www.caching.com που φαίνεται στο σχήμα 2.5 (φαίνεται μόνο το πάνω μέρος της σελίδας. Αν δούμε τον κώδικα HTML της σελίδας αυτής θα παρατηρήσουμε ότι περιέχει 24 ενσωματωμένα εικονίδια, άλλα μικρά και άλλα μεγαλύτερα. Αν και η ιστοσελίδα δεν περιέχει διαφημίσεις, από τα αντικείμενα που βλέπουμε πιο πιθανό να αλλάξει είναι το κείμενο που θα προβάλλει τα άρθρα του επόμενου μήνα και η φωτογραφία του ρεπόρτερ, παρά τα στρογγυλά εικονίδια δεξιά που διακοσμούν τις επιλογές για το περιεχόμενο που μπορούμε να δούμε μέσα από τη συγκεκριμένη ιστοσελίδα.

Υπάρχει όμως ένας ακόμα παράγοντας που πρέπει να υπολογίσουμε όταν μετρήσουμε το κόστος μεταφοράς μιας ιστοσελίδας. Πριν από μία σύνδεση HTTP πρέπει να γίνει μία αναζήτηση DNS, δηλαδή να βρεθεί η IP διεύθυνση του Εξυπηρέτη από το hostname του. Αυτό γίνεται μέσω της υπηρεσίας DNS (Domain Name Service) μέσω μία σύνδεσης σε ένα Εξυπηρέτη DNS που συνήθως βρίσκεται στο τοπικό δίκτυο του Πελάτη. Παρ' όλα αυτά η αναζήτηση DNS μπορεί να πάρει αρκετά δευτερόλεπτα γιατί η σύνδεση συνήθως προωθείται σε πιο μακρινούς Εξυπηρέτες DNS. Βέβαια όταν ανακτούμε μια ιστοσελίδα, δεν χρειάζεται να κάνουμε αναζήτηση DNS για κάθε αντικείμενο της ιστοσελίδας αφού λογικά όλα ή τουλάχιστον τα περισσότερα από αυτά θα βρίσκονται στον ίδιο Εξυπηρέτη WWW. Αν πάντως κάποιο ή κάποια αντικείμενα βρίσκονται σε διαφορετικό Εξυπηρέτη WWW, θα χρειαστεί να ξανακάνουμε αναζήτηση DNS [43].

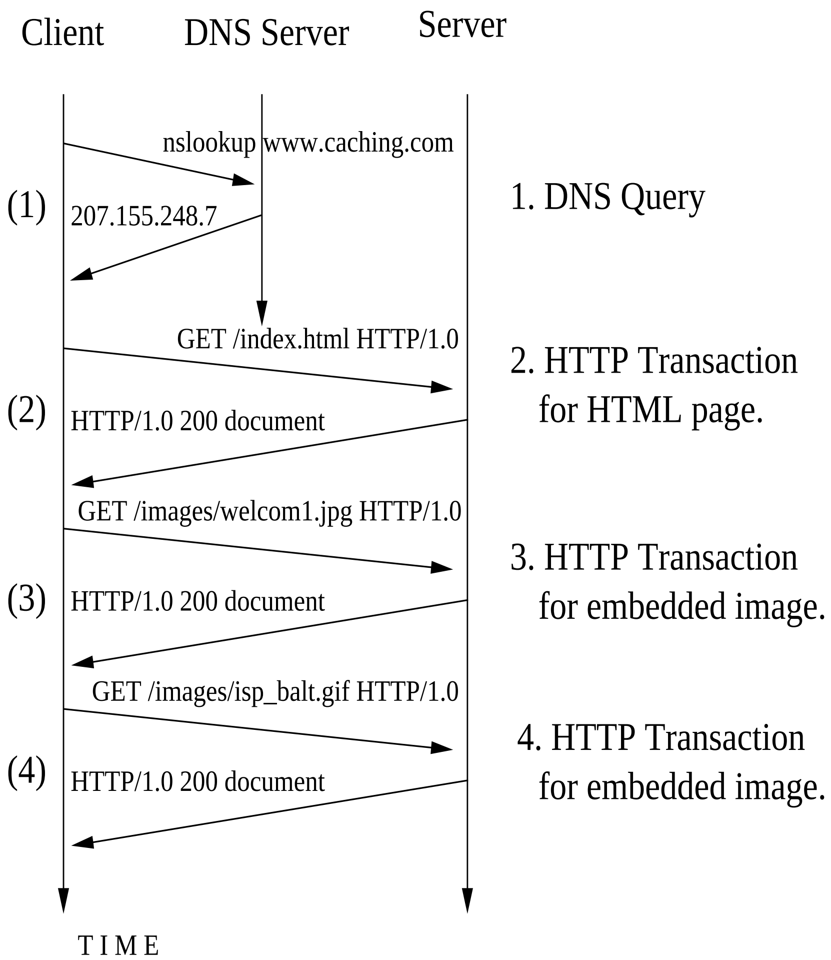
Δεδομένου λοιπόν ότι στη συντριπτική τους πλειοψηφία οι ιστοσελίδες αποτελούνται από πολλά ενσωματωμένα αντικείμενα, το κόστος ανάκτησης της ιστοσελίδας πρέπει να υπολογιστεί προσθέτοντας το κόστος που έχουμε όταν φέρνουμε καθένα από αυτά τα αντικείμενα ξεχωριστά όπως γίνεται σήμερα με την έκδοση 1.0 του πρωτοκόλλου HTTP καθώς και το κόστος της αναζήτησης DNS.




**Σχήμα 2.5** Η ιστοσελίδα στη διεύθυνση http://www.caching.com. Η συγκεκριμένη ιστοσελίδα εκτός από το κείμενο σε μορφή HTML περιέχει και 24 ενσωματωμένα εικονίδια.

Στο σχήμα 2.6 φαίνεται μια πιο ολοκληρωμένη ακολουθία για μια ανάκτηση ιστοσελίδας σύμφωνα με το πρωτόκολλο HTTP έκδοσης 1.0. Όπως μπορεί να παρατηρήσει ο αναγνώστης η ακολουθία ξεκινάει με μία αναζήτηση DNS που ακολουθείται από συναλλαγές HTTP για κάθε αντικείμενο της ιστοσελίδας. Κάθε μία από αυτές τις συναλλαγές χρειάζεται μία σύνδεση TCP/IP ανάμεσα στον Πελάτη και στον Εξυπηρέτη.

Η δημιουργία της σύνδεσης μπορεί να διαρκέσει αρκετά δευτερόλεπτα γιατί συνήθως περνάει μέσα από αρκετούς ενδιάμεσους δρομολογητές ή μεταγωγείς του Διαδικτύου μέχρι να φτάσει στο μακρινό Εξυπηρέτη [43].




**Σχήμα 2.6** Ακολουθία ανταλλαγής μηνυμάτων ανάμεσα στον Εξυπηρέτη και τον Πελάτη σύμφωνα με το πρωτόκολλο HTTP 1.0 για τη μεταφορά μιας ιστοσελίδας. Παρατηρούμε ότι δημιουργούνται ξεχωριστές συνδέσεις και γίνονται ξεχωριστές αιτήσεις για κάθε αντικείμενο της ιστοσελίδας.

Αν υποθέσουμε δηλαδή ότι χρειαζόμαστε ένα τέταρτο του δευτερολέπτου για να συνδεθούμε με τον κόμβο www.caching.com τότε για να φέρουμε ολόκληρη την ιστοσελίδα που φαίνεται στο σχήμα 2.5 χρειαζόμαστε τουλάχιστον 8 δευτερόλεπτα για τις συνδέσεις με τον Εξυπηρέτη WWW, χωρίς να υπολογίσουμε το χρόνο μεταφοράς των αντικειμένων. Αυτή βέβαια που περιγράφουμε είναι η χείριστη περίπτωση και οι περισσότεροι Web browsers λειτουργούν πιο έξυπνα ανοίγοντας πολλές συνδέσεις παράλληλα με τον Εξυπηρέτη WWW. Αυτή η τεχνική από την άλλη πλευρά μπορεί να λειτουργήσει εις βάρος του χρήστη, γιατί ο Εξυπηρέτης WWW δέχεται πολλές αιτήσεις ταυτόχρονα και υπερφορτώνεται χωρίς να μπορεί να τις εξυπηρετήσει όλες παράλληλα. Συνεπώς το αποτέλεσμα είναι κατά προσέγγιση όμοιο με την περίπτωση των σειριακών αιτήσεων.

Όλα αυτά που περιγράψαμε παραπάνω συμβαίνουν στη έκδοση 1.0 του πρωτοκόλλου HTTP [99]. Η έκδοση 1.1 [88] έρχεται να βελτιώσει την κατάσταση χρησιμοποιώντας μακροχρόνιες συνδέσεις (persistent connections) για πολλά αντικείμενα και ακολουθία συνεχόμενων αιτήσεων (pipelined requests) για να μειώσει την καθυστέρηση των πολλών ενσωματωμένων αντικειμένων. Στο σχήμα 2.7 παρακάτω φαίνεται μία ακολουθία γεγονότων σύμφωνα με το πρωτόκολλο HTTP 1.1 που αντιστοιχεί στην ακολουθία του σχήματος 2.6. Όπως παρατηρούμε στο σχήμα αυτό μετά την αναζήτηση DNS ανοίγεται μόνο μία TCP/IP σύνδεση με τον μακρινό Εξυπηρέτη WWW και αφού ανακτηθούν τα αρχικά HTML δεδομένα, μπορούν να σταλούν πολλές μαζί αιτήσεις HTTP για τα ενσωματωμένα αντικείμενα της ιστοσελίδας και να ληφθούν οι αντίστοιχες αποκρίσεις πάνω από την ίδια σύνδεση.

Οι μόνιμες συνδέσεις που περιγράφονται στο πρωτόκολλο HTTP έκδοση 1.1, έχουν τρία σημαντικά πλεονεκτήματα [88]:

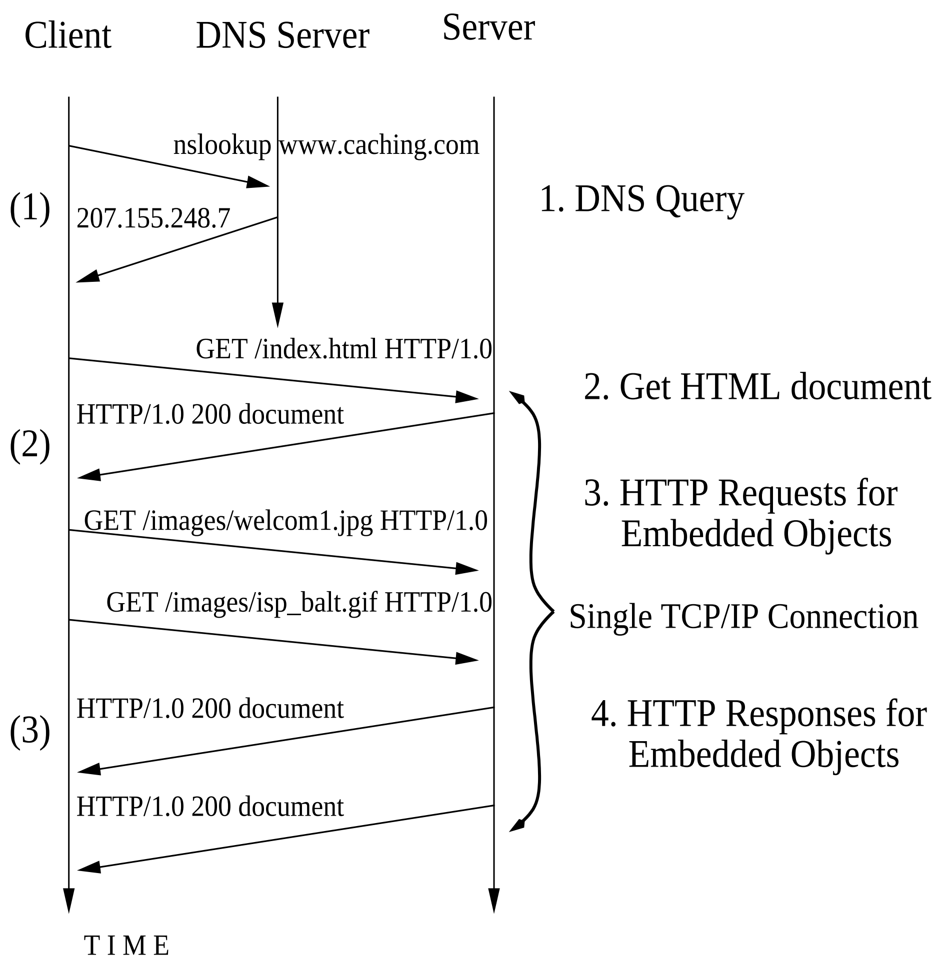
◇ Με τη δυνατότητα της δημιουργίας μακροχρόνιων συνδέσεων (persistent connections), δημιουργούμε λιγότερες συνδέσεις TCP και κάνουμε μεγάλη οικονομία στους υπολογιστικούς πόρους όλων τον κόμβων που έχουν κάποιο δικτυακό ρόλο στη διαδρομή των συνδέσεων (πελάτης, εξυπηρέτης, proxies, δρομολογητές, μεταγωγείς, πύλες κτλ.).

◇ Με τη δυνατότητα της ακολουθίας συνεχόμενων αιτήσεων (pipelined requests), χρησιμοποιούμε πιο αποδοτικά τις μακροχρόνιες συνδέσεις και μειώνουμε τις καθυστερήσεις για τις αποκρίσεις του Εξυπηρέτη WWW.

◇ Η συνολική κίνηση του δικτύου βελτιώνεται γιατί μειώνεται ο αριθμός των δικτυακών πακέτων που χρησιμοποιούνται για το άνοιγμα συνδέσεων TCP.

◇ Βελτιώνεται η συμπεριφορά του πρωτοκόλλου TCP που χρειάζεται κάποιο χρόνο για να προσαρμοστεί στο φόρτο του δικτύου και να έχει καλύτερη ταχύτητα μεταφοράς δεδομένων.

Σήμερα γίνεται προσπάθεια για τη χρήση της έκδοσης 1.1 του πρωτοκόλλου HTTP από όλους τους Web browsers και Εξυπηρέτες WWW στο Διαδίκτυο, κάτι που αποδεικνύεται δύσκολο λόγω του τεράστιου αριθμού των ήδη εγκατεστημένων συστημάτων που χρησιμοποιούν την έκδοση 1.0. Υπάρχουν όμως ακόμα θέματα όπου το πρωτόκολλο HTTP μπορεί να βελτιωθεί, μέσω π.χ. συμπίεσης δεδομένων, όπως δείχνουν διάφορες ερευνητικές προσπάθειες που γίνονται [65], [33].




**Σχήμα 2.7** Ακολουθία ανταλλαγής μηνυμάτων ανάμεσα στον Εξυπηρέτη και τον Πελάτη σύμφωνα με το πρωτόκολλο HTTP 1.1 για τη μεταφορά μιας ιστοσελίδας. Παρατηρούμε ότι ανοίγεται μόνο μία TCP/IP σύνδεση και γίνονται συνεχείς αιτήσεις για όλα τα αντικείμενα της ιστοσελίδας μετά το πρώτο (κείμενο HTML) που περιγράφει ποια είναι τα υπόλοιπα αντικείμενα.

## 2.4 Η ΠΡΟΚΛΗΣΗ ΤΩΝ ΣΥΣΤΗΜΑΤΩΝ WWW CACHING

Όπως είδαμε στην παράγραφο 1.3 το ιδανικό σύστημα ενδιάμεσης μνήμης απέχει αρκετά από το σημείο που βρισκόμαστε σήμερα. Ο τομέας που υπάρχουν τα περισσότερα προβλήματα είναι οι υψηλές επιδόσεις που πρέπει να έχει ένα Web proxy. Ο σχεδιασμός και η κατασκευή συστημάτων ενδιάμεσης μνήμης με υψηλές επιδόσεις παρουσιάζει αρκετές προκλήσεις κυρίως για τους παρακάτω λόγους [43], [44]:

- ◇ Ο αριθμός των αντικειμένων στον Παγκόσμιο Ιστό είναι τεράστιος (υπολογίζεται στην τάξη των δισεκατομμυρίων [46], [25], [87], [87]). Ένα Web proxy υψηλών επιδόσεων χρειάζεται τεράστιο χώρο (της τάξης των δεκάδων GigaBytes) για να αποθηκεύσει εκατομμύρια αντικείμενα στη μνήμη του, παρέχοντας παράλληλα πολύ μικρή καθυστέρηση στην ανάκτηση οποιουδήποτε από τα αντικείμενα αυτά.

◇ Ένα Web proxy πρέπει να είναι σωστά σχεδιασμένο ώστε να μπορεί να δέχεται αιτήσεις από εκατοντάδες χρήστες ταυτόχρονα, τις οποίες πρέπει να εξυπηρετεί με συνέπεια, με το μέγιστο δυνατό ρυθμό και την χαμηλότερη δυνατή καθυστέρηση. Η λειτουργία του δεν πρέπει να διακόπτεται από υπερβολικό φόρτο, γιατί αυτό μπορεί να σημαίνει διακοπή παροχής της υπηρεσίας του Παγκόσμιου Ιστού σε ένα μεγάλο αριθμό χρηστών.

◇ Το Web proxy αποτελεί ένα κεντρικό σημείο του Διαδικτύου με την έννοια ότι λαμβάνει εκατοντάδες αιτήσεις ανά δευτερόλεπτο που αφορούν εκατομμύρια Εξυπηρέτες WWW [25] καθένας από τους οποίους βρίσκεται σε διαφορετικά σημεία του Διαδικτύου. Έτσι πρέπει να κάνει εκατοντάδες αναζητήσεις DNS ανά δευτερόλεπτο, να δημιουργεί και να δέχεται εκατοντάδες συνδέσεις TCP ανά δευτερόλεπτο κτλ. Αρα πρέπει να σχεδιαστεί έτσι ώστε η απόδοσή του να είναι η βέλτιστη.

◇ Επειδή ένα Web proxy συνδέεται με εκατοντάδες Εξυπηρέτες καθένας από τους οποίους βρίσκεται σε διαφορετικά σημεία του Διαδικτύου και άρα παρουσιάζει πολύ διαφορετικούς χρόνους αποκρίσεων [25], πρέπει όταν αποφασίζει για την αποθήκευση αντικειμένων στη μνήμη του να λαμβάνει υπόψη του το χρόνο προσπέλασής τους [75], [38], [105].

◇ Τα αντικείμενα του Παγκόσμιου Ιστού μεταβάλλονται κατά τη διάρκεια του χρόνου και ένα σωστά σχεδιασμένο Web proxy πρέπει να ενημερώνεται συχνά για τις μεταβολές των αντικειμένων που αποθηκεύει στη μνήμη του [43], [44].

Τα εμπορικά Web proxies που είναι διαθέσιμα σήμερα στην αγορά αντιμετωπίζουν τις προκλήσεις αυτές χρησιμοποιώντας ειδικό υλικό (hardware), ειδικού σκοπού λειτουργικά συστήματα (π.χ. CacheOS) και ειδικά βελτιστοποιημένο λογισμικό Web proxy για να ανεβάσουν τις επιδόσεις τους. Όπως είναι λογικό οι λύσεις αυτές έχουν υψηλό κόστος το οποίο όμως είναι πολλές φορές υπερβολικά υψηλό για τις επιδόσεις που παρουσιάζουν [47], [67], [45], [68], [13].

Πιστεύουμε, όπως και άλλοι ερευνητές [58], [64], ότι αξίζει να ασχοληθούμε με Web proxies ανοιχτού κώδικα (open source) που διατίθενται ελεύθερα προς χρήση και μπορούν να τρέξουν σε κοινό υλικό και ευρέως διαδεδομένα λειτουργικά συστήματα (π.χ. FreeBSD, Linux, Sun Solaris, Windows NT, κτλ.) σαν εφαρμογές χρήστη χρησιμοποιώντας τα γενικού σκοπού συστήματα αρχείων (File Systems) των λειτουργικών αυτών συστημάτων. Πιστεύουμε ότι βελτιώνοντας τη λειτουργία τέτοιων συστημάτων και δίνοντάς τα ελεύθερα προς χρήση μειώνουμε το κόστος λειτουργίας ενός Web proxy ο οποίος με την σειρά του θα βελτιώσει την ποιότητα υπηρεσίας του Παγκόσμιου Ιστού.

Ένα άριστο παράδειγμα τέτοιου συστήματος που περιγράψαμε παραπάνω, το οποίο είναι αρκετά δημοφιλές και του οποίου η απόδοση έχει μελετηθεί αρκετά είναι το Squid [102]. Για τους λόγους αυτούς, στη συγκεκριμένη μεταπτυχιακή εργασία θα ασχοληθούμε σε έκταση με τις μελέτες που έχουν γίνει στο Squid και θα συγκρίνουμε τη δική μας δουλειά με αυτή που έχει γίνει στο Squid. Αντίστοιχα θα χρησιμοποιήσουμε την απόδοση του Squid σαν μέτρο σύγκρισης της απόδοσης των δικών μας μεθόδων.

Υπάρχουν αρκετές μελέτες για την απόδοση του Squid [19], [73], αλλά και σε άλλα Web proxies [43], [55]. Όλες αυτές αναφέρουν ότι τα Web proxies δεν παρουσιάζουν χαμηλή απόδοση εξαιτίας της υπολογιστικής ισχύος ούτε λόγω της κεντρικής μνήμης. Δίνοντας λογικές ποσότητες υπολογιστικής ισχύος και κεντρικής μνήμης παρατηρούμε ότι το όριο στην αύξηση των επιδόσεων (bottleneck) παρουσιάζεται στην αποθήκευση των αντικειμένων στη δευτερεύουσα μνήμη του Web proxy, όπου χρησιμοποιείται σύστημα μαγνητικών δίσκων [60], [79], [92], [90]. Σε αυτό το συμπέρασμα έχουν καταλήξει όλες οι ομάδες που έχουν μελετήσει την απόδοση των Web proxies [62], [6], [58], [78] Λόγω της χαμηλής απόδοσης του Squid όταν αποθηκεύει αντικείμενα σε μαγνητικούς δίσκους, σε μερικές εγκαταστάσεις του ρυθμίζεται έτσι ώστε να μην αποθηκεύει καθόλου αντικείμενα στους δίσκους του [64].

Η παρούσα μεταπτυχιακή εργασία ασχολείται με το ειδικό αυτό πρόβλημα της βελτίωσης της χρήσης δευτερεύουσας μνήμης, ένα θέμα που αποτελεί βασική επιδίωξη για τη βελτίωση της απόδοσης των Web proxies. Στο επόμενο κεφάλαιο λοιπόν θα αρχίσουμε την περιήγηση μας στο χώρο των προβλήματων που δημιουργούν οι δίσκοι παρουσιάζοντας τη δουλειά που έχει γίνει από άλλες ερευνητικές ομάδες, αλλά και τη δική μας ερευνητική δουλειά.

# Το Πρόβλημα της Δευτερεύουσας Μνήμης ενός Web proxy

Beware of bugs in the above code;
I have only proved it correct, not tried it.
—Donald Knuth

## 3.1 ΟΡΙΣΜΟΣ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ ΤΗΣ ΔΕΥΤΕΡΕΥΟΥΣΑΣ ΜΝΗΜΗΣ

Πρόσφατες μελέτες της απόδοσης των Web proxies [60], [90], [78], [41] αναφέρουν ότι η απόδοση του συστήματος δίσκων, που χρησιμοποιείται ως δευτερεύουσα μνήμη για την αποθήκευση αντικειμένων του Παγκόσμιου Ιστού, αποτελεί σημαντικό παράγοντα μείωσης της απόδοσης (bottleneck) τους [62], [6], [58], [79]

Οι Maltzahn, Richardson και Grunwald υπολόγισαν ότι χρειάζονται μέχρι και τέσσερις πράξεις *εισόδου - εξόδου δεδομένων στο δίσκο* (θα αναφέρεται στο εξής ως *"I/O δεδομένων σε δίσκους"*) για την εγγραφή ή την ανάγνωση ενός μόνο αντικειμένου WWW [60]. Κατά συνέπεια είναι λογική η παρατήρηση των Rousskov και Soloviev [77], [78], [79] που αναφέρουν ότι η καθυστέρηση του συστήματος δίσκων είναι υπεύθυνη για το 30% του συνολικού χρόνου μίας επιτυχημένης αίτησης στο Web proxy.

Ο Jeff Mogul παρατηρεί [64] ότι η καθυστέρηση από το "I/O δεδομένων σε δίσκους" στο Squid [102], είναι στην πράξη πολύ μεγαλύτερη από τη βελτίωση στον χρόνο απόκρισης λόγω της επιτυχημένης αίτησης. Λόγω της υψηλής αυτής καθυστέρησης, όπως ο ίδιος αναφέρει, σε πολλές περιπτώσεις το Squid χρησιμοποιείται ρυθμισμένο έτσι ώστε να μην αποθηκεύει καθόλου αντικείμενα WWW στη δευτερεύουσα μνήμη του. Ο μόνος ρόλος που έχει σε τέτοιες εγκαταστάσεις είναι να προσφέρει πρόσβαση στον Παγκόσμιο Ιστό στους χρήστες τοπικού δικτύου που προστατεύεται από firewall.

Η απλή, αλλά ακριβή λύση στο πρόβλημα την οποία θα μπορούσε κάποιος να προτείνει είναι η αποθήκευση αντικειμένων μόνο στην πρωτεύουσα κύρια μνήμη και όχι στο δίσκο. Όμως, όπως έχουν δείξει διάφορες ερευνητικές εργασίες, το ποσοστό επιτυχημένων αιτήσεων για αντικείμενα του Παγκόσμιου Ιστού σε ένα Web proxy αυξάνει με λογαριθμικό ρυθμό τόσο ανάλογα με τον όγκο της κίνησης και τον αριθμό των Πελατών του Web proxy [34], [15], [11], όσο και με το μέγεθος της μνήμης του [34], [15], [11], [104]. Στην πράξη αυτό σημαίνει ότι για να είναι ένα Web proxy αποτελεσματικό πρέπει να χρησιμοποιεί συνολική κεντρική μνήμη (RAM) της τάξης των δεκάδων ως εκατοντάδων GBytes. Όπως είναι κατανοητό η χρήση τόσης κεντρικής μνήμης σε ένα Web proxy, λόγω του πολύ υψηλού κόστους, είναι στις περισσότερες περιπτώσεις αδύνατη.

Μέχρι λοιπόν η τιμή της κύριας μνήμης να γίνει αρκετά χαμηλή, τα Web proxies θα χρησιμοποιούν δίσκους για την αποθήκευση αντικειμένων WWW. Εμπορικά Web proxies λύνουν το πρόβλημα του I/O στους δίσκους χρησιμοποιώντας ειδικά συστήματα αρχείων ή ακόμα και ειδικά λειτουργικά συστήματα. Για παράδειγμα το NetCache της Network Appliance χρησιμοποιεί το WAFL, ένα log-based σύστημα αρχείων, ο Traffic Server της Inktomi χρησιμοποιεί άμεσα τις συσκευές του δίσκου παρακάμπτοντας το σύστημα αρχείων, ενώ η CacheFlow έχει υλοποιήσει το ειδικό λειτουργικό σύστημα CacheOS το οποίο δεν χρησιμοποιεί καθόλου σύστημα αρχείων.

Παρ' όλες αυτές τις ειδικού σκοπού λύσεις, υπάρχει μεγάλο ερευνητικό ενδιαφέρον για λύσεις στο πρόβλημα της απόδοσης που προκαλεί στα Web proxies η αποθήκευση αντικειμένων σε δίσκους, και ειδικά σε γενικού σκοπού λειτουργικά συστήματα και κλασσικά συστήματα αρχείων. Όπως έχουν δείξει οι μέχρι τώρα προσπάθειες [4], [90], [58], [62] τέτοιες λύσεις είναι εφικτές και όπως είναι κατανοητό είναι περισσότερο επιθυμητές από τις ειδικού σκοπού λύσεις εξαιτίας κυρίως του χαμηλότερου κόστους εγκατάστασης και λειτουργίας, της μεγαλύτερης διαθεσιμότητας, αλλά και τη μικρότερη σε πολυπλοκότητα υλοποίηση.

### 3.1.1 ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΑΠΟΘΗΚΕΥΣΗΣ ΑΝΤΙΚΕΙΜΕΝΩΝ WWW ΣΤΟΥΣ ΔΙΣΚΟΥΣ ΕΝΟΣ WEB PROXY

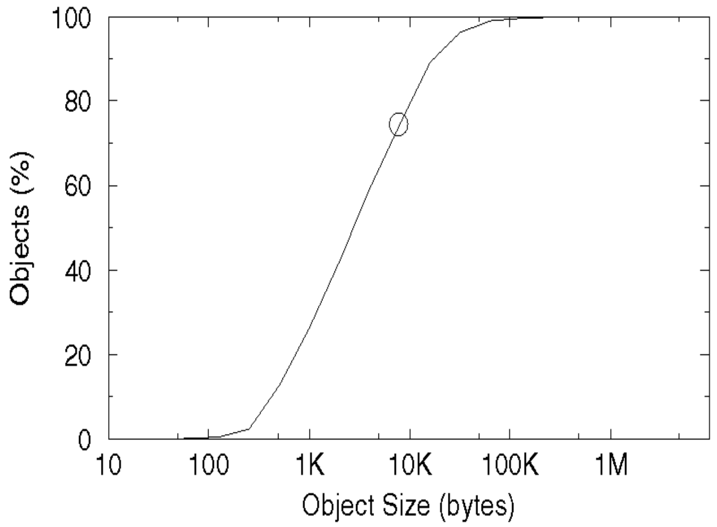
Η αποθήκευση αντικειμένων του Παγκόσμιου Ιστού στο δίσκο των Web proxies έχει αρκετά ιδιαίτερα χαρακτηριστικά. Τα σημαντικότερα από αυτά περιγράφουμε αμέσως παρακάτω:

◇ **Ολόκληρα Αντικείμενα Μόνο**

Τα αντικείμενα του Παγκόσμιου Ιστού γράφονται ή διαβάζονται στο δίσκο πάντα ολόκληρα και δεν υπάρχουν σταδιακές προσθήκες ή διαγραφές μέρους του αντικειμένου. Όταν ένα αντικείμενο αλλάζει, αυτό προκαλεί την καθ' ολοκληρίαν αντικατάστασή του και όχι μόνο μέρους του. Αυτό δεν αποτελεί σημαντική διαφορά σε σχέση με τον καθημερινό φόρτο των συστημάτων αρχείων, όπου το 65% των προσπελάσεων στο δίσκο είναι για την εγγραφή ή την ανάγνωση ολόκληρων αρχείων [58]. Το χαρακτηριστικό ότι τα αντικείμενα στο δίσκο του Web proxy διαβάζονται ή γράφονται ολόκληρα και ότι δεν υπάρχουν σταδιακές προσθήκες ή διαγραφές από αυτά, μπορεί να προκαλέσει fragmentation στο δίσκο γιατί δεν υπάρχουν μικρά κομμάτια για να γεμίσουν τα μικρά κενά που δημιουργούνται με τις εγγραφές και διαγραφές αρχείων.

◇ **Μέγεθος Αντικειμένων WWW**

Το μέσο μέγεθος των αντικειμένων του Παγκόσμιου Ιστού που αποθηκεύονται στη μνήμη του Web proxy είναι 3-5 Kbytes, ενώ το 74% των αντικειμένων είναι μικρότερο από 8 Kbytes [58], [60], [90].




**Σχήμα 3.1** Η κατανομή του μεγέθους των αρχείων στον WWW. Το σημείο που είναι ο κύκλος (3-5 KB) είναι το μέσο μέγεθος αρχείου [58].

Τα 8 Kbytes είναι ένα πολύ συνηθισμένο μέγεθος για τις σελίδες λειτουργικών συστημάτων, αλλά και τα περισσότερα συστήματα αρχείων των λειτουργικών συστημάτων Unix έχουν μέγεθος block ίσο με 8 Kbytes ενώ το μέγεθος του fragment τους είναι ίσο με 1 Kbyte. Δεδομένου ότι τα περισσότερα συστήματα αρχείων είναι βελτιστοποιημένα για να αποθηκεύουν γρήγορα πολλαπλάσια των blocks (8 Kbytes), ενώ για τα fragments χρησιμοποιούν πιο πολύπλοκες μεθόδους για εύρεση χώρου στο δίσκο (π.χ. αλγορίθμους best-fit), είναι κατανοητό ότι τα αντικείμενα WWW (που στη συντριπτική τους πλειοψηφία είναι μικρότερα από το block του συστήματος αρχείων) προκαλούν μεγαλύτερο φόρτο από την περίπτωση όπου θα ήταν μεγαλύτερα σε μέγεθος.

◇ **Κατανομή Δημοφιλέστερων Αντικειμένων**

Η κατανομή των δημοφιλέστερων αντικειμένων στον Παγκόσμιο Ιστό είναι παρόμοια της κατανομής Zipf[1], $\Omega/i^{\alpha}$ όπου $\Omega = (\sum_{i=1}^{N} 1/i^{\alpha})$ και i είναι το i-οστό πιο δημοφιλές αντικείμενο του Παγκόσμιου Ιστού. Οι τιμές του $\alpha$ κυμαίνονται από 0.64 μέχρι 0.83, ενώ ομογενείς κοινότητες χρηστών (οι οποίοι έχουν πιο κοινά ενδιαφέροντα) παρουσιάζουν μεγαλύτερες τιμές του $\alpha$ από ότι πιο ετερογενείς κοινότητες [11].

Πάντως τα δημοφιλέστερα αντικείμενα σε ένα Web proxy αποτελούν ένα πολύ μικρό μέρος του συνόλου των αντικειμένων, ενώ συνήθως η πλειοψηφία των αντικειμένων προσπελαύνονται μόνο μία φορά. Αντίθετα, οι αναφορές που γίνονται στη συνηθισμένη δραστηριότητα του συστήματος αρχείων έχουν πολύ μεγαλύτερη χρονική και χωρική τοπικότητα από ότι τα αντικείμενα WWW [58]. Αυτό σημαίνει ότι χρησιμοποιώντας μία κρυφή μνήμη στο σύστημα αρχείων μπορούμε να βελτιώσουμε πολύ τις επιδόσεις του [70].

◇ **Τοπικότητα Δεδομένων**

Ο μεγαλύτερος αριθμός ιστοσελίδων του Παγκόσμιου Ιστού αποτελούνται από ένα αριθμό ενσωματωμένων αντικειμένων, όπως είδαμε στην παράγραφο 2.3 τα οποία πάντα αναχτούνται μαζί, όταν ζητηθεί μια ιστοσελίδα από ένα Εξυπηρέτη WWW ή ένα Web proxy. Συνήθως μάλιστα οι διευθύνσεις των αντικειμένων της ίδιας ιστοσελίδας αναφέρονται στην διεύθυνση του ίδιου Εξυπηρέτη WWW. Αυτό το χαρακτηριστικό είναι παρόμοιο με την παρατήρηση που έχει γίνει στα συστήματα αρχείων ότι τα αρχεία που βρίσκονται στον ίδιο κατάλογο (directory) τείνουν να προσπελαύνονται μαζί σε μικρά χρονικά διαστήματα. Αρα αν αποθηκεύουμε τα αντικείμενα μιας ιστοσελίδας σε αρχεία στο δίσκο, αυτά θα παρουσιάζουν χωρική και χρονική τοπικότητα. Αν θέλουμε να μειώσουμε το χρόνο εγγραφής και ανάγνωσης των αρχείων αυτών από το δίσκο, είναι κατανοητό ότι θα πρέπει να εκμεταλλευτούμε τη χωρική και χρονική τους τοπικότητα.

◇ **Τοπικότητα Μεταδεδομένων**

Μεταδεδομένα του δίσκου για ένα αντικείμενο είναι τα δεδομένα που περιγράφουν τη θέση των δεδομένων του αντικειμένου στο δίσκο. Το γεγονός ότι τα αρχεία των αντικειμένων που βρίσκονται στην ίδια ιστοσελίδα τείνουν να προσπελαύνονται μαζί σε μικρά χρονικά διαστήματα σημαίνει ότι και τα μεταδεδομένα

[1] Η κατανομή Zipf είναι $\Omega/i$.

τους θα αναφέρονται μαζί στα ίδια χρονικά διαστήματα. Συνεπώς και τα μεταδεδομένα των αρχείων των αντικειμένων της ίδιας ιστοσελίδας θα παρουσιάζουν χωρική και χρονική τοπικότητα, την οποία είμαστε σε θέση να εκμεταλλευτούμε για να βελτιώσουμε το χρόνο εγγραφής και ανάγνωσης αντικειμένων από/προς το δίσκο.

◇ **Αναλογία Εγγραφών / Αναγνώσεων**

Το ποσοστό επιτυχημένων αιτήσεων σε ένα Web proxy είναι της τάξης του 30-50% [3], [34], [85]. Κάθε επιτυχημένη αίτηση περιλαμβάνει μία ανάγνωση των μεταδεδομένων και στη συνέχεια των δεδομένων ενός αντικειμένου. Αντίθετα, κάθε αποτυχημένη αίτηση της οποίας το αντικείμενο πρέπει να αποθηκευτεί στο δίσκο περιλαμβάνει μία εγγραφή των μεταδεδομένων και στη συνέχεια των δεδομένων ενός αντικειμένου στο σύστημα αρχείων.

Εφόσον οι αποτυχημένες αιτήσεις είναι συνήθως περισσότερες (50-70%) από τις επιτυχημένες, είναι κατανοητό ότι θα έχουμε περισσότερες εγγραφές από ότι αναγνώσεις στο δίσκο. Σύμφωνα με μετρήσεις [79] η δημιουργία και το γράψιμο νέων αρχείων στο δίσκο αντιστοιχεί για το 45-65% των πράξεων I/O του δίσκου. Το γεγονός αυτό επιβεβαιώθηκε και με μετρήσεις που περιλαμβάνονται στο [62].

Αυτή η αναλογία αναγνώσεων - εγγραφών αποτελεί μια εντελώς αντίθετη κατάσταση από ότι συμβαίνει συνήθως στα συστήματα αρχείων, όπου έχουμε συνήθως περισσότερες αναγνώσεις από ότι εγγραφές [70]. Συνεπώς τα περισσότερα συστήματα αρχείων είναι βελτιστοποιημένα για αναγνώσεις, όπου χρησιμοποιώντας μια μετρίου μεγέθους κρυφή μνήμη μπορούν να μειώσουν μέχρι και 90% την κίνηση δεδομένων προς το δίσκο [70]. Σε αντίθεση, τα Web proxies κάνοντας συνήθως εγγραφές στο δίσκο γεμίζουν γρήγορα την κρυφή μνήμη και δεν μπορούν να αποφύγουν την πρόσβαση στο δίσκο. Παράλληλα, με τις συνεχείς εγγραφές και διαγραφές αντικειμένων στο δίσκο δημιουργείται fragmentation και γίνεται δύσκολη η εύρεση χώρου για καινούρια αντικείμενα.

◇ **Σημαντικότητα Δεδομένων**

Τα αντικείμενα του Παγκόσμιου Ιστού που αποθηκεύει στο δίσκο του ένα Web proxy αποτελούν σίγουρα χρήσιμα αντίγραφα των πρωτότυπων αντικειμένων με σκοπό την παροχή καλύτερης ποιότητας υπηρεσίας στους χρήστες. Κατά συνέπεια η αποθήκευσή τους δεν είναι κρίσιμη για τους χρήστες, γιατί οσαδήποτε από αυτά κι αν χαθούν από το δίσκο του Web proxy, μπορούν πάντα να ανακτηθούν από τους αρχικούς Εξυπηρέτες WWW. Μπορεί να κοστίσει λίγο σε Διαδικτυακούς πόρους, αλλά είναι πάντα εφικτό.

Αντίθετα, τα δεδομένα που αποθηκεύουμε συνήθως στα συστήματα αρχείων είναι πολύ μεγαλύτερης σημασίας και εφόσον σβηστούν από το δίσκο είναι μη ανακτήσιμα (εκτός και αν έχουμε αντίγραφά τους σε άλλο φυσικό μέσο, όπως μαγνητική ταινία, CD-ROM, κτλ.). Αυτή είναι άλλη μία θεμελιώδης διαφορά του ρόλου που έχει ένα σύστημα αρχείων που αποθηκεύει τα αντικείμενα ενός Web proxy.

## 3.2 ΠΡΟΣΕΓΓΙΣΕΙΣ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ ΤΗΣ ΔΕΥΤΕΡΕΥΟΥΣΑΣ ΜΝΗΜΗΣ

Οι προσεγγίσεις που γίνονται από ερευνητές για την αντιμετώπιση του προβλήματος του I/O των δίσκων είναι κυρίως τρεις, οι οποίες ευτυχώς μπορεί να είναι αλληλοσυμπληρώμενες. Με άλλα λόγια, μπορούμε να συνδυάσουμε προσεκτικά επιλεγμένες προτεινόμενες λύσεις από κάθε κατεύθυνση και να βρούμε ένα συνδυασμό λύσεων που είναι καλύτερος από κάθε λύση χωριστά. Στις επόμενες τρεις παραγράφους, δίνουμε μια συνοπτική εικόνα των προσεγγίσεων προς τις οποίες έχουν κινηθεί οι πρόσφατες ερευνητικές προσπάθειες.

Όπως θα παρατηρήσει ο αναγνώστης οι τρεις αυτές κατευθύνσεις ψάχνουν για λύσεις σε τρεις ανεξάρτητους χώρους. Αυτό σημαίνει ότι λύσεις και από τους τρεις χώρους μπορούν να συνδυαστούν για να έχουν αθροιστική βελτίωση των επιδόσεων. Για παράδειγμα, μπορούμε να χρησιμοποιήσουμε μια καλή πολιτική αντικατάστασης αντικειμένων που μειώνει τον όγκο των I/O στο δίσκο σε συνεργασία με μία καλή πολιτική για τη βελτιστοποίηση της αλληλεπίδρασης με το σύστημα αρχείων, ενώ παράλληλα θα χρησιμοποιούμε μία συστοιχία παράλληλων δίσκων με ένα καλό αλγόριθμο ισοκατανομής του φορτίου ανάμεσα στους δίσκους.

### 3.2.1 ΠΑΡΑΛΛΗΛΗ ΧΡΗΣΗ ΠΟΛΛΑΠΛΩΝ ΔΙΣΚΩΝ

Η έρευνα στην κατεύθυνση αυτή στρέφεται προς τη βέλτιστη χρήση πολλών δίσκων παράλληλα (π.χ. συστοιχίες δίσκων RAID), ώστε να μπορούν να εξυπηρετήσουν το μεγάλο όγκο των πράξεων I/O ενός Web proxy. Τα θέματα που απασχολούν όσους ασχολούνται με την παράλληλη χρήση δίσκων, αφορούν πολιτικές βέλτιστης κατανομής των αντικειμένων στους δίσκους ώστε να μειωθεί ο χρόνος εγγραφής και ανάκτησής τους, ενώ παράλληλα θα πετυχαίνουν ισοκατανομή του φορτίου στους δίσκους.

Οι Soloviev και Yahin μελετούν [89], [90] λύσεις' χρησιμοποιώντας παράλληλη χρήση πολλαπλών δίσκων, που όπως αναφέρουν χρησιμοποιούνται ήδη σε αρκετές εγκαταστάσεις του Squid. Για παράδειγμα στο NLANR χρησιμοποιούνται 5 δίσκοι, στο Surfnet χρησιμοποιούνται 6 δίσκοι, και στα Uninett, Ruu και Mexcom από 3 δίσκοι σε κάθε proxy server. Υποστήριξη για πολλαπλούς δίσκους υπάρχει ήδη στο Squid, όπου το φορτίο ισοκατανέμεται χρησιμοποιώντας τον κυκλικό αλγόριθμο (round-robin) για το μοίρασμα των αρχείων στους δίσκους[2]. Έτσι ισοκατανέμεται μεν ο όγκος των δεδομένων ανάμεσα στους δίσκους, αλλά όχι και ο πραγματικός φόρτος των δίσκων, ενώ παράλληλα δημιουργούνται προβλήματα άνισης κατανομής του φόρτου όταν έχουμε άνισους σε μέγεθος δίσκους.

Η συνεισφορά των Soloviev και Yahin αφορά δύο φάσεις. Σε πρώτη φάση ασχολούνται με την ισοκατανομή του φορτίου των δίσκων προτείνοντας και δοκιμάζοντας αρκετές πολιτικές τοποθέτησης νέων αρχείων στους δίσκους, ενώ σε δεύτερη φάση μελετούν την ιδέα της κατάτμησης (partitioning) για να βελτιώσουν το χρόνο αναζήτησης θέσεων του δίσκου (disk seek time).

Αποκλείοντας αλγορίθμους που δεν μπορούν να υλοποιηθούν σε επίπεδο εφαρμογής χρήστη, οι Soloviev και Yahin οδηγούνται στην επιλογή του αλγόριθμου **Earliest Last Response (ELR)** που όπως έδειξαν τα αποτελέσματα των πειραμάτων έχει το μικρότερο μέσο χρόνο απόκρισης. Σύμφωνα με τον αλγόριθμο αυτό, όταν το Web proxy αποφασίζει σε ποιο δίσκο θα στείλει μια αίτηση για δημιουργία νέου αρχείου, συγκρίνει τις ουρές των αιτήσεων στους δίσκους και στέλνει την αίτηση για νέο αρχείο στο δίσκο με τη μικρότερη σε μέγεθος ουρά. Αν δύο ή περισσότεροι δίσκοι έχουν ίδιο ελάχιστο μήκος ουράς τότε επιλέγεται αυτός που τελείωσε την τελευταία αίτηση που του δόθηκε στο μικρότερο χρόνο.

Το δεύτερο ζήτημα με το οποίο ασχολούνται οι Soloviev και Yahin είναι η ιδέα της κατάτμησης (partitioning) των δίσκων για τη βελτίωση του χρόνου αναζήτησης (seek time). Υποστηρίζουν ότι η κατάτμηση των δίσκων βοηθάει στη μείωση του χρόνου αναζήτησης ενός αρχείου στο δίσκο.

Στην εργασία λοιπόν [90], ασχολούνται με τη σύγκριση διαφόρων αλγορίθμων κατάτμησης εκ των οποίων επιλέγουν τον αλγόριθμο **Any Request (AR)**, εφόσον τα πειράματά τους δείχνουν ότι χρησιμοποιώντας τον με 100 κατατμήσεις στο δίσκο, παρουσιάζει το χαμηλότερο χρόνο απόκρισης από τους υπόλοιπους που μελετούν. Κατά τη λειτουργία του αλγορίθμου αυτού, όταν το Web proxy αποφασίζει σε ποιο δίσκο θα στείλει μια αίτηση για δημιουργία νέου αρχείου, επιλέγει μία αίτηση από αυτές που υπάρχουν στις ουρές των δίσκων αλλά δεν έχουν εξυπηρετηθεί ακόμα και τοποθετεί το νέο αρχείο στην κατάτμηση που θα εγγραφεί η επιλεγμένη αίτηση.

Στη συνέχεια οι Soloviev και Yahin μελετούν συνδυασμούς αλγορίθμων ισοκατανομής φορτίου και κατάτμησης δίσκων χρησιμοποιώντας τους πλέον επιτυχημένους

[2]Συγκεκριμένα ο αλγόριθμος που χρησιμοποιείται στο Squid είναι ο round-robin αλλά με περισσότερα του ενός αρχεία κάθε φορά. Έτσι π.χ. αν το Squid έχει δύο δίσκους, γράφει τα πρώτα 20 αρχεία στον ένα και τα επόμενα 20 στον άλλον.

αλγόριθμους από κάθε τομέα με την προϋπόθεση βέβαια να μπορούν να υλοποιηθούν σε επίπεδο εφαρμογής χρήστη. Η τελική τους πρόταση, που σύμφωνα με τα πειράματά τους δίνει τα βέλτιστα αποτελέσματα, είναι ο συνδυασμός των αλγορίθμων **ELR + AR**, όπου για self-similar κίνηση (Pareto) όπως είναι συνήθως η κίνηση WWW [72], ο μέσος χρόνος απόκρισης μπορεί να μειωθεί κατά 30-40%.

Η δουλειά των Soloviev και Yahin, αν και καταφέρνει να μειώσει το μέσο χρόνο απόκρισης των δίσκων χρησιμοποιώντας πολλούς δίσκους και μελετώντας αρκετούς αλγορίθμους, μας προβληματίζει για δύο λόγους. Πρώτα από όλα λόγω της μετρικής του χρόνου απόκρισης που χρησιμοποιείται για όλες τις συγκρίσεις των αλγορίθμων. Πιστεύουμε ότι η σωστή μετρική για την απόδοση του υποσυστήματος δίσκων σε ένα Web proxy είναι ο ρυθμός εξυπηρέτησης αιτήσεων, γιατί συνήθως ο χρόνος απόκρισης ανά αίτηση είναι αμελητέος μπροστά στην καθυστέρηση μιας αίτησης του Παγκόσμιου Ιστού. Αν για παράδειγμα ένα Web proxy εξυπηρετεί 100 αιτήσεις ανά δευτερόλεπτο και έχει μέση καθυστέρηση ανά αίτηση της τάξης των 30 msec, ενώ ένα δεύτερο έχει ρυθμό εξυπηρέτησης αιτήσεων 50 αιτήσεις ανά δευτερόλεπτο και μέση καθυστέρηση ανά αίτηση 15 msec, είναι προφανές ότι όλοι οι χρήστες θα προτιμήσουν το πρώτο, γιατί η μεγαλύτερη καθυστέρηση των 15 msec ανά αίτηση θα τους είναι ανεπαίσθητη μέσα στα δευτερόλεπτα που κρατάει συνήθως μια αίτηση WWW. Ο δεύτερος λόγος προβληματισμού είναι τα μοντέλα κίνησης και οι προσομοιωτές που χρησιμοποιήθηκαν. Σίγουρα μια μελέτη με πραγματικά WWW traces και πραγματικούς δίσκους θα είχε περισσότερο κύρος.

### 3.2.2 ΒΕΛΤΙΩΣΗΣ ΤΗΣ ΠΟΛΙΤΙΚΗΣ ΑΝΤΙΚΑΤΑΣΤΑΣΗΣ ΑΝΤΙΚΕΙΜΕΝΩΝ

Σε αυτή την κατεύθυνση έχουν γίνει αρκετές ερευνητικές προσπάθειες. Η λογική των προσεγγίσεων αυτών είναι να χρησιμοποιήσουμε μία πολιτική αντικατάστασης που θα ευνοεί την αποθήκευση αντικειμένων στο δίσκο και παράλληλα θα καταφέρνει να έχει ένα “αρκετά καλό” ποσοστό επιτυχημένων αιτήσεων (Document Hit Rate) και δεδομένων (Byte Hit Rate). Οι παραδοσιακές πολιτικές αντικατάστασης αντικειμένων WWW είναι σχεδιασμένες για αποθήκευση αντικειμένων στην κύρια μνήμη και γι’ αυτό αγνοούν συνήθως τα θέματα απόδοσης που έχει η αποθήκευση αντικειμένων WWW σε δίσκους [78].

#### 3.2.2.1 Η ΠΟΛΙΤΙΚΗ “GREEDYDUAL-SIZE”

Οι Cao και Irani στην εργασία τους [15] προτείνουν ως αλγόριθμο αντικατάστασης αντικειμένων μία παραλλαγή του αρχικού αλγορίθμου GreedyDual [106], [107] που αποτελεί γενίκευση του αλγορίθμου LRU.

Ο παραδοσιακός αλγόριθμος GreedyDual συνδέει μαζί με κάθε αντικείμενο μία τιμή H που αντιστοιχεί στο κόστος που έχει το αντικείμενο να ανακτηθεί από τη μνήμη. Όταν λοιπόν πρόκειται να αντικαταστήσει ένα αντικείμενο, επιλέγει το αντικείμενο με τη χαμηλότερη τιμή του H ($min_H$) και στη συνέχεια μειώνει όλες τις τιμές H των υπολοίπων αντικειμένων κατά $min_H$. Ο αλγόριθμος GreedyDual-Size αποτελεί μία επέκταση του GreedyDual, αλλάζοντας την τιμή του H, από το κόστος που ήταν στον GreedyDual, στο πηλίκο κόστος/μέγεθος. Στην εργασία [15], αποδεικνύεται ότι ο αλγόριθμος GreedyDual-Size είναι βέλτιστος ανάμεσα στους αλγορίθμους πραγματικού χρόνου.

Με τη βοήθεια προσομοιώσεων που οδηγούνται από συμβάντα (trace-driven simulations), οι Cao και Irani αποδεικνύουν επίσης ότι ο αλγόριθμος GreedyDual-Size συνδυάζει την τοπικότητα αντικειμένων μαζί με το κόστος ανάκτησης και το μέγεθός τους και παρέχει υψηλότερο ποσοστό επιτυχημένων αιτήσεων, χαμηλότερη μέση καθυστέρηση, και δημιουργεί τη λιγότερη δικτυακή κίνηση από όλους τους υπάρχοντες αλγορίθμους αντικατάστασης αντικειμένων.

#### 3.2.2.2 ΟΙ ΠΟΛΙΤΙΚΕΣ "FUTURE-LOOKING" ΚΑΙ "FREQUENCY-BASED CYCLIC"

Οι Maltzahn και Richardson ασχολήθηκαν [58] με το θέμα των βελτιστοποιημένων ως προς το I/O των δίσκων πολιτικών αντικατάστασης αντικειμένων WWW. Στην εργασία [58] προτείνουν δύο πολιτικές, την "Future-Looking" και "Frequency-Based Cyclic" (FBC).

Η πολιτική "Future-Looking" χρησιμοποιεί ένα πίνακα που έχει υπολογιστεί γνωρίζοντας από πριν τις μελλοντικές αιτήσεις στα traces που χρησιμοποιήθηκαν για τις μετρήσεις απόδοσης των πολιτικών. Έτσι μπορούμε να πούμε ότι είναι πολύ κοντά στο βέλτιστο αλγόριθμο αντικατάστασης αντικειμένων αφού ξέρει από πριν ποια ακριβώς αντικείμενα θα ζητηθούν. Όπως όμως είναι κατανοητό η πολιτική αυτή δεν έχει νόημα να υλοποιηθεί σε πραγματικά συστήματα και άρα μπορεί να χρησιμεύσει μόνο για να δούμε ποιο είναι το άνω όριο στη βελτιστοποίηση που μπορούμε να κάνουμε σε μια πολιτική αντικατάστασης.

Από την άλλη η πολιτική "Frequency-Based Cyclic" (FBC) είναι υλοποιήσιμη σε πραγματικά Web proxies και λειτουργεί χρησιμοποιώντας μετρητές συχνότητας για όλα τα αντικείμενα που υπάρχουν στο δίσκο και ένα δείκτη που δείχνει το αντικείμενο που είναι υποψήφιο για αντικατάσταση. Όταν χρειαστεί να αντικατασταθεί ένα αντικείμενο, συγκρίνεται ο μετρητής συχνότητας του αντικειμένου που δείχνει ο δείκτης με ένα αριθμό $C_{max}$. Αν είναι μικρότερος από τον αριθμό $C_{max}$ το αντικείμενο αντικαθίσταται, αλλιώς ο δείκτης μετακινείται στο επόμενο αντικείμενο και συγκρίνεται κ.ο.κ.

Σύμφωνα με τα αποτελέσματα της εργασίας [58] και οι δύο πολιτικές καταφέρνουν να μειώσουν τον όγκο των πράξεων I/O στο δίσκο με την "Future-Looking" να είναι φυσικά αρκετά καλύτερη, αφού γνωρίζει τις μελλοντικές αιτήσεις. Παράλληλα και οι δύο πολιτικές έχουν υψηλότερο ποσοστό επιτυχημένων αιτήσεων από την κλασσική πολιτική LRU.

#### 3.2.2.3 Η ΠΟΛΙΤΙΚΗ LDF

Οι Aggarwal και Yu, στην εργασία τους [4] προτείνουν μία δική τους πολιτική αντικατάστασης που αποτελεί μια γενίκευση της πολιτικής LRU (Least Recently Used), ενώ παράλληλα λαμβάνει υπόψη της την αποθήκευση των αντικειμένων στο δίσκο. Η πολιτική αυτή χρησιμοποιεί την έννοια της Ελάχιστης Δυναμικής Συχνότητας (Least Dynamic Frequency ή LDF). Η LDF αποτελεί μια αντικειμενική συνάρτηση που ορίζεται για ένα σύνολο αντικειμένων που βρίσκονται στη μνήμη του Web proxy και βασίζεται στις χρονικές στιγμές εισόδου τους στη μνήμη αυτή. Στην πράξη πρόκειται για ένα μέτρο για το πόσο επιθυμητή είναι η ύπαρξή των αντικειμένων αυτών στη μνήμη του proxy.

Έστω ότι $t_i$ είναι η χρονική στιγμή εισόδου (timestamp) του αντικειμένου i στη μνήμη του Web proxy και t είναι η τιμή ενός καθολικού μετρητή του χρόνου. Τότε η δυναμική του συχνότητα (Dynamic Frequency ή DF) του αντικειμένου i ορίζεται ως $1/(t - t_i)$ και συμβολίζεται ως $DF(i, t)$. Η δυναμική συχνότητα (DF) ενός συνόλου αντικειμένων $S = 1, ..., k$ που βρίσκονται σε συνεχόμενα τμήματα της μνήμης του Web proxy ορίζεται ως το άθροισμα των δυναμικών συχνοτήτων των αντικειμένων του συνόλου S. Δηλαδή αν $t_1, t_2, ...t_k$ είναι οι χρονικές στιγμές εισόδου των αντικειμένων του συνόλου S στη μνήμη του Web proxy, τότε η δυναμική συχνότητα του συνόλου S τη στιγμή t θα είναι $\sum_{i \in S} DF(i, t) = \sum_{i=1}^{k} 1/(t - t_i)$. Πρέπει να παρατηρήσουμε ότι τα διαστήματα ανάμεσα στα αντικείμενα του συνόλου S που είναι άδεια δεν επηρεάζουν τη δυναμική συχνότητα του συνόλου S.

Χρησιμοποιώντας τη συνάρτηση της δυναμικής συχνότητας, η πολιτική αντικατάστασης αντικειμένων της Ελάχιστης Δυναμικής Συχνότητας (Least Dynamic Frequency ή LDF) έχει ως εξής:

*Αντικατέστησε το σύνολο των συνεχόμενων αντικειμένων με την ελάχιστη δυναμική συχνότητα, έτσι ώστε να δημιουργηθεί χώρος για το νέο αντικείμενο στη μνήμη του Web proxy.*

Το μειονέκτημα της μεθόδου αυτής είναι ότι χρειάζεται να διατρέξουμε τις χρονικές στιγμές (timestamps) όλων των αντικειμένων στην μνήμη του proxy για να αποφασίσουμε ποια αντικείμενα θα αντικαταστήσουμε. Όπως καταλαβαίνουμε αυτό είναι πολύ χρονοβόρο. Για το λόγο αυτό δημιουργήθηκε η πολιτική Τοπικά Ελάχιστης Δυναμικής Συχνότητας (Localized Least Dynamic Frequency ή LLDF) που προσεγγίζει την LDF ψάχνοντας μόνο τοπικά σε ένα μικρό μέρος των αντικειμένων της μνήμης που χρησιμοποιήθηκαν πιο πρόσφατα.

Τέλος, οι Aggarwal και Yu ορίζουν και μία πολιτική ελέγχου εισόδου (admission control) αντικειμένων για να αποφασίζουν αν αξίζει ένα νέο αντικείμενο να εισαχθεί στην μνήμη. Χρησιμοποιώντας προσομοιώσεις που καθοδηγούνται από συμβάντα (event-driven simulations) δείχνουν ότι η LLDF ακόμη και χωρίς την πολιτική ελέγχου εισόδου αντικειμένων έχει υψηλότερο ποσοστό επιτυχημένων αιτήσεων από την FIFO, ενώ παράλληλα γράφει σειριακά αντικείμενα στο δίσκο πετυχαίνοντας καλή απόδοση. Η χρήση της πολιτικής ελέγχου εισόδου αντικειμένων σε συνδυασμό με την LLDF δίνει ακόμα καλύτερα αποτελέσματα.

#### 3.2.2.4 Η ΠΟΛΙΤΙΚΗ STAGER

Οι Rousskov και Soloviev στην εργασία τους [78] προτείνουν επίσης τη νέα πολιτική αντικατάσταση αντικειμένων "static caching", ενώ αργότερα στην επόμενη εργασία τους [77] βελτιώνουν τον αλγόριθμο αυτό και παρουσιάζουν το νέο αλγόριθμο "Stager".

Ο αλγόριθμος "Stager" αποτελείται από δύο μέρη, εκ των οποίων το ένα είναι μελετημένο να δουλεύει σε νεκρό χρόνο (δηλαδή μία φορά την ημέρα όταν το Web proxy έχει πολύ λίγες αιτήσεις, όπως γίνεται π.χ. αργά το βράδυ) και το άλλο σε πραγματικό χρόνο. Επίσης ο αλγόριθμος αυτός προϋποθέτει την ύπαρξη δύο ξεχωριστών χώρων για αποθήκευση αντικειμένων, έναν στην κύρια μνήμη που έχει μικρότερο μέγεθος αλλά μεγαλύτερη ταχύτητα και έναν στο δίσκο με μεγαλύτερο μέγεθος και μικρή ταχύτητα.

Σε πραγματικό χρόνο η πολιτική Stager εξυπηρετεί τις επιτυχημένες αιτήσεις από το δίσκο ή τη μνήμη (ανάλογα όπου βρίσκεται το αντικείμενο της αίτησης), ενώ τις αποτυχημένες τις διαχωρίζει σε δύο κατηγορίες: στα "γνωστά" αντικείμενα που είναι αυτά που υπάρχουν ήδη στο δίσκο (ή έχουν αναγνωριστεί όπως θα δούμε παρακάτω από το δεύτερο τμήμα του αλγορίθμου) και στα "άγνωστα" που συναντώνται πρώτη φορά. Έτσι τα μεν "γνωστά" αντικείμενα τα αποθηκεύει στο δίσκο, ενώ τα "άγνωστα" στο χώρο της κύριας μνήμης. Τέλος, το τμήμα αυτό καταγράφει επίσης όλες τις αιτήσεις που έγιναν στο Web proxy είτε είναι επιτυχημένες είτε αποτυχημένες σε ένα αρχείο στο δίσκο.

Το μεγαλύτερο μέρος του αλγορίθμου "Stager" γίνεται από το δεύτερο τμήμα του σε νεκρό χρόνο και περιλαμβάνει το διάβασμα του αρχείου με τα στοιχεία για τις αιτήσεις που εξυπηρετήθηκαν από το Web proxy την προηγούμενη μέρα και τον διαχωρισμό των αντικειμένων σε ομάδες που έχουν παρόμοια χαρακτηριστικά. Τα χαρακτηριστικά αυτά μπορεί να είναι το μέγεθος του αντικειμένου, ο αριθμός των αιτήσεων για το αντικείμενο, ο χρόνος της τελευταίας αίτησης που έγινε για το αντικείμενο, κτλ. Όλες οι ομάδες αυτές στοιχειοθετούν ένα υπερκύβο περιέχοντας όλους τους δυνατούς συνδυασμούς χαρακτηριστικών. Στη συνέχεια, αφού ενημερωθούν, οι ομάδες αυτές διατάσσονται χρησιμοποιώντας κάποιο κριτήριο όπως π.χ. το ποσοστό επιτυχημένων αιτήσεων που είχαν (Document Hit Ratio) ή το ποσοστό επιτυχημένων δεδομένων (Byte Hit Ratio). Οι πληροφορίες για τις ομάδες αντικειμένων συλλέγονται σε κάποιο αρχείο του δίσκου και ενημερώνονται σε νεκρό χρόνο μία φορά τη μέρα.

Το μέρος του αλγορίθμου που δουλεύει σε νεκρό χρόνο, αφού διατάξει τις ομάδες των αντικειμένων, ελέγχει, αρχίζοντας από τα αντικείμενα των υψηλότερων ομάδων, αν υπάρχουν στο δίσκο του proxy. Αν δεν συμβαίνει αυτό τα φέρνει από το δίκτυο και τα αποθηκεύει στο δίσκο, αντικαθιστώντας λιγότερο "πολύτιμα" αντικείμενα. Όλα αυτά γίνονται την ώρα που δεν έχει φόρτο (π.χ. αργά το βράδυ). Η ενέργεια αυτή λέγεται *prefetching* γιατί φέρνουμε αντικείμενα στη μνήμη του proxy πριν μας τα ζητήσει κάποιος πελάτης. Η λογική του prefetching είναι ότι αυτά τα αντικείμενα έχουν μεγάλη πιθανότητα να ξαναζητηθούν, και όταν γίνει αυτό θα υπάρχουν ήδη στο δίσκο του proxy.

Η επιδόσεις της πολιτικής Stager μετρήθηκαν χρησιμοποιώντας trace-driven προσομοιώσεις και συγκρίθηκαν με τον παραδοσιακό αλγόριθμο LRU-TH (βλέπε παράγραφο 1.3.2). Τα traces που χρησιμοποιήθηκαν προέρχονταν από μεγάλα Web proxies όπως το DEC, RUU, ADFA και το NLANR. Τα συμπεράσματα από τις προσομοιώσεις δείχνουν ότι ο αλγόριθμος Stager μπορεί να μειώσει την δραστηριότητα του δίσκου κατά 25-40%. Επίσης δείχνουν ότι χρησιμοποιώντας σχετικά μικρά τμήματα κύριας μνήμης (π.χ. 40 MBytes) για την αποθήκευση των "άγνωστων" αντικειμένων μπορούμε να αυξήσουμε το ποσοστό επιτυχημένων αιτήσεων (DHR) και δεδομένων (BHR) κατά 1-2% [77].

### 3.2.3 ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΤΗΣ ΧΡΗΣΗΣ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ ΑΡΧΕΙΩΝ

Η ιδέα πίσω από την προσέγγιση αυτή είναι να λάβουμε υπόψη μας τον τρόπο λειτουργίας του συστήματος αρχείων που χρησιμοποιείται για την αποθήκευση των δεδομένων στο δίσκο και να βελτιώσουμε τον τρόπο που το χρησιμοποιούμε. Μελέτη του έργου που αναλαμβάνει ένα γενικού σκοπού σύστημα αρχείων όταν χρησιμοποιείται από ένα Web proxy, δείχνει ότι διαφέρει σε αρκετά σημεία από το έργο για το οποίο είναι συνήθως σχεδιασμένο και βελτιστοποιημένο ένα σύστημα αρχείων.

Ένα βασικό στοιχείο για παράδειγμα που είδαμε παραπάνω είναι ότι ένα Web proxy χρησιμοποιεί ένα σύστημα αρχείων κυρίως για την εγγραφή δεδομένων, ενώ συνήθως τα συστήματα αρχείων κάνουν λιγότερες εγγραφές και περισσότερες αναγνώσεις δεδομένων από το δίσκο. Μία λύση που θα μπορούσε λοιπόν να προταθεί είναι να κατασκευάσουμε ένα ειδικό σύστημα αρχείων που θα έλυνε το πρόβλημα. Ήδη για παράδειγμα υπάρχουν συστήματα αρχείων όπως το LFS (Log-structured File System) [83] που είναι βελτιστοποιημένο για εγγραφές δεδομένων[3] [82]. Η συγκεκριμένη ιδιότητα όμως της βελτιστοποίησης των εγγραφών είναι μία μόνο από τις επιθυμητές βελτιώσεις, συνεπώς θα χρειαζόταν πιθανότατα να κατασκευάσουμε μία παραλλαγή του LFS. Η αντίστροφη αναλογία εγγραφών - αναγνώσεων πάντως, έχει οδηγήσει ήδη εταιρείες όπως η Network Appliance στη χρήση Log-Structured συστημάτων αρχείων όπως το WAFL.

Από την άλλη πλευρά, μπορούμε[58], [62] μελετώντας την εξάρτηση του Web proxy με το σύστημα αρχείων να χρησιμοποιήσουμε σε επίπεδο χρήστη ένα κλασσικό σύστημα αρχείων με έξυπνο τρόπο χωρίς να αναγκαστούμε να κατασκευάσουμε νέο, ειδικά βελτιστοποιημένο, σύστημα αρχείων.

Οι Maltzahn και Richardson στην εργασία [58] ασχολήθηκαν με τη βελτιστοποίηση της χρήσης του συστήματος αρχείων από το Web proxy. Οι βελτιώσεις τους στηρίζονται κυρίως στο Squid proxy. Υλοποίησαν και προσομοίωσαν αρκετές πολιτικές βελτίωσης οι οποίες είναι:

◇ **SquidL**
Η πρώτη παρατήρησή τους ήταν ότι ο μεγάλος αριθμός καταλόγων για την αποθήκευση αρχείων δεν ευνοεί την τοπικότητα των δεδομένων στο δίσκο. Έτσι η πολιτική *SquidL* δημιουργεί ένα κατάλογο για κάθε Εξυπηρέτη WWW με όνομα το όνομα του κόμβου του Εξυπηρέτη και σκοπό την αποθήκευση των αντικειμένων που προέρχονται από αυτόν τον συγκεκριμένο Εξυπηρέτη WWW.

◇ **SquidM**
Στην πολιτική αυτή χρησιμοποίησαν αρχεία απεικονισμένα στη μνήμη (memory-mapped files) για την αποθήκευση όλων των αρχείων του Web proxy. Την πολιτική αυτή την ονομάζουν *SquidM*.

◇ **SquidML**
Η πολιτική *SquidML* αποτελεί συνδυασμό των πολιτικών SquidM και SquidL. Στο συνδυασμό αυτό χρησιμοποιείται ένα αρχείο απεικονισμένο στη μνήμη για

[3] Ο αναγνώστης μπορεί να βρει περισσότερες πληροφορίες για το LFS στα [83], [84], [69], [82].

την αποθήκευση των αρχείων που είναι μικρότερα από 8 KBytes (δηλαδή η πολιτική SquidM), ενώ για τα υπόλοιπα αρχεία χρησιμοποιείται η πολιτική SquidL.

◇ **SquidMLA**
Η πολιτική *SquidMLA* αποτελεί συνδυασμό της πολιτικής SquidML με ένα αλγόριθμο για την ευθυγράμμιση των μικρών αντικειμένων που γράφονται στο απεικονισμένο στη μνήμη αρχείο στα όρια της σελίδας μνήμης του λειτουργικού συστήματος. Η τελευταία αυτή ιδιότητα είναι επιθυμητή διότι σε αντίθετη περίπτωση το διάβασμα κάποιου αντικειμένου από το απεικονισμένο στη μνήμη αρχείο μπορεί να δημιουργήσει περισσότερα σφάλματα σελίδας.

Οι παραπάνω πολιτικές δοκιμάστηκαν χρησιμοποιώντας μια γεννήτρια αιτήσεων στο δίσκο (disk workload generator), πραγματικά traces από το Web proxy της DEC και το σύστημα αρχείων UFS του λειτουργικού συστήματος Digital Unix 4.0.

Τα αποτελέσματα δείχνουν ότι τη μέγιστη βελτίωση παρουσιάζουν οι πολιτικές SquidML και SquidMLA, ενώ μεταξύ τους δεν υπάρχει μεγάλη διαφορά απόδοσης. Επίσης, χρησιμοποιώντας τη μέθοδο SquidL που αποθηκεύει απλά τα αρχεία από τον ίδιο Εξυπηρέτη WWW σε ίδιο κατάλογο διατηρώντας την τοπικότητα, μπορούμε να έχουμε 50% μείωση στον όγκο των I/O δεδομένων προς το δίσκο. Χρησιμοποιώντας και αρχεία απεικονισμένα στη μνήμη, η βελτίωση αυτή μπορεί να φτάσει μέχρι και 70%.

# Μέθοδοι Βελτίωσης της Απόδοσης Web Proxy

"We all agree that your theory is crazy, but is it crazy enough?"
—Niels Bohr (1885-1962)

Στο κεφάλαιο αυτό παρουσιάζονται μέθοδοι για τη λύση του προβλήματος της δευτερεύουσας μνήμης των Web proxies που αποτελούν το θεωρητικό υπόβαθρο της μεταπτυχιακής αυτής εργασίας. Η ερευνητική αυτή δουλειά αναπτύχθηκε από μία ομάδα ανθρώπων με πρωτεργάτη τον κ. Ευάγγελο Μαρκάτο και στην οποία συμμετείχε και ο συγγραφέας. Η αναλυτική περιγραφή της ερευνητικής αυτής προσπάθειας γίνεται στη δημοσίευση [62] και έχει παρουσιαστεί στο συνέδριο 2nd USENIX Symposium on Internet Technologies and Systems (USITS'99).

Αξίζει να αναφέρουμε ότι οι παρατηρήσεις που οδήγησαν στη σχεδίαση αυτών των μεθόδων είναι σε πολλά σημεία όμοιες με αυτές στη δουλειά των Maltzahn και Richardson που περιγράψαμε στην παράγραφο 3.2.3. Αν και η δημοσίευση της εργασίας τους [58] έγινε λίγο καιρό πριν τη δική μας [62], οι μέθοδοι αναπτύχθηκαν παράλληλα και από τις δύο ομάδες. Αλλωστε, όπως θα δούμε στη συνέχεια αυτού του κεφαλαίου, οι μέθοδοι που προτείνονται από αυτή την εργασία προχωρούν περισσότερο και δίνουν πολύ πιο ολοκληρωμένες λύσεις.

## 4.1 ΜΕΛΕΤΗ ΤΟΥ ΦΟΡΤΟΥ ΕΝΟΣ ΣΥΣΤΗΜΑΤΟΣ ΑΡΧΕΙΩΝ ΑΠΟ ΕΝΑ WEB PROXY

Στην παράγραφο 3.1.1 περιγράφηκαν τα χαρακτηριστικά των αντικειμένων που χειρίζεται ένα Web proxy σε λειτουργία. Έχοντας υπόψη αυτά τα πολύ βασικά χαρακτηριστικά, μελετάται ο χειρισμός των αντικειμένων WWW από τα σημερινά Web proxies, πάντα σε σχέση με το σύστημα αρχείων. Όπως αποδεικνύεται στην παράγραφο αυτή, το σύστημα αρχείων παρεμβαίνει αρνητικά στην απόδοση ενός Web proxy με συγκεκριμένους τρόπους. Στην επόμενη παράγραφο θα δούμε με ποιους τρόπους μπορούμε να μειώσουμε δραστικά αυτή την αρνητική αλληλεπίδραση.

### 4.1.1 ΔΙΑΧΕΙΡΙΣΗ ΑΡΧΕΙΩΝ

Τα περισσότερα και δημοφιλέστερα Web proxies ανοιχτού κώδικα σήμερα αποθηκεύουν κάθε αντικείμενο του Παγκόσμιου Ιστού σε ένα ξεχωριστό αρχείο. Πιστεύουμε ότι η διαχείριση αυτών των αρχείων από το σύστημα αρχείων αποτελεί μεγάλο εμπόδιο στην αύξηση των επιδόσεων ενός Web proxy. Για να επιβεβαιωθεί αυτή η αρνητική επίδραση του συστήματος αρχείων, μετρήθηκε ο χρόνος δημιουργίας και διαγραφής αρχείων. Για τις μετρήσεις χρησιμοποιήθηκε το μετροπρόγραμμα (benchmark) latfs από το σύνολο μετροπρογραμμάτων HBENCH-OS [12]. Στο πείραμα δημιουργήθηκαν 64 αρχεία διαφόρων μεγεθών και στη συνέχεια διαγράφηκαν.

Στα πειράματα παρατηρήθηκε ότι ο χρόνος δημιουργίας αρχείου μηδενικού μεγέθους είναι πάνω από 20 msec ενώ για αρχείο μεγέθους 10 KBytes χρειάζονται τουλάχιστον γύρω στα 40 msec. Αντίστοιχα ο χρόνος διαγραφής αρχείου μη μηδενικού μεγέθους είναι περισσότερος από 20 msec. Από τα παραπάνω νούμερα πρέπει να παρατηρήσουμε ότι εφόσον ο μέσος χρόνος δημιουργίας και διαγραφής αρχείου 4 KBytes είναι γύρω στα 50 msec, αυτό σημαίνει ότι το σύστημα αρχείων δεν μπορεί να κατασκευάσει πάνω από 20 αρχεία ανά δευτερόλεπτο. Αρα, με λίγα λόγια, αν ένα Web proxy αποθηκεύει κάθε URL σε ένα αρχείο (όπως κάνει για παράδειγμα το Squid), δεν θα μπορεί να εξυπηρετήσει πάνω από 20 αιτήσεις πελατών για URLs το δευτερόλεπτο λόγω της καθυστέρησης από τη διαχείριση αρχείων. Αν σκεφτούμε ότι το μέσο μέγεθος αρχείου στον Παγκόσμιο Ιστό είναι 3 KBytes [79], τότε το Web proxy θα μπορεί να εξυπηρετεί αιτήσεις που αντιστοιχούν σε ρυθμό δεδομένων 3 KBytes ανά 50 msec, ή περίπου 60 KBytes ανά δευτερόλεπτο, ποσότητα δύο τάξεις μεγέθους μικρότερη από το ρυθμό δεδομένων που μπορούν να προσφέρουν οι περισσότεροι δίσκοι σήμερα.

Τα αποτελέσματα αυτά συμφωνούν με τα αποτελέσματα δημοσιευμένων πειραμάτων με άλλα μετροπρογράμματα [63], [84], [82], [69].

## 4.2 ΒΕΛΤΙΣΤΟΠΟΙΩΝΤΑΣ ΤΗΝ ΔΙΑΧΕΙΡΙΣΗ ΑΡΧΕΙΩΝ

Τα περισσότερα και δημοφιλέστερα Web proxies ανοιχτού κώδικα σήμερα (όπως το Squid [102] και το CERN [57]) αποθηκεύουν κάθε αντικείμενο του Παγκόσμιου Ιστού σε ένα ξεχωριστό αρχείο. Τα αρχεία αυτά αποθηκεύονται σε μία ρηχή (όπως π.χ. το Squid) ή μια βαθύτερη ιεραρχία καταλόγων. Γνωρίζοντας από τα παραπάνω πειράματα ότι το κόστος της διαχείρισης πολλών αρχείων μειώνει πολύ το ρυθμό εξυπηρέτησης αιτήσεων, πιστεύουμε ότι αυτό το κόστος αποτελεί το μεγαλύτερο παράγοντα μείωσης της απόδοσης ενός Web proxy.

### 4.2.1 Ο ΑΛΓΟΡΙΘΜΟΣ BUDDY

Τα σημερινά Web proxies χρησιμοποιούν συνήθως κάποιον από τους εξής τρεις αλγορίθμους για την ομαδοποίηση των αντικειμένων WWW σε αρχεία [58], [62]:

- ◇ **"SIMPLE":** Στον αλγόριθμο αυτό κάθε αντικείμενο αποθηκεύεται σε διαφορετικό αρχείο και όλα αυτά τα αρχεία βρίσκονται στον ίδιο κατάλογο.

- ◇ **"SQUID":** Ο αλγόριθμος αυτός, που χρησιμοποιείται από το Squid, κατασκευάζει αρχικά μία ιεραρχία καταλόγων που αποτελείται από δύο επίπεδα. Το πρώτο επίπεδο περιέχει 16 καταλόγους (με ονόματα 0 ... F) και το δεύτερο 256 καταλόγους (με ονόματα 00 ... FF) για κάθε κατάλογο του πρώτου επιπέδου. Τα αρχεία γράφονται στους καταλόγους με ένα κυκλικό αλγόριθμο (Round - Robin), δηλαδή το πρώτο αρχείο δημιουργείται στον κατάλογο 0/00 το δεύτερο 0/01, ... μέχρι το 0/FF και το επόμενο στο 1/00 κ.ο.κ.

- ◇ **"MULTIPLE-DIRS":** Στον αλγόριθμο αυτό κάθε αντικείμενο αποθηκεύεται σε διαφορετικό αρχείο, αλλά όλα τα αντικείμενα που προέρχονται από τον ίδιο Εξυπηρέτη WWW γράφονται στον ίδιο κατάλογο και κάθε κατάλογος περιέχει αντικείμενα από ένα μόνο Εξυπηρέτη WWW. Όλοι οι κατάλογοι βρίσκονται στο ίδιο επίπεδο.

Για να μειωθεί το κόστος της διαχείρισης αρχείων προτάθηκε [62] ένας νέος αλγόριθμος για την ομαδοποίηση των αντικειμένων σε αρχεία ο οποίος ονομάστηκε "BUDDY". Η κύρια ιδέα πίσω από τον αλγόριθμο αυτό είναι η μείωση του πλήθους των αρχείων με την αποθήκευση περισσοτέρων του ενός αντικειμένων σε ένα αρχείο. Έτσι τα αντικείμενα που χωράνε μέσα σε ένα block του δίσκου (512 bytes) αποθηκεύονται όλα σε ένα αρχείο, αυτά που χωράνε μέσα σε δύο blocks (αλλά είναι μεγαλύτερα από ένα block) αποθηκεύονται όλα σε ένα άλλο αρχείο κ.ο.κ. Τελικά λοιπόν, κάθε αρχείο περιέχει αντικείμενα που χρειάζονται τον ίδιο αριθμό blocks στο δίσκο, οπότε μπορούμε να πούμε ότι κάθε αρχείο αποτελείται από μια σειρά ισομεγέθων "θέσεων" (slots). Σύμφωνα με τον αλγόριθμο, τοποθετούμε κάθε αντικείμενο στην πρώτη ελεύθερη "θέση" του αντίστοιχου αρχείου.

Ο αλγόριθμος BUDDY λειτουργεί ως εξής:

- ◇ Αρχικά ο αλγόριθμος δημιουργεί ένα αρχείο για τα αντικείμενα που χωράνε μέσα σε ένα block του δίσκου, ένα δεύτερο για αυτά που χωράνε μέσα σε δύο blocks (αλλά όχι σε ένα block), κ.ο.κ. μέχρι ένα "οριακό" αριθμό από blocks (threshhold). Τα αντικείμενα που είναι μεγαλύτερα ή ίσα με αυτόν τον αριθμό blocks αποθηκεύονται το καθένα στο δικό του αρχείο.

◇ Σε μια αίτηση εγγραφής για ένα αντικείμενο δεδομένου μεγέθους, ο αλγόριθμος "BUDDY" βρίσκει την πρώτη άδεια "θέση" στο κατάλληλο αρχείο και αποθηκεύει τα δεδομένα του αρχείου στη θέση αυτή. Αν ένα αντικείμενο είναι μεγαλύτερο ή ίσο με τον "οριακό" αριθμό (π.χ. 256 blocks ή 128 KBytes) τότε ο BUDDY φτιάχνει ένα αρχείο και αποθηκεύει εκεί το αντικείμενο μόνο του.

◇ Σε μια αίτηση διαγραφής αντικειμένου μεγέθους μικρότερου του "οριακού", ο BUDDY σημειώνει ότι η αντίστοιχη "θέση" που ήταν γραμμένο το αρχείο είναι τώρα κενή. Αν το αντικείμενο είναι μεγαλύτερο του "οριακού" μεγέθους διαγράφει το αρχείο που ήταν αποθηκευμένο το αντικείμενο.

◇ Σε μια αίτηση ανάγνωσης αντικειμένου βρίσκει την ανάλογη "θέση" στο κατάλληλο αρχείο και διαβάζει το περιεχόμενο του αντικειμένου.

Το μεγαλύτερο πλεονέκτημα του BUDDY είναι ότι πρακτικά εξαφανίζει το κόστος της διαχείρισης από τη δημιουργία και διαγραφή αρχείων. Τα αντικείμενα που κατέχουν μόνα τους ένα αρχείο είναι ένα πολύ μικρό ποσοστό των συνολικών αντικειμένων (όπως φαίνεται από την κατανομή του μεγέθους των αντικειμένων WWW στην παράγραφο 3.1.1) και άρα το κόστος της διαχείρισης των αρχείων αυτών είναι αμελητέο. Ένα ακόμα πλεονέκτημα του BUDDY είναι ότι τοποθετώντας τα αντικείμενα ίσου μεγέθους στο ίδιο αρχείο μειώνει το fragmentation εντός των αρχείων, γιατί κάθε αντικείμενο καταλαμβάνει συνεχόμενες θέσεις στο ίδιο αρχείο.

Η ίδια λογική που χρησιμοποιήθηκε στον αλγόριθμο BUDDY για τη μείωση της διαχείρισης πολλών μικρών αρχείων από το σύστημα αρχείων, βρίσκεται και πίσω από τη χρήση αρχείου απεικονισμένου στην μνήμη (memory mapped file) στην εργασία [58]. Στην εργασία αυτή, οι Maltzahn και Richardson χρησιμοποίησαν ένα μεγάλο αρχείο απεικονισμένο στην μνήμη για την αποθήκευση των αντικειμένων WWW που ήταν μικρότερα από 8 KBytes, ενώ για τα υπόλοιπα έφτιαχναν ξεχωριστά αρχεία. Αντίθετα, ο αλγόριθμος BUDDY δεν σταματάει στο μέγεθος των 8 KBytes. Όπως έδειξαν τα πειράματα του [62], υπάρχει βελτίωση της απόδοσης του συστήματος αρχείων για αντικείμενα μεγέθους μέχρι και 128 KBytes. Συνεπώς τα αποτελέσματα των πειραμάτων του BUDDY είναι πολύ καλύτερα από αυτά των Maltzahn και Richardson.

## 4.3 ΒΕΛΤΙΣΤΟΠΟΙΩΝΤΑΣ ΤΟ ΡΥΘΜΟ ΕΓΓΡΑΦΗΣ ΣΤΟ ΔΙΣΚΟ

Μετά τη μείωση του κόστους από τη διαχείριση αρχείων παρατηρήθηκε ότι το μεγαλύτερο επόμενο εμπόδιο για την αύξηση των επιδόσεων ήταν η καθυστέρηση του δίσκου από τις εγγραφές σε πολλά σημεία διασκορπισμένα σε όλη την επιφάνειά του. Ο αλγόριθμος BUDDY, αν και μειώνει το κόστος της διαχείρισης αρχείων, δεν κάνει καμμία ενέργεια για την τοποθέτηση των δεδομένων στο δίσκο ώστε να αυξηθεί η απόδοση των εγγραφών (ή/και των αναγνώσεων).

Γνωρίζοντας από την παράγραφο 3.1.1 ότι ο φόρτος του δίσκου ενός Web proxy κυριαρχείται από εγγραφές, είναι λογικό να γίνει μια προσπάθεια για την καλύτερη απόδοση των εγγραφών. Λαμβάνοντας υπόψη μας τη λειτουργία των μαγνητικών δίσκων, η απόδοση των εγγραφών θα είναι βέλτιστη αν γράφουμε στο δίσκο όπως τα Log-Structured συστήματα αρχείων [84], [83], [69], [82], δηλαδή ξεκινώντας από την αρχή του δίσκου να γράφουμε σειριακά προσθέτοντας συνεχώς δεδομένα μέχρι να φτάσουμε στο τέλος. Τότε αν θέλουμε να προσθέσουμε κι άλλα δεδομένα επιστρέφουμε στην αρχή και ξαναγράφουμε σειριακά.

Η ιδέα λοιπόν που προτείνεται [62] για τη βελτιστοποίηση των εγγραφών είναι να χρησιμοποιήσουμε μια τέτοια μέθοδο, όχι βέβαια φτιάχνοντας καινούριο σύστημα αρχείων, αλλά σε επίπεδο εφαρμογής χρήστη κάνοντας το Web proxy να γράφει με τον ίδιο log-structured τρόπο. Ο αλγόριθμος λοιπόν που επινοήθηκε [62] ονομάστηκε "STREAM" και η λειτουργία του είναι ως εξής:

◇ Το Web proxy αποθηκεύει όλα τα αντικείμενά του σε ένα αρχείο που αποτελείται από "θέσεις" ("slots") σταθερού μεγέθους 512 bytes, όσο δηλαδή ένα block του δίσκου. Κάθε αντικείμενο καταλαμβάνει ακέραιο αριθμό θέσεων (π.χ. ένα αντικείμενο με μέγεθος 1300 bytes θα καταλάβει 3 "θέσεις"). Αυτό γίνεται για να αποφύγουμε το διάβασμα περισσότερων blocks στην περίπτωση που ένα αντικείμενο περνάει τα όρια του ενός block. Στο [58] που χρησιμοποιούνται αρχεία απεικονισμένα στη μνήμη, προσπαθούν να ευθυγραμμίσουν τα αντικείμενα στα όρια της σελίδας μνήμης. Έτσι όμως σπαταλάνε περισσότερο χώρο, γιατί η σελίδα μνήμης είναι συνήθως μεγαλύτερη σε μέγεθος (8 KBytes) από ότι το block του δίσκου (512 bytes).

◇ Οι εντολές διαγραφής ενός αντικειμένου σημειώνουν το χώρο που καταλάμβαναν τα περιεχόμενα του αντικειμένου σαν ελεύθερο χώρο έτσι ώστε να εγγραφούν εκεί νέα αντικείμενα.

◇ Οι εντολές ανάγνωσης ενός αντικειμένου διαβάζουν, από τα αντίστοιχα μέρη του αρχείου, τα περιεχόμενα του αντικείμενου και τα επιστρέφουν.

◇ Οι εντολές εγγραφής τέλος ενός αντικειμένου γράφουν από την αρχή του δίσκου προσθέτοντας συνεχώς μέχρι να τελειώσει ο διαθέσιμος χώρος του δίσκου. Σε αυτή την περίπτωση επιστρέφουν στην αρχή του μεγάλου αυτού αρχείου και συνεχίζουν να γράφουν προσθέτοντας δεδομένα μόνο στον ελεύθερο χώρο που έχει δημιουργηθεί.

Ο αλγόριθμος STREAM έχει τη δυνατότητα να κάνει μεγάλες σειριακές εγγραφές στο δίσκο, το μέγεθος των οποίων εξαρτάται από το διαθέσιμο χώρο στο δίσκο, το οποίο με τη σειρά του εξαρτάται από το διαθέσιμο χώρο που έχουμε δώσει στο αρχείο. Αν δεν έχουμε δώσει καθόλου ελεύθερο χώρο, τότε θα υπάρχει συνεχώς μόνο μία ελεύθερη θέση στο δίσκο η οποία θα κινείται μη σειριακά και θα προκαλεί συνεχείς κινήσεις της κεφαλής του δίσκου μειώνοντας δραματικά την απόδοση. Η φιλοσοφία του αλγορίθμου STREAM είναι ότι οι δίσκοι πρέπει να δουλεύουν σε χωρητικότητα πολύ λιγότερη του 100%, ώστε να υπάρχει πάντα αρκετός ελεύθερος χώρος στο δίσκο. Ο χώρος αυτός θα βελτιώνει την απόδοση επιτρέποντας τις μεγάλες συνεχείς εγγραφές.

Όταν μετρήθηκαν αρχικά οι επιδόσεις του αλγορίθμου STREAM [62] παρατηρήθηκε ότι αν και υπήρχε πάντα ελεύθερος χώρος και καθόλου αναγνώσεις στο δίσκο, ο STREAM δεν έγραφε στο δίσκο με το μεγαλύτερο ρυθμό. Το πρόβλημα εντοπίστηκε στις μικρές εγγραφές, δηλαδή στο ότι γράφονταν λίγα δεδομένα (512 bytes) κάθε φορά με αποτέλεσμα να γίνεται μία ανάγνωση και μία εγγραφή στο δίσκο. Ο λόγος για το φαινόμενο αυτό είναι ότι αν γραφτούν στο δίσκο δεδομένα μεγέθους μικρότερου από μία σελίδα μνήμης, το λειτουργικό σύστημα θα διαβάσει τη σελίδα αυτή από το δίσκο, θα ενημερώσει το κομμάτι των δεδομένων που γράφτηκε και θα ξαναγράψει ολόκληρη τη σελίδα πίσω στο δίσκο. Για να μειωθεί αυτό το φαινόμενο των μικρών εγγραφών, επινοήθηκε [62] ένας νέος αλγόριθμος, που ομαδοποιεί τις εγγραφές σε πακέτα και ονομάστηκε "STREAM-PACKETIZER". Ο αλγόριθμος STREAM-PACKETIZER δουλεύει όπως ακριβώς ο STREAM με μόνη εξαίρεση την επόμενη:

- ◇ Υπάρχει ένας buffer που έχει μέγεθος όσο μία σελίδα μνήμης και είναι ευθυγραμμισμένος στο όριο μιας σελίδας. Οι εγγραφές αντικειμένων δεν προωθούνται κατευθείαν στο σύστημα αρχείων, αλλά συγκεντρώνονται μέσα στον buffer σε συνεχόμενες θέσεις το ένα από το άλλο. Όταν ο buffer γεμίσει, ή η θέση που πρέπει να γραφτεί ένα αντικείμενο δεν είναι συνεχόμενη, ο buffer στέλνεται στο σύστημα αρχείων για να γραφτεί στο δίσκο.

Η απόδοση των αλγορίθμων BUDDY, STREAM και STREAM-PACKETIZER μετρήθηκε [62] σαν συνάρτηση του χρησιμοποιημένου χώρου στο δίσκο. Τα πειράματα έδειξαν ότι όταν η χρήση του δίσκου είναι μεγάλη (περίπου 95%) οι STREAM και STREAM-PACKETIZER έχουν παρόμοια απόδοση με τον BUDDY. Αυτό συμβαίνει γιατί δεν υπάρχουν μεγάλα συνεχή κομμάτια στο δίσκο ώστε να έχουμε αποδοτικές εγγραφές. Αντίθετα, όταν η χρήση του δίσκου είναι μικρότερη του 72% ο STREAM αποδίδει δύο φορές καλύτερα από τον BUDDY και ο STREAM-PACKETIZER περίπου 2.5 φορές καλύτερα. Στην πράξη, το Web proxy χρησιμοποιώντας τον αλγόριθμο STREAM-PACKETIZER μπορεί να ανεβάσει το ρυθμό εξυπηρέτησης αιτήσεων σε περισσότερες από 350 ανά δευτερόλεπτο. Βέβαια, για να πετύχουν αυτές τις αποδόσεις, οι STREAM και STREAM-PACKETIZER χρειάζονται περίπου 30% επιπλέον χώρο στο δίσκο από ότι ο χώρος των αντικειμένων που αποθηκεύουν. Από την άλλη όμως οι αλγόριθμοι αυτοί, χρησιμοποιώντας ένα μόνο αρχείο, καταφέρνουν να μειώσουν το χώρο των μεταδεδομένων του συστήματος αρχείων κερδίζοντας χρόνο σε σχέση με τους παραδοσιακούς αλγορίθμους SIMPLE, SQUID και MULTIPLE-DIRS. Σχετικά με τον επιπλέον χώρο στο δίσκο, ευτυχώς οι τιμές των δίσκων μειώνονται στο μισό κάθε χρόνο [27], ενώ όπως πρόσφατες μετρήσεις αναφέρουν [30], τα περισσότερα συστήματα αρχείων δουλεύουν με διαθέσιμο το μισό χώρο στο δίσκο.

## 4.4 ΒΕΛΤΙΣΤΟΠΟΙΩΝΤΑΣ ΤΟ ΡΥΘΜΟ ΑΝΑΓΝΩΣΗΣ ΑΠΟ ΤΟ ΔΙΣΚΟ

Αν και οι αλγόριθμοι STREAM και STREAM-PACKETIZER μειώνουν στο ελάχιστο τις μετακινήσεις της κεφαλής του δίσκου για εγγραφές αντικειμένων, το Web proxy μπορεί ακόμη να υποφέρει από τις ίδιες αιτίες λόγω ανάγνωσης δεδομένων από το δίσκο. Έστω δηλαδή ότι η κεφαλή του δίσκου γράφει σειριακά κάποια αντικείμενα στη θέση X και λόγω μιας επιτυχημένης αίτησης στο Web proxy πρέπει να διαβαστεί ένα αντικείμενο από το δίσκο. Κατά συνέπεια, η κεφαλή του δίσκου πρέπει να μετακινηθεί στο σημείο Y που βρίσκονται τα περιεχόμενα του αρχείου για να τα διαβάσει και στη συνέχεια να επιστρέψει και πάλι στο X για να συνεχίσει τις εγγραφές. Αν και οι αναγνώσεις αντικειμένων είναι λιγότερες από τις εγγραφές, κάθε μια από αυτές προκαλεί συνήθως δύο μετακινήσεις της κεφαλής προκαλώντας έτσι μεγάλη καθυστέρηση.

### 4.4.1 Ο ΑΛΓΟΡΙΘΜΟΣ LAZY-READS

Για την αντιμετώπιση αυτού του προβλήματος επινοήθηκε [62] ένας νέος αλγόριθμος, ο “LAZY-READS”. Η λειτουργία του αλγορίθμου αυτού μοιάζει με αυτή του STREAM-PACKETIZER με την εξής διαφορά:

◇ Όταν λαμβάνεται μια αίτηση ανάγνωσης, στέλνεται σε ένα ενδιάμεσο buffer. Όταν ο buffer γεμίσει με αιτήσεις ανάγνωσης τις προωθεί στο σύστημα αρχείων διατεταγμένες ανάλογα με τη θέση (στο ένα αρχείο που περιέχονται όλα τα αντικείμενα) των δεδομένων κάθε αίτησης.

Στα πειράματά που έγιναν, με τον αλγόριθμο LAZY-READS να στέλνει ανά δέκα τις αιτήσεις ανάγνωσης στο σύστημα αρχείων, παρατηρήθηκε ότι βελτιώνει την απόδοση του STREAM-PACKETIZER κατά 10%.

### 4.4.2 Ο ΑΛΓΟΡΙΘΜΟΣ LAZY-READS-LOC

Όπως παρατηρήσαμε στην παράγραφο 2.3 και 3.1.1 τα αντικείμενα του Παγκόσμιου Ιστού παρουσιάζουν αρκετή τοπικότητα [15]. Για παράδειγμα τα αντικείμενα μιας ιστοσελίδας ανακτούνται όλα μαζί. Είναι λοιπόν σκόπιμο να αποθηκεύουμε τα αντικείμενα αυτά σε κοντινές τοποθεσίες στο δίσκο ώστε μελλοντικές αναγνώσεις τους (που θα παρουσιάζουν χρονική τοπικότητα) να γίνονται με τη μέγιστη ταχύτητα. Αυτή είναι και η φιλοσοφία πίσω από τον αλγόριθμο MULTIPLE-DIRS που είδαμε παραπάνω. Δυστυχώς τα περισσότερα από τα σημερινά Web proxies καταστρέφουν την τοπικότητα των αντικειμένων πολυπλέκοντας τα αντικείμενα από πολλούς διαφορετικούς σε ανάμεικτα αρχεία και καταλόγους. Έτσι τα περιεχόμενα των αντικειμένων που βρίσκονται στην ίδια ιστοσελίδα και προσπελαύνονται μαζί μπορούν να βρίσκονται εκατοντάδες KBytes μακριά το ένα από το άλλο στο δίσκο.

Για να βελτιωθεί αυτή την κατάσταση επινοήθηκε [62] ο αλγόριθμος "LAZY-READS-LOC" που αποτελεί μία βελτίωση του LAZY-READS με την εξής διαφορά:

◇ Υπάρχουν ένα σύνολο από buffers οι οποίοι ονομάζονται "locality buffers" και έχουν στόχο να ομαδοποιούν τις εγγραφές αντικειμένων που προέρχονται από τους ίδιους Εξυπηρέτες WWW. Ο στόχος είναι να βελτιωθούν οι αναγνώσεις των αντικειμένων αυτών.

◇ Όταν ο Web proxy θέλει να αποθηκεύσει στο δίσκο του ένα αντικείμενο ψάχνει στους "locality buffers" του αν υπάρχει κάποιος buffer που συγκεντρώνει αντικείμενα από τον ίδιο Εξυπηρέτη WWW και σε τέτοια περίπτωση αποθηκεύει τα περιεχόμενά του σε αυτόν. Αν δεν υπάρχει τέτοιος buffer, επιλέγεται ένας victim buffer τα περιεχόμενα του οποίου γράφονται σειριακά στο δίσκο ενώ στη συνέχεια αποθηκεύονται σ' αυτόν τα περιεχόμενα του αντικειμένου.

# Σχεδιασμός και Υλοποίηση ενός Εξελιγμένου Web Proxy

C isn't that hard: void (*(*f[])())() defines f as an array of unspecified size, of pointers to functions that return pointers to functions that return void.

—

Είδαμε στο κεφάλαιο 4 αρκετούς νέους αλγορίθμους για τη βελτίωση της απόδοσης των Web proxies τόσο για την εγγραφή δεδομένων όπου ο καλύτερος ήταν ο STREAM-PACKETIZER, όσο και για την ανάγνωση με τους LAZY-READS και τους "locality buffers". Τα αποτελέσματα ωστόσο που είχαμε γι' αυτούς προήλθαν μέσα από προσομοιωτές που, αν και χρησιμοποιούσαν πραγματικές αναγνώσεις και εγγραφές στο δίσκο, δεν είχαν να αντιμετωπίσουν και τα πολλά άλλα προβλήματα που έχουν να αντιμετωπίσουν τα πραγματικά Web proxies. Τα πιο σημαντικά από τα προβλήματα αυτά είναι το άνοιγμα και κλείσιμο πολλών συνδέσεων TCP/IP τόσο από τους πελάτες αλλά και προς τους Εξυπηρέτες WWW, την διαχείριση των μεταδεδομένων των αντικειμένων και το fragmentation που προέρχεται από την πολλή χρήση μνήμης τόσο για την αποθήκευση του περιεχομένου των αντικειμένων, όσο και των μεταδεδομένων τους.

Με σκοπό να μελετήσουμε και αυτές τις παραμέτρους των αλγορίθμων σε ένα πραγματικό Web proxy, υλοποιήσαμε στα πλαίσια της μεταπτυχιακής αυτής εργασίας τους αλγορίθμους που περιγράψαμε στο κεφάλαιο 4. Στο κεφάλαιο αυτό περιγράφουμε τόσο την υλοποίησή μας όσο και τα προβλήματα που αντιμετωπίσαμε κατά τη διάρκεια της προσπάθειας και τις λύσεις που προτείνουμε και υλοποιήσαμε.

## 5.1 ΠΡΟΣΠΑΘΕΙΕΣ ΜΕΤΑΤΡΟΠΗΣ ΥΠΑΡΧΟΝΤΩΝ ΣΥΣΤΗΜΑΤΩΝ

Η αρχική προσπάθειά μας ήταν να δοκιμά-σουμε τον αλγόριθμο STREAM-PACKETIZER σε ένα από τα πιο δημοφιλή και εξελιγμένα Web proxies ανοιχτού κώδικα σήμερα, όπως το Squid ή ο Apache.

### 5.1.1 ΜΕΤΑΤΡΟΠΗ ΤΟΥ SQUID

Οι προσπάθειές μας με το Squid δεν οδήγησαν κάπου, γιατί η υλοποίηση του συγκεκριμένου Web proxy στηρίζεται εξ' ολοκλήρου στην ύπαρξη ενός αρχείου για κάθε αντικείμενο. Το Squid χειρίζεται τα αντικείμενά WWW στη μνήμη του καλώντας δικές του συναρτήσεις που βασίζονται στις κλήσεις συστήματος open(), close(), read() και write(). Επίσης για την αποθήκευση μεταδεδομένων όπως η ώρα αποθήκευσης κάθε αντικειμένου χρησιμοποιείται η χρονική στιγμή (timestamp) τελευταίας προσπέλασης του αρχείου που αποθηκεύει τα περιεχόμενα του αντικειμένου. Τέλος εφόσον το Squid χρησιμοποιεί ένα αρχείο για κάθε αντικείμενο είναι πολύ πιο εύκολο να μην χρησιμοποιεί δικούς του buffers για κάθε αντικείμενο που φέρνει από τον Εξυπηρέτη WWW αλλά όποιο κομμάτι ενός αντικειμένου διαβάζει από το δίκτυο να το γράφει κατευθείαν στο αρχείο που αποθηκεύει αυτό το αντικείμενο.

Εξερευνώντας λοιπόν τη δυνατότητα να ενσωματώσουμε τους αλγορίθμους στο Squid, είδαμε ότι αυτό ήταν πρακτικά αδύνατο χωρίς να χρειαστεί να αλλάξουμε ένα μεγάλο μέρος από τις περίπου 82000 γραμμές κώδικα του. Κρίνοντας λοιπόν ότι αυτό δεν ήταν συμφέρον από πλευράς χρόνου, οι προσπάθειές μας για υλοποίηση των αλγορίθμων στράφηκαν αλλού. Οι υπολογισμοί μας αυτοί αποδείχτηκαν σωστοί όταν και οι ίδιοι οι προγραμματιστές που ανέπτυξαν και αναπτύσσουν τον κώδικα του Squid δεν μπόρεσαν να αλλάξουν τον τρόπο που γράφει στο δίσκο τα αρχεία και στην τελευταία έκδοση του, την 2.3, ανέπτυξαν ένα ειδικό “storage API” ώστε να μπορέσουν να χρησιμοποιήσουν ένα επίσης σύστημα αρχείων ώστε να βελτιώσουν τις επιδόσεις του [103].

### 5.1.2 ΜΕΤΑΤΡΟΠΗ ΤΟΥ APACHE

Η δεύτερη προσπάθειά μας έγινε με τον Εξυπηρέτη Apache (έκδοση 1.3.6) ο οποίος παρέχει και ένα module που λειτουργεί ως Web proxy. Το πρώτο που διαπιστώσαμε δοκιμάζοντας τον Apache είναι ότι υστερεί σε επιδόσεις από το Squid μέχρι και 50%. Επίσης εφόσον είναι κυρίως κατασκευασμένος για χρήση ως Εξυπηρέτης WWW και όχι ως Web proxy δεν υλοποιεί αρκετά features που υλοποιεί το Squid.

Ο Apache χρησιμοποιεί πολλές θυγατρικές διεργασίες για να χειριστεί τις αιτήσεις των πελατών οι οποίες λειτουργούν εντελώς ανεξάρτητα μεταξύ τους χάρη στην ύπαρξη ενός αρχείου για κάθε αντικείμενο. Σε διαφορετική περίπτωση, όπως αυτή που θέλουμε να υλοποιήσουμε εμείς, θα χρειαζόταν επικοινωνία και συγχρονισμός ανάμεσά τους για να αποθηκεύουν τα περιεχόμενα των αντικειμένων σε ένα μεγάλο αρχείο (όπως υπαγορεύει ο αλγόριθμος STREAM-PACKETIZER), χωρίς να υπάρχει πολυπλεξία των περιεχομένων των αντικειμένων. Για να το πετύχουμε αυτό υλοποιήσαμε μία ξεχωριστή διεργασία "storage manager" με την οποία συνδέονταν όλες οι ξεχωριστές διεργασίες του Apache και έστελναν τα αντικείμενα που έπρεπε να αποθηκευτούν στον δίσκο. Στη συνέχεια όμως παρατηρήσαμε ότι η διαγραφή αντικειμένων από το δίσκο του Apache (δηλαδή η υλοποίηση της πολιτικής αντικατάστασης), γινόταν με μια ξεχωριστή διεργασία, η οποία αναλάμβανε να σβήσει από το δίσκο τα αρχεία που δεν χρειάζονταν. Για να μπορέσει να δουλέψει σωστά η υλοποίησή μας θα έπρεπε να υλοποιήσουμε τη δική μας πολιτική αντικατάστασης στον Apache.

Έχοντας φτάσει στο παραπάνω σημείο, αποφασίσαμε να μη χρησιμοποιήσουμε τελικά τον Apache και λόγω του ότι θα έπρεπε να διαθέσουμε ένα μεγάλο μέρος του χρόνου μας για να κάνουμε αυτές τις μεγάλες αλλαγές σε άγνωστο για εμάς κώδικα, αλλά και λόγω των χαμηλών επιδόσεων που είχε. Η τελική μας απόφαση ήταν να αναπτύξουμε ένα δικό μας Web proxy, το οποίο δεν θα υλοποιούσε βέβαια όλα τα χαρακτηριστικά που έχει για παράδειγμα το Squid (πρωτόκολλο ICP [101], client access control, cache-digests, πρωτόκολλο FTP κτλ.), αλλά θα ήταν εξαρχής σχεδιασμένο να έχει υψηλές επιδόσεις και θα ήταν πιο εύκολο και λιγότερο χρονοβόρο να υλοποιήσουμε τους αλγορίθμους μας. Έτσι δημιουργήθηκε το FOXY.

### 5.1.3 Η ΔΗΜΙΟΥΡΓΙΑ ΤΟΥ FOXY

Το FOXY είναι ένα Web proxy σχεδιασμένο με κύριους σκοπούς:

- ◊ τις υψηλές επιδόσεις.
- ◊ την αυτονομία των τμημάτων κώδικα (modules) που υλοποιούν τις διάφορες λειτουργίες και την ξεκάθαρη αλληλεπίδρασή μεταξύ τους ώστε να είναι εύκολη η αντικατάστασή τους με διαφορετικούς αλγορίθμους.
- ◊ την υλοποίηση του αλγορίθμου STREAM-PACKETIZER για την αποθήκευση αντικειμένων στο δίσκο ώστε να μετρηθεί η απόδοσή του σε πραγματικές συνθήκες.

Δυστυχώς δεν υπήρχε ο απαραίτητος χρόνος για να υλοποιηθούν όλες οι δυνατότητες που προσφέρει το Squid στις περίπου 82000 γραμμές κώδικά του, αλλά προσπαθήσαμε και τελικά καταφέραμε μέσα σε περίπου 8000 γραμμές κώδικα να προσφέρουμε τις αναγκαίες λειτουργίες για τη διεξαγωγή πειραμάτων με πολύ γνωστά μετροπρογράμματα (benchmarks) για Web proxy, όπως το Web Polygraph [5] τα οποία συμπεριφέρονται ακριβώς όπως εξελιγμένοι Πελάτες και Εξυπηρέτες WWW. Τέλος πρέπει να αναφέρουμε ότι το FOXY λειτούργησε άψογα και με δημοφιλείς Web browsers (π.χ. Netscape, Internet Explorer) με τους οποίους δοκιμάστηκε.

Πιστεύουμε ότι καταφέραμε να πετύχουμε τους στόχους που είχαμε θέσει, δηλαδή τις υψηλές επιδόσεις, την υλοποίηση του αλγορίθμου STREAM-PACKETIZER και την ξεκάθαρη αλληλεπίδραση των τμημάτων κώδικα (modules). Λόγω του στόχου των υψηλών επιδόσεων, η επιλογή μας ήταν να υλοποιηθεί το FOXY σε ένα λειτουργικό σύστημα από την οικογένεια των συστημάτων Unix. Έτσι το FOXY αναπτύχθηκε στο λειτουργικό σύστημα Solaris 2.6 της Sun Microsystems (δοκιμάστηκε όμως επιτυχώς και στο Redhat Linux έκδοσης 6.1) και αποτελείται από περίπου 8000 γραμμές κώδικα σε γλώσσα προγραμματισμού C.

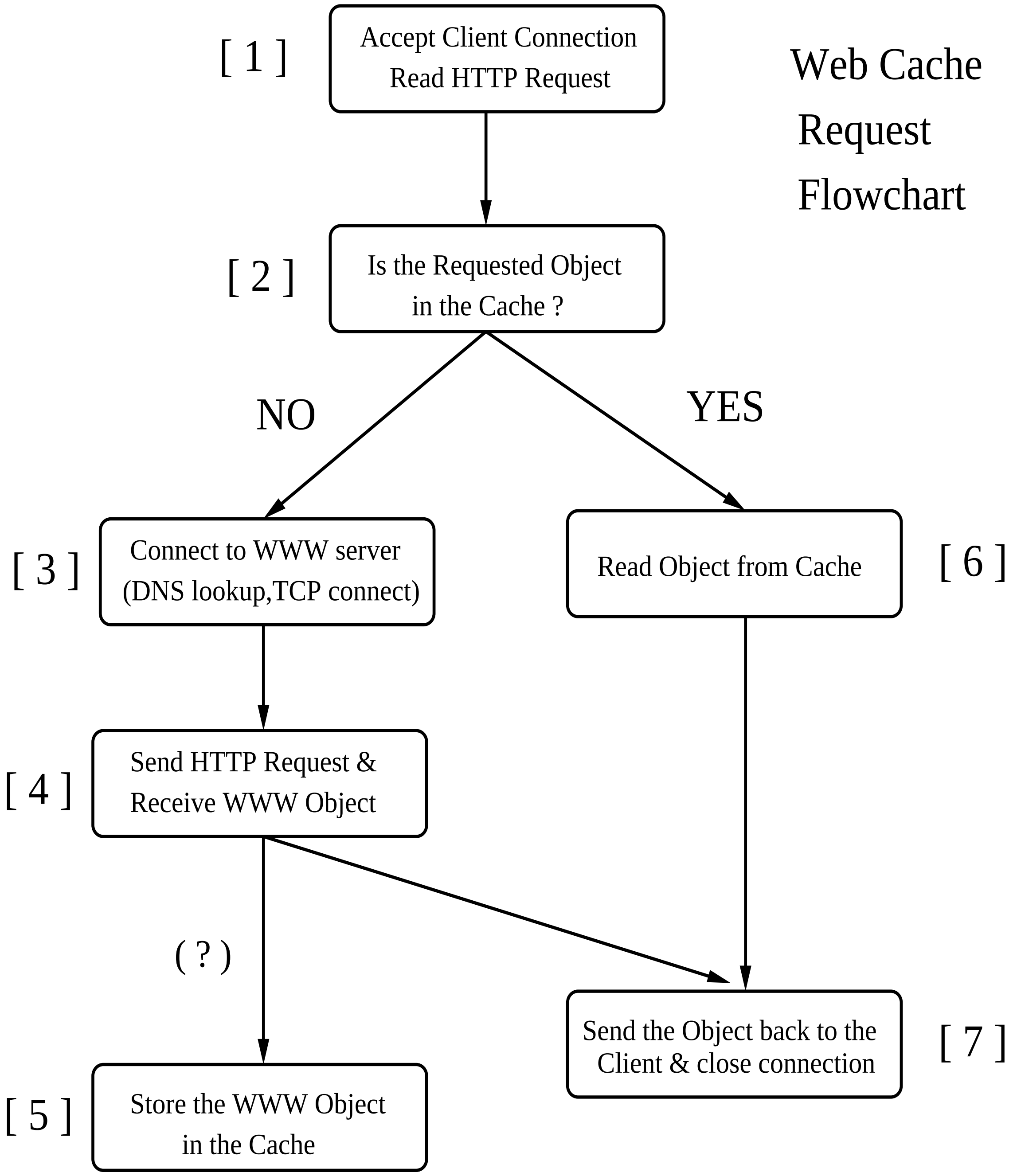
## 5.2 ΤΟ ΣΥΣΤΗΜΑ ΕΝΔΙΑΜΕΣΗΣ ΜΝΗΜΗΣ FOXY

### 5.2.1 ΕΞΥΠΗΡΕΤΗΣΗ ΑΙΤΗΣΗΣ HTTP

Πριν ξεκινήσουμε να περιγράφουμε αναλυτικότερα την υλοποίηση του FOXY πρέπει να δούμε πιο αφαιρετικά την ακολουθία καταστάσεων που περνάει ένα Web proxy για να εξυπηρετήσει μία αίτηση. Η ακολουθία αυτή από καταστάσεις παριστάνεται γραφικά στο σχήμα 5.1 όπου οι διάφορες καταστάσεις έχουν αριθμηθεί. Αναλυτικότερα κάθε μία από τις καταστάσεις αυτές περιλαμβάνει τις ακόλουθες ενέργειες:

1. Στην κατάσταση 1 το Web proxy δέχεται μία σύνδεση TCP από κάποιον πελάτη. Εάν πρόκειται για κάποιο πελάτη που έχει άδεια να χρησιμοποιήσει το Web proxy η σύνδεση γίνεται δεκτή και το Web proxy διαβάζει την αίτηση HTTP του Πελάτη (η μορφή μιας τέτοιας αίτησης φαίνεται στον πίνακα 2.1).

2. Στην κατάσταση 2 το Web proxy αποκωδικοποιεί την πληροφορία που υπάρχει στην αίτηση και ανακαλύπτει καταρχήν τα απαραίτητα στοιχεία της αίτησης, όπως τη μέθοδο HTTP για την ανάκτηση του αντικειμένου (συνήθως η μέθοδος GET), το URL του ζητούμενου αντικειμένου (και άρα το πρωτόκολλο επικοινωνίας, το όνομα και το port του Εξυπηρέτη WWW τον κατάλογο και το όνομα του αρχείου κτλ.) και την έκδοση του πρωτοκόλλου HTTP που τρέχει ο Πελάτης. Επίσης μπορεί να βρει κι άλλες λιγότερο απαραίτητες αλλά χρήσιμες πληροφορίες όπως το πρόγραμμα του Πελάτη, την επιθυμητή γλώσσα για το έγγραφο κτλ.




**Σχήμα 5.1** Διάγραμμα λειτουργίας ενός Web proxy για την εξυπηρέτηση μιας αίτησης.

Στο στάδιο αυτό το Web proxy κάνει μια πολύ βασική λειτουργία, την αναζήτηση του αντικειμένου στη μνήμη[1] του. Η αναζήτηση αυτή γίνεται με τη βοήθεια ενός

[1]Εννοούμε το χώρο αποθήκευσης αντικειμένων του Web proxy και όχι την κύρια (δυναμική) μνήμη του. Στο εξής όταν αναφερόμαστε απλά στη "μνήμη του Web proxy" θα εννοούμε *ολόκληρο* το χώρο αποθήκευσης αντικειμένων του, δηλαδή την κύρια μνήμη και τους δίσκους του.

μηχανισμού ευρετηριασμού (indexing) και αναζήτησης των μεταδεδομένων κάθε αντικειμένου που έχει αποθηκευτεί στη μνήμη του Web proxy. Τα μεταδεδομένα που χρειάζεται να κρατάμε μπορεί να είναι από το όνομα κάθε αντικειμένου, το χρόνο της τελευταίας αλλαγής του, το μέρος της μνήμης που το έχουμε αποθηκεύσει, το μέγεθος των περιεχομένων του κτλ. Ο μηχανισμός λοιπόν αναζήτησης αποφαίνεται για το αν το αντικείμενο βρίσκεται στη μνήμη του Web proxy ή όχι. Στην περίπτωση που το αντικείμενο υπάρχει στη μνήμη (δηλαδή η αίτηση ήταν επιτυχής) η εξυπηρέτηση της αίτησης συνεχίζεται προχωρώντας στην κατάσταση 6, αλλιώς μεταβαίνει στην κατάσταση 3.

3. Στην κατάσταση 3 (όπως και στις 4 και 5 που ακολουθούν) γνωρίζουμε ότι το αντικείμενο της αίτησης δεν βρίσκεται στη μνήμη του Web proxy. Έτσι, αρχικά γίνεται μια αναζήτηση DNS για το όνομα του Εξυπηρέτη WWW που περιέχεται στο URL του αντικειμένου (καλώντας την gethostbyname())και αν είναι επιτυχής ανοίγεται μια TCP/IP σύνδεση με αυτόν τον Εξυπηρέτη WWW (δημιουργείται ένα TCP socket με την κλήση συστήματος open() και η σύνδεση επιτυγχάνεται με κλήση της connect(). Οι παραπάνω ενέργειες μπορεί να διαρκέσουν αρκετά δευτερόλεπτα. Εφόσον ολοκληρωθούν η αίτηση προχωρά στην κατάσταση 4.

4. Στην κατάσταση 4 έχει δημιουργηθεί μια σύνδεση με τον μακρινό Εξυπηρέτη WWW και το Web proxy στέλνει μια αίτηση HTTP για το αντικείμενο. Εφόσον η αίτηση είναι επιτυχής το Web proxy ανακτά το αντικείμενο από τον Εξυπηρέτη WWW όπως θα έκανε ένα πρόγραμμα Πελάτη, και μεταβαίνει παράλληλα στις καταστάσεις 5 και 7.

5. Στην κατάσταση 5 το Web proxy αποφασίζει σύμφωνα με την πολιτική ελέγχου εισόδου (admission control) αν χρειάζεται να αποθηκευτεί το αντικείμενο στη μνήμη του για την εξυπηρέτηση μελλοντικών αιτήσεων. Εφόσον αποφασίσει ότι είναι σκόπιμη η αποθήκευση του αντικειμένου, τα περιεχόμενα του τελευταίου εισάγονται στην ιεραρχία μνήμης του Web proxy, ενώ παράλληλα τα μεταδεδομένα της εισαγωγής του εισάγονται στο σύστημα ευρετηριασμού για μελλοντικές αναζητήσεις.

   Εδώ, με την είσοδο του αντικειμένου στην ιεραρχία μνήμης του Web proxy, γίνονται πολλές ενέργειες που παίζουν κρίσιμο ρόλο στην απόδοση του Web proxy τόσο όσον αφορά το ποσοστό επιτυχημένων αιτήσεων που θα έχει (και άρα πόσο θα ελαττωθεί η κίνηση στο δίκτυο), όσο και το πόσο υψηλός θα είναι ο ρυθμός εξυπηρέτησης αιτήσεων και πόσο χαμηλή θα είναι η μέση καθυστέρηση των αιτήσεων. Τέτοιες λειτουργίες αφορούν την πολιτική αντικατάστασης αντικειμένων, την πολιτική αποθήκευσης αντικειμένων στην κύρια μνήμη και/ή στο δίσκο του Web proxy και τον γρήγορο και (κατά το μέτρο του δυνατού) λιτό σε μέγεθος ευρετηριασμό των μεταδεδομένων.

6. Στην κατάσταση 6, αντίθετα από την κατάσταση 3, γνωρίζουμε ότι η αίτηση για το αντικείμενο ήταν επιτυχής και το τελευταίο υπάρχει στη μνήμη του Web proxy. Έχοντας λοιπόν ανακτήσει τα μεταδεδομένα του γνωρίζουμε σε ποια θέση της ιεραρχίας μνήμης βρίσκονται τα περιεχόμενά του τα οποία στη συνέχεια διαβάζονται από τη μνήμη και προχωράμε στην τελική κατάσταση της αίτησης, την κατάσταση 7.

7. Στην κατάσταση 7 τα περιεχόμενα του αντικειμένου έχουν ανακτηθεί είτε από το δίκτυο (κατάσταση 4), είτε από τη μνήμη του Web proxy (κατάσταση 6) και επιστρέφονται στον Πελάτη. Εφόσον η σύνδεση του Πελάτη έχει γίνει με την έκδοση 1.0 του πρωτοκόλλου HTTP η σύνδεση τερματίζεται, ενώ στην έκδοση 1.1 μπορεί να επαναχρησιμοποιηθεί για νέες αιτήσεις.

Η παραπάνω αποτελεί μια γενική εικόνα των καταστάσεων που μπορεί να βρεθεί μια αίτηση καθώς εξυπηρετείται από ένα Web proxy. Σχεδιάζοντας όμως ένα Web proxy υψηλών επιδόσεων, πρέπει να λάβουμε υπόψη μας και τους παρακάτω παράγοντες:

◇ Σε κάθε στάδιο της αίτησης μπορεί να παρουσιαστεί λάθος το οποίο το Web proxy πρέπει να χειριστεί με τον καλύτερο δυνατό τρόπο. Σε μερικές τέτοιες περιπτώσεις λαθών η μόνη λύση είναι να επιστρέψουμε στον Πελάτη ένα μήνυμα ότι συνέβη κάποιο κρίσιμο λάθος και δεν μπορούμε να ανακτήσουμε το αντικείμενο. Τέτοιες περιπτώσεις είναι για παράδειγμα στην κατάσταση 3 όταν η αναζήτηση DNS του Εξυπηρέτη WWW αποτύχει, ή όταν η σύνδεση TCP με τον Εξυπηρέτη WWW δεν είναι εφικτή για κάποιο λόγο ή στην κατάσταση 4 ο Εξυπηρέτης WWW μας απαντήσει ότι το αντικείμενο της αίτησης δεν υπάρχει.

  Υπάρχουν όμως και περιπτώσεις όπου συμβαίνουν εσωτερικά λάθη στο Web proxy. Αν και ένα σωστά σχεδιασμένο και υλοποιημένο σύστημα δεν πρέπει να έχει λάθη, πολλές φορές αυτά είναι αναπόφευκτα, λόγω π.χ. προβλημάτων στο υλικό (hardware) του υπολογιστικού συστήματος στο οποίο εκτελείται ο Web proxy ή συχνότερα από λάθη στο λογισμικό του Web proxy ή στο λογισμικό του λειτουργικού συστήματος. Έτσι στην κατάσταση 6, μπορεί μεν το σύστημα ευρετηριασμού να μας δείχνει ότι το αντικείμενο υπάρχει στη μνήμη, αλλά να μην μπορεί για κάποιο λόγο να το ανακτήσει. Σε τέτοιες περιπτώσεις έχουμε την εναλλακτική λύση να φέρουμε το αντικείμενο από το δίκτυο, δηλαδή να μεταπηδήσουμε στην κατάσταση 3.

◇ Για να έχει μεγάλο ρυθμό εξυπηρέτησης αιτήσεων, ένα Web proxy πρέπει να εξυπηρετεί πολλές αιτήσεις ταυτόχρονα. Έτσι όταν για παράδειγμα περιμένει την αναζήτηση DNS από μία αίτηση που βρίσκεται στην κατάσταση 3, μπορεί να δέχεται την επόμενη αίτηση η οποία θα βρίσκεται σε μια διαφορετική κατάσταση και να την εξυπηρετεί. Μια σημαντική παρατήρηση που πρέπει να κάνουμε είναι ότι τέτοιες περιπτώσεις όπου το Web proxy περιμένει κάποια στάδια μιας αίτησης να εκτελεστούν υπάρχουν σε αρκετές καταστάσεις του σχήματος 5.1.

Στην κατάσταση 3 χρειάζεται να περιμένουμε αρκετά δευτερόλεπτα, τόσο για την αναζήτηση DNS όσο και για τη δημιουργία της σύνδεσης TCP με τον μακρινό Εξυπηρέτη WWW. Έπειτα στην κατάσταση 4 αφού στείλουμε την αίτηση HTTP πρέπει να περιμένουμε να μεταφερθούν τα περιεχόμενα του αντικειμένου από το Εξυπηρέτη WWW. Τέλος στις καταστάσεις 5 και 6 δεν χρειάζεται πολλές φορές να περιμένουμε την ανάγνωση ή την εγγραφή των δεδομένων από/στη μνήμη, αλλά μπορούμε όση ώρα γίνεται αυτό να εξυπηρετούμε κάποια άλλη αίτηση.

Η ταυτόχρονη εξυπηρέτηση πολλών αιτήσεων μπορεί να υλοποιηθεί με τους παρακάτω τρόπους:

▷ Δημιουργώντας μια ξεχωριστή διεργασία για κάθε αίτηση. Η λύση αυτή είναι ασύμφορη λόγω του μεγάλου κόστους της κλήσης fork() και επιπλέον γιατί η επικοινωνία που χρειάζεται να γίνεται ανάμεσα στις διεργασίες είναι δυσκολότερη λόγω του ξεχωριστού χώρου διευθύνσεων της καθεμιάς.

▷ Δημιουργώντας ένα ξεχωριστό thread για κάθε αίτηση το οποίο περνάει από όλα τα στάδια της αίτησης μέχρι τον τερματισμό της. Η λύση αυτή έχει το πλεονέκτημα ότι είναι πιο εύκολη στην υλοποίησή της και προσφέρει καλή κατανομή του υπολογιστικού φόρτου σε πολυεπεξεργαστικά συστήματα. Από την άλλη όμως υπάρχει και η ανάγκη συγχρονισμού των threads μεταξύ τους όταν προσπελαύνουν κρίσιμα τμήματα, όπως το τμήμα ευρετηριασμού μεταδεδομένων ή το τμήμα που γράφει τα αντικείμενα σειριακά στο δίσκο.

▷ Χρησιμοποιώντας non-blocking sockets (ασύγχρονες κλήσεις σε TCP/IP sockets) μπορούμε να χωρίσουμε κάθε αίτηση σε τμήματα που μπορούν να εξυπηρετηθούν ασύγχρονα (δηλαδή χωρίς να χρειάζεται το Web proxy να περιμένει την ολοκλήρωσή τους). Με τη βοήθεια λοιπόν της κλήσης συστήματος select() να επιλέγουμε κάθε φορά μια αίτηση που έχει ολοκληρώσει το ασύγχρονο τμήμα της και να την προχωρούμε στις επόμενες ενέργειές της.

◇ Ένα Web proxy μπορεί πολλές φορές να δέχεται περισσότερες αιτήσεις από όσες μπορεί να εξυπηρετήσει, και η λειτουργία του να παρουσιάζει προβλήματα. Για το σκοπό αυτό, πρέπει να είναι έτσι σχεδιασμένο ώστε να ελέγχει τον αριθμό αιτήσεων που δέχεται, ανάλογα με τις δυνατότητές του.

Έχοντας στο μυαλό μας το παραπάνω σχέδιο λειτουργίας κάθε αίτησης στο Web proxy και τα υπόλοιπα θέματα που πρέπει να φροντίσουμε, προχωράμε στην περιγραφή της αρχιτεκτονικής του FOXY.

### 5.2.2 ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΛΕΙΤΟΥΡΓΙΑΣ ΤΟΥ FOXY

Το FOXY είναι σχεδιασμένο να εξυπηρετεί ένα μέγιστο αριθμό ταυτόχρονων συνδέσεων. Ο αριθμός αυτός ορίζεται από το πρόσωπο που το εγκαθιστά και εξαρτάται από τον αναμενόμενο φόρτο που θα έχει το FOXY, αλλά και από τις δυνατότητες του υπολογιστικού συστήματος στο οποίο θα τρέχει. Ο μέγιστος αριθμός ταυτόχρονων συνδέσεων, τον οποίο στη συνέχεια θα ονομάζουμε "maxconn", δεν έχει άμεση σχέση με το πόσες αιτήσεις θα εξυπηρετούνται στη μονάδα του χρόνου, αλλά σημαίνει ότι αν υπάρχουν "maxconn" συνδέσεις που βρίσκονται σε εξέλιξη κάποια συγκεκριμένη στιγμή, τότε θα πρέπει κάποια από αυτές να ολοκληρωθεί για να πάρει τη θέση της μια τυχόν νέα σύνδεση. Η τιμή του "maxconn" δεν πρέπει να είναι πολύ μικρή (π.χ. κάτω του 50) γιατί έτσι θα μειωθούν οι επιδόσεις, λόγω του ότι δεν θα μπορούμε να εκμεταλλευτούμε την καθυστέρηση του δικτύου για να εξυπηρετήσουμε πολλές αιτήσεις παράλληλα[2].

Όπως είναι κατανοητό, η έννοια του μέγιστου αριθμό ταυτόχρονων συνδέσεων προστατεύει το FOXY από επιπλέον φόρτο.

Ξεκινώντας τη λειτουργία του το FOXY διαβάζει το αρχείο με τις ρυθμίσεις του χρήστη (configuration files) που συνήθως ονομάζεται *foxy.conf*. Τις επιλογές που δίνονται στο αρχείο αυτό θα τις περιγράψουμε παρακάτω στην παράγραφο 5.2.7. Στη συνέχεια αρχικοποιείται το σύστημα διαχείρισης μνήμης, οι διάφορες μεταβλητές και τα διάφορα υποσυστήματα του FOXY τη λειτουργία των οποίων θα περιγράψουμε παρακάτω. Τέλος, το FOXY δημιουργεί ένα socket στο port που έχει οριστεί στο αρχείο

[2]Εννοούμε ότι την ώρα που περιμένουμε να δημιουργηθεί για παράδειγμα μια σύνδεση TCP για μία συγκεκριμένη αίτηση, μπορούμε να κάνουμε μια αναζήτηση στο ευρετήριο για το αντικείμενο μιας άλλης αίτησης.

επιλογών (configuration file) από το χρήστη (με την επιλογή *foxy__port_num*), έτσι ώστε να δέχεται συνδέσεις από Πελάτες.

Γνωρίζοντας το μέγιστο όριο των συνδέσεων που θα εξυπηρετεί ταυτόχρονα, το FOXY επιλέγει το δεύτερο τρόπο εξυπηρέτησης αιτήσεων που περιγράψαμε στην παράγραφο 5.2.1, χρησιμοποιώντας την κλήση συστήματος select(). Αφού λοιπόν γίνουν οι αρχικοποιήσεις, το FOXY μπαίνει σε ένα ατέρμονο βρόχο όπου σε πρώτη καλείται πάντα η συνάρτηση foxy_select_fd() περιμένοντας συνδέσεις. Η συνάρτηση foxy_select_fd() χρησιμοποιεί την κλήση συστήματος select() για να διαπιστώσει σε ποια σύνδεση (μέσα σε από σύνολο συνδέσεων) έχουν φτάσει δεδομένα για ανάγνωση ή μπορεί να γίνει εγγραφή νέων δεδομένων. Χρησιμοποιώντας λοιπόν την κλήση select() και non-blocking sockets μπορούμε να υλοποιήσουμε ένα ασύγχρονο τρόπο εξυπηρέτησης πολλών αιτήσεων ταυτόχρονα.

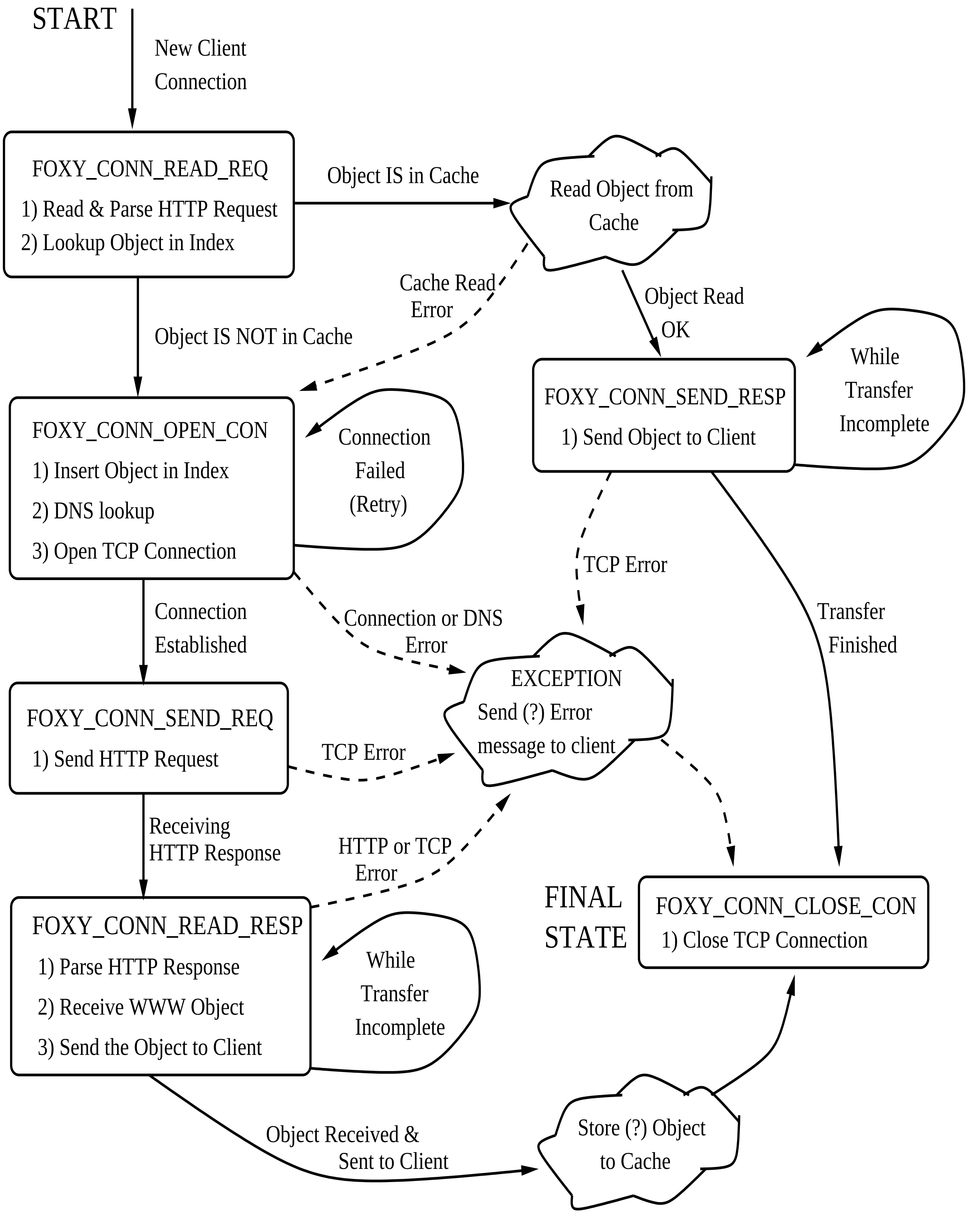
Για να το πετύχουμε αυτό, χωρίσαμε τη διαδικασία εξυπηρέτησης κάθε αίτησης σε καταστάσεις από σύγχρονες ενέργειες οι οποίες πρέπει να γίνουν από το FOXY. Αυτό σημαίνει ότι οι ενέργειες που λαβαίνουν χώρα ανάμεσα στο τέλος μιας τέτοιας κατάστασης και στην αρχή της επόμενης δεν εξαρτώνται από το FOXY, αλλά από άλλους παράγοντες όπως το λειτουργικό σύστημα (π.χ. για μια ανάγνωση δεδομένων από το δίσκο) ή από το δίκτυο (π.χ. για τη δημιουργία μιας σύνδεσης TCP με ένα μακρινό Εξυπηρέτη). Για να διαπιστώσουμε πότε ολοκληρώθηκε μια τέτοια ασύγχρονη ενέργεια χρησιμοποιούμε την κλήση select()[3]. Η υλοποίηση του βρόχου αυτού με τις καταστάσεις των συνδέσεων (τις οποίες θα περιγράψουμε παρακάτω) και της κλήσης της foxy_select_fd() φαίνεται σε ψευδοκώδικα στον πίνακα 5.1.

Παράλληλα, πρέπει να αναφέρουμε ότι η κλήση foxy_select_fd() δίνει προτεραιότητα στις ενέργειες των συνδέσεων που έχουν ήδη ξεκινήσει να εξυπηρετούνται και όχι στις νέες συνδέσεις. Όπως έδειξαν πειράματα που κάναμε αυτή η προτεραιότητα στις υπάρχουσες συνδέσεις δίνει υψηλότερο μέσο ρυθμό εξυπηρέτησης αιτήσεων, γιατί προχωρά με τη μέγιστη ταχύτητα τις ήδη υπάρχουσες συνδέσεις πελατών πριν αναλάβει καινούργιες.

Οι καταστάσεις αυτές που περιλαμβάνουν τα στάδια των σύγχρονων ενεργειών για μία σύνδεση στο FOXY, έχουν οργανωθεί στο διάγραμμα καταστάσεων που φαίνεται στο σχήμα 5.2.

Στο σχήμα αυτό, τα τετράγωνα παριστάνουν τις καταστάσεις στις οποίες μπορεί να βρεθεί μια σύνδεση ενός πελάτη, ενώ τα ονόματα τους φαίνονται με κεφαλαία γράμματα όπως ακριβώς έχουν οριστεί στον κώδικα του FOXY. Τα σύννεφα περιλαμβάνουν ενέργειες που γίνονται σύγχρονα στο πέρασμα από μια κατάσταση στην άλλη.

[3] Αν χρησιμοποιούσαμε threads θα αρκούσε να καλέσουμε blocking read() ή write() στα sockets τα οποία θα μπλόκαραν το thread μέχρι να ολοκληρωθούν οι σύγχρονες ενέργειες.




**Σχήμα 5.2** Διάγραμμα καταστάσεων του FOXY για την εξυπηρέτηση μιας σύνδεσης Πελάτη. Τα τετράγωνα παριστάνουν τις καταστάσεις, ενώ τα σύννεφα περιλαμβάνουν ενέργειες που γίνονται σύγχρονα στο πέρασμα από μια κατάσταση στην άλλη.

```
for (;;) { /* For ever ... */
    connection_ready = foxy_select_fd();
    switch (connection_ready.state) {
        case FOXY_CONN_READ_REQ:
            ...
        break;
        case FOXY_CONN_SEND_REQ:
            ...
        break;
        case FOXY_CONN_READ_RESP:
            ...
        break;
        case FOXY_CONN_OPEN_CON:
            ...
        break;
        case FOXY_CONN_SEND_RESP:
            ...
        break;
        case FOXY_CONN_CLOSE_CONN:
            ...
        break;
    }
}
```

**Πίνακας 5.1** Βρόχος ασύγχρονης επιλογής συνδέσεων στο FOXY.

Αναλυτικά οι λειτουργίες που ενεργοποιούνται στις καταστάσεις του σχήματος 5.2 είναι:

◇ **FOXY_CONN_READ_REQ**
Στην κατάσταση αυτή έχει μόλις δημιουργηθεί μία νέα σύνδεση από ένα πελάτη. Αφού κληθεί η συνάρτηση foxy_register_new_connection() η οποία αρχικοποιεί τις μεταβλητές που χρειάζονται για τα μεταδεδομένα της σύνδεσης και περιλαμβάνονται σε δύο πίνακες δομών τύπου Foxy_Connection_t και Foxy_Connection_Buffer_t. Στη συνέχεια καλείται η συνάρτηση read_and_parse_http_request() που (όπως είναι προφανές από το όνομά της) διαβάζει την αίτηση HTTP από τον Πελάτη, διαχωρίζει τις χρήσιμες πληροφορίες και τις επιστρέφει μέσα στις δομές της σύνδεσης (Foxy_Connection_t

και Foxy_Connection_Buffer_t).

Προχωρώντας, το FOXY καλεί τη συνάρτηση foxy_index_lookup_object_fill() η οποία αναζητά το αντικείμενο της αίτησης μέσα στο ευρετήριο (index) και, αν το βρει, επιστρέφει τα μεταδεδομένα του σε μια δομή τύπου Foxy_Object_Index_t. Αν το αντικείμενο βρεθεί τότε διαβάζεται κατευθείαν από την μνήμη, όπως φαίνεται στο σχήμα και στη συνέχεια προχωράμε σύγχρονα στην κατάσταση FOXY_CONN_SEND_RESP. Αντίθετα, αν το αντικείμενο δεν βρεθεί προχωράμε κατευθείαν στην κατάσταση FOXY_CONN_OPEN_CON. Αν τέλος το αντικείμενο βρεθεί στο ευρετήριο, αλλά γίνει κάποιο λάθος στη διαδικασία ανάκτησής του, τότε όπως φαίνεται στο σχήμα, μεταφερόμαστε στην κατάσταση FOXY_CONN_OPEN_CON συμπεριφερόμενοι σαν να μην είχαμε βρει το αντικείμενο στο ευρετήριο.

◇ **FOXY_CONN_OPEN_CON**

Στην κατάσταση αυτή γνωρίζουμε ότι το αντικείμενο δεν βρίσκεται στη μνήμη του FOXY και πρέπει να το αναχτήσουμε από το μακρινό Εξυπηρέτη WWW. Η πρώτη μας ενέργεια είναι να εισάγουμε το αντικείμενο στο ευρετήριο θέτοντας την κατάστασή του σε FOXY_INDEX_STATE_RETR που σημαίνει ότι αυτή τη στιγμή μεταφέρεται. Αυτό γίνεται γιατί μπορεί κάποιος πελάτης να ξαναζητήσει το ίδιο αντικείμενο όση ώρα το μεταφέρουμε, και αν δεν το βρει ευρετηριασμένο θα ανοίξει δεύτερη σύνδεση για το ίδιο αντικείμενο. Για να αποφύγουμε τέτοια προβλήματα, έχουμε αναπτύξει στο FOXY ένα μηχανισμό callback, στον οποίο η δεύτερη σύνδεση Πελάτη (αλλά και οι επόμενες) για αντικείμενο που βρίσκεται υπό μεταφορά θέτει ένα callback στην σύνδεση που έχει αρχίσει να μεταφέρει το αντικείμενο. Όταν λοιπόν ολοκληρωθεί η μεταφορά του αντικειμένου, η σύνδεση που το μετέφερε θα ελέγξει για τυχόν callbacks και θα εξυπηρετήσει τις συνδέσεις που τα έχουν θέσει. Στην υλοποίησή μας, για να έχουμε καλύτερη διαχείριση μνήμης, έχουμε ένα μέγιστο όριο στα callbacks που μπορεί να τεθούν σε μια σύνδεση, υποθέτοντας ότι δεν υπάρχουν πολλές προσπάθειες για την ανάκτηση ενός αντικειμένου στο σχετικά μικρό χρονικό διάστημα που διαρκεί η μεταφορά του.

Στη συνέχεια της κατάστασης FOXY_CONN_OPEN_CON, το FOXY κάνει μια αναζήτηση DNS για το όνομα του Εξυπηρέτη WWW. Η αναζήτηση αυτή γίνεται σύγχρονα στην παρούσα υλοποίηση του FOXY όπου θεωρήσαμε ότι δεν επηρεάζει το χρόνο των πειραμάτων μας, λόγω της υλοποίησης μιας DNS cache. Η DNS cache μας επέτρεψε να μειώσουμε στο ελάχιστο τις αναζητήσεις DNS αποθηκεύοντας τις τελευταίες διευθύνσεις IP των Εξυπηρετών WWW. Τέτοιες

DNS caches χρησιμοποιούν τα περισσότερα Web proxies σήμερα, συμπεριλαμβανομένου και του Squid.

Μετά την αναζήτηση DNS, ανοίγεται μια σύνδεση με τον Εξυπηρέτη WWW με κλήση της ρουτίνας connect(), κάτι που μπορεί να διαρκέσει αρκετή ώρα. Γι' αυτό το λόγο γίνεται ασύγχρονα, δηλαδή θέτοντας το αρχικό μας socket non-blocking, η connect() επιστρέφει αμέσως και μας επιτρέπει να φροντίσουμε μια άλλη σύνδεση. Σε μελλοντική κλήση της foxy_select_fd() θα διαπιστώσουμε αν η connect() επέστρεψε επιτυχώς, οπότε και περνάμε στην κατάσταση FOXY_CONN_SEND_REQ. Σε περίπτωση άρνησης της σύνδεσης, επιχειρούμε να ξανασυνδεθούμε μερικές φορές (όπως ορίζεται από τη σταθερά FOXY_CONN_MAX_RETRIES). Αν αποτύχουμε, τότε επιστρέφουμε ένα μήνυμα λάθους στον Πελάτη (όπως φαίνεται στο σχήμα 5.2) και κλείνουμε τη σύνδεση.

◇ **FOXY_CONN_SEND_REQ**

Στην κατάσταση αυτή έχει ολοκληρωθεί η σύνδεσή μας με τον Εξυπηρέτη WWW, οπότε στέλνουμε μία αίτηση HTTP σε αυτόν για το αντικείμενο, και επειδή η απάντηση μπορεί να καθυστερήσει επιστρέφουμε στο βρόχο καλώντας την foxy_select_fd() η οποία θα μας ειδοποιήσει όταν ο Εξυπηρέτης WWW έχει στείλει δεδομένα στη σύνδεσή μας. Σε αυτή την περίπτωση περνάμε στην επόμενη κατάσταση, την FOXY_CONN_READ_RESP.

◇ **FOXY_CONN_READ_RESP**

Στην κατάσταση αυτή διαβάζουμε τα δεδομένα που έχει στείλει ο Εξυπηρέτης WWW και διαχωρίζουμε καταρχήν αν η αίτησή μας ήταν επιτυχημένη. Σε περίπτωση αποτυχίας κλείνουμε τη σύνδεση με τον Εξυπηρέτη WWW και αφού επιστρέψουμε μήνυμα λάθους στον Πελάτη, μεταβαίνουμε στην κατάσταση FOXY_CONN_CLOSE_CONN.

Αντίθετα σε περίπτωση επιτυχημένης αίτησης HTTP, μεταφέρουμε το αντικείμενο από τον Εξυπηρέτη WWW σε έναν buffer στη μνήμη του FOXY. Η μεταφορά γίνεται συνήθως σιγά σιγά, ειδικά για μεγάλα αντικείμενα, οπότε αφού διαβάσουμε κάθε φορά τα δεδομένα που μας έχει στείλει ο Εξυπηρέτης WWW επιστρέφουμε στον κεντρικό βρόχο και αφήνουμε την foxy_select_fd() να μας ειδοποιήσει όταν υπάρχουν νέα δεδομένα στη σύνδεσή μας[4]. Η επιστροφή των περιεχομένων του αντικειμένου στον Πελάτη, γίνεται επίσης συγχρόνως με τη μεταφορά τους από τον Εξυπηρέτη. Δεν περιμένουμε δηλαδή να φέρουμε όλο

[4]Αν χρησιμοποιούσαμε threads θα αρκούσε να καλέσουμε ένα blocking read() στο socket το οποίο θα μπλοκάρει το thread μέχρι να φτάσουν τα δεδομένα.

το αντικείμενο και μετά να το δώσουμε μαζεμένο στον Πελάτη, γιατί έτσι αυξάνουμε τη μέση καθυστέρηση της αίτησης. Επίσης με την επαναχρησιμοποίηση μνήμης, γλιτώνουμε το χώρο που θα ξοδεύαμε για την αποθήκευση ολόκληρων μεγάλων αντικειμένων.

Στην εξέλιξη της κατάστασης αυτής και αφού το αντικείμενο έχει επιστραφεί στον Πελάτη, αποφασίζουμε αν θα το κρατήσουμε στη μνήμη του FOXY. Στην παρούσα υλοποίηση η απόφαση αυτή εξαρτάται μόνο από το μέγεθος του αντικειμένου. Αποθηκεύουμε δηλαδή τα μικρά σε μέγεθος αντικείμενα και αποφεύγουμε τα μεγαλύτερα. Το όριο των αντικειμένων που θα αποθηκευτούν στην μνήμη του FOXY ελέγχεται από την επιλογή *maxim_obj_size* που ορίζεται στο αρχείο επιλογών χρήστη *foxy.conf*. Αν αποφασίσουμε να το αποθηκεύσουμε στη μνήμη, το εισάγουμε στην ιεραρχία μνήμης του FOXY που μπορεί να αποτελείται από ένα ή δύο επίπεδα. Στην περίπτωση που έχουμε δύο επίπεδα, σαν πρώτο επίπεδο χρησιμοποιείται ένα μέρος κύριας μνήμης και σαν δεύτερο επίπεδο χώρος στο μαγνητικό δίσκο. Τα μεγέθη αυτά ορίζονται επίσης στο αρχείο επιλογών χρήστη *foxy.conf*.

Σε κάθε επίπεδο μνήμης υπάρχει μια ξεχωριστή πολιτική αντικατάστασης αντικειμένων, η οποία για το πρώτο επίπεδο μνήμης αντικαθιστά αντικείμενα μεταφέροντάς τα στο δεύτερο επίπεδο, ενώ όσα αντικαθιστώνται από το δεύτερο επίπεδο (δηλαδή από το δίσκο), διαγράφονται. Η πολιτική που υλοποιήσαμε και στα δύο επίπεδα είναι η LRU-TH[5] [15] (το threshold ελέγχεται στην απόφαση για είσοδο των αντικειμένων στη μνήμη). Περισσότερες λεπτομέρειες για την ιεραρχία μνήμης του FOXY και την πολιτική αντικατάστασης αντικειμένων που υλοποιήσαμε στην παράγραφο 5.2.3.

◇ **FOXY_CONN_SEND_RESP**

Στην κατάσταση αυτή η αναζήτηση του αντικειμένου στο ευρετήριο ήταν επιτυχής το οποίο σημαίνει ότι το αντικείμενο βρίσκεται στη μνήμη του FOXY και όπως περιγράψαμε στην κατάσταση FOXY_CONN_READ_REQ έχουμε ήδη διαβάσει τα περιεχόμενα του αντικειμένου από τη μνήμη. Μεταφέρουμε λοιπόν τα περιεχόμενα αυτά στον Πελάτη χρησιμοποιώντας την κλήση συστήματος write() και εφόσον δεν συμβεί κανένα λάθος στη σύνδεσή μας θέτουμε την κατάσταση της σύνδεσης σε FOXY_CONN_CLOSE_CONN, και επιστρέφουμε στον κυρίως βρόχο. Αυτό γίνεται γιατί ίσως τα δεδομένα να χρειάζονται χρόνο για να μεταφερθούν μέχρι τον Πελάτη. Η ολοκλήρωση της μεταφοράς θα διαπιστωθεί με την κλήση της foxy_select_fd() στην αρχή του βρόχου.

[5] Βλέπε παράγραφο 1.3.2.

◇ **FOXY_CONN_CLOSE_CONN**

Στην τελευταία αυτή κατάσταση, είτε η αίτηση έχει εξυπηρετηθεί και τα περιεχόμενα του αντικειμένου βρίσκονται στο Πελάτη, είτε έχει συμβεί κάποιο λάθος και έχει σταλεί μήνυμα λάθους στον τελευταίο. Σε κάθε περίπτωση, η αίτηση έχει ολοκληρωθεί και είμαστε έτοιμοι να κλείσουμε τη σύνδεση.

Προλαβαίνοντας τυχόν ερώτημα του αναγνώστη όσον αφορά τη χρήση μόνιμων συνδέσεων όπως υποστηρίζει η έκδοση 1.1 του πρωτοκόλλου HTTP πρέπει να αναφέρουμε ότι στην παρούσα έκδοση δεν υλοποιήσαμε τέτοιες συνδέσεις, γιατί δε βρήκαμε κανένα αξιόλογο μετροπρόγραμμα (benchmark) για Web proxies που να τις υποστηρίζει. Έτσι τυχόν υλοποίησή τους δε θα ήταν χρήσιμη στα πειράματα. Πάντως στην παρούσα υλοποίηση, η πρόσθεση τέτοιων συνδέσεων μπορεί να γίνει πολύ εύκολα, αν δεν τερματίσουμε τη σύνδεση και μεταβούμε στην κατάσταση FOXY_CONN_READ_REQ όπου θα εξυπηρετήσουμε τη νέα αίτηση.

### 5.2.3 Η ΙΕΡΑΡΧΙΑ ΜΝΗΜΗΣ ΚΑΙ Η ΠΟΛΙΤΙΚΗ ΑΝΤΙΚΑΤΑΣΤΑΣΗΣ

Όπως αναφέραμε και παραπάνω το FOXY μπορεί να έχει δύο επίπεδα ιεραρχίας μνήμης, το πρωτεύον στην κύρια μνήμη και το δευτερεύον στο δίσκο, ή μόνο ένα επίπεδο στο δίσκο. Η ύπαρξη του επιπέδου κύριας μνήμης επιλέγεται από τον χρήστη μέσω της επιλογής *FOXY_USE_MEMCACHE* στο αρχείο *foxy.h*. Ωστόσο η ύπαρξη του επιπέδου κύριας μνήμης δεν δίνει υψηλότερες επιδόσεις, γιατί τον ίδιο ρόλο μπορεί να παίξει η κρυφή μνήμη του συστήματος αρχείων που παρέχεται από τα περισσότερα λειτουργικά συστήματα και λειτουργεί με τον αλγόριθμο LRU. Όπως όμως διαπιστώσαμε από τα πειράματα που έγιναν με το FOXY πολλές φορές η κρυφή μνήμη του συστήματος αρχείων δεν έχει την αναμενόμενη συμπεριφορά (δηλαδή σε πολλές περιπτώσεις δεν χρησιμοποιεί τον αλγόριθμο LRU), και άρα τότε το πρώτο αυτό επίπεδο μνήμης μπορεί να μειώσει πολύ τις αναγνώσεις στο δίσκο και κατά συνέπεια να αυξήσει την απόδοση του Web proxy.

Η λειτουργία των επιπέδων μνήμης είναι ανεξάρτητη και στηρίζεται στην ύπαρξη δύο οριακών αριθμών, του "υψηλού ορίου πληρότητας" ("high watermark") και του "χαμηλού ορίου πληρότητας" ("low watermark"). Καθένας από τους δύο αποτελεί ένα εκατοστιαίο ποσοστό και ο ορισμός τους γίνεται στο αρχείο επιλογών χρήστη *foxy.conf* (παράμετροι mem_high_wmark, mem_low__wmark για την κύρια μνήμη και disk_hi__wmark, disk_low_wmark για το δίσκο. Ο αλγόριθμος λειτουργίας των δύο επιπέδων μνήμης λειτουργεί λοιπόν με τον εξής τρόπο:

◇ Αρχικά, εφόσον υπάρχει το επίπεδο κύριας μνήμης και η πληρότητα της μνήμης αυτής είναι μικρότερη του "υψηλού ορίου πληρότητας", το αντικείμενο γράφεται εκεί. Αν δεν υπάρχει τέτοιο επίπεδο μνήμης, και η πληρότητα του δίσκου είναι μικρότερη του "υψηλού ορίου πληρότητας", το αντικείμενο γράφεται στο δίσκο.

◇ Εφόσον υπάρχει το επίπεδο κύριας μνήμης και σε κάποια εγγραφή αντικειμένου το όριο πληρότητας ξεπεράσει το "υψηλό όριο πληρότητας", τότε καλείται η πολιτική αντικατάστασης για την πρωτεύουσα μνήμη, η οποία αντικαθιστά αντικείμενα από την κύρια μνήμη εισάγοντάς τα στη μνήμη του δίσκου μέχρι η πληρότητα να φθάσει στο "χαμηλό όριο πληρότητας".

◇ Εφόσον τέλος σε κάποια εγγραφή αντικειμένου στο δίσκο το όριο πληρότητας ξεπεράσει το "υψηλό όριο πληρότητας", τότε καλείται η πολιτική αντικατάστασης για το δίσκο, η οποία διαγράφει αντικείμενα από το δίσκο μέχρι η πληρότητά του να φτάσει στο "χαμηλό όριο πληρότητας".

Η πολιτική αντικατάστασης που υλοποιήθηκε είναι η LRU-TH [15], και υλοποιήθηκε με μία λίστα τελευταίας προσπέλασης αντικειμένων. Η λίστα αυτή ενημερώνεται σε κάθε προσπέλαση ενός νέου ή παλιότερου αντικειμένου τοποθετώντας το στην κορυφή της. Έτσι όταν πρόκειται να γίνει αντικατάσταση αντικειμένων, τα αντικείμενα προς αντικατάσταση βρίσκονται στο τέλος της λίστας.

### 5.2.4 ΤΟ ΣΥΣΤΗΜΑ ΕΥΡΕΤΗΡΙΑΣΜΟΥ (INDEXING)

Το σύστημα ευρετηριασμού είναι υπεύθυνο για τον ευρετηριασμό των μεταδεδομένων των αντικειμένων που έχει αποθηκεύσει το FOXY. Ο κώδικάς του περιέχεται στο αρχείο foxy_index.c. Το ευρετήριο έχει υλοποιηθεί με τη μορφή ενός hash table με κλειδί που υπολογίζεται (μέσω μιας συνάρτησης) από το όνομα του αντικειμένου (δηλαδή το URL του). Στην παρούσα υλοποίηση η συνάρτηση αυτή υπολογίζει το κλειδί από το άθροισμα των χαρακτήρων του URL modulo τον αριθμό των buckets.

Ο αριθμός των buckets που χρησιμοποιεί το hash table ορίζεται στο αρχείο foxy.h από τη σταθερά FOXY_INDEX_BUCKETS, ενώ καθένας από αυτούς έχει χωρητικότητα για μεταδεδομένα "FOXY_INDEX_ALLOC" αντικειμένων. Εφόσον ο αριθμός FOXY_INDEX_ALLOC είναι σταθερός, μπορεί κάποιος bucket να γεμίσει. Στην περίπτωση αυτή χρησιμοποιώντας την κλήση realloc() αυξάνουμε το μέγεθός του κατά FOXY_INDEX_ALLOC.

### 5.2.5 ΑΛΓΟΡΙΘΜΟΙ I/O ΣΤΟ ΔΙΣΚΟ

Για την εγγραφή αντικειμένων στο δίσκο υλοποιήθηκε η πολιτική STREAM-PACKETIZER την οποία περιγράψαμε στην παράγραφο 4.3. Για την εγγραφή στο δίσκο χρησιμοποιείται ένα μόνο αρχείο και τα δεδομένα ομαδοποιούνται μέσα σε ένα buffer που το μέγεθός του ορίζεται από τη σταθερά *FOXY_WRITE_BUFSIZE* στο αρχείο *foxy.h*. Πάντα λοιπόν γράφουμε (καλώντας την κλήση συστήματος *write()*) έναν buffer κάθε φορά στο αρχείο του δίσκου. Όταν φτάσουμε στο σημείο που ξεπερνάμε το "υψηλό όριο πληρότητας" στο δίσκο, καλείται η πολιτική αντικατάστασης που διαγράφει αντικείμενα μέχρι η πληρότητα του δίσκου να φτάσει στο "χαμηλό όριο πληρότητας". Συνεχίζοντας να γράφουμε, όταν φτάσουμε στο τέλος του διαθέσιμου χώρου (που έχει οριστεί από το χρήστη με την επιλογή *foxy_cache_dir* στο αρχείο *foxy.conf*) επαναφέρουμε το δείκτη αρχείου (file pointer) στην αρχή και συνεχίζουμε τις εγγραφές. Ο κώδικας για το χειρισμό των δίσκων περιέχεται στο αρχείο foxy_disk.c.

Η πολιτική LAZY-READS-LOC που περιγράφηκε στην παράγραφο 4.4.2 δεν υλοποιήθηκε γιατί η απόδοσή της δεν μπορούσε να μετρηθεί με μετροπρογράμματα. Αυτό είναι κατανοητό διότι η πολιτική αυτή χρησιμοποιεί τους "locality buffers" για την ομαδοποίηση των αντικειμένων που προέρχονται από τον ίδιο Εξυπηρέτη. Τα μετροπρογράμματα όμως τρέχουν σε ένα ή δύο Εξυπηρέτες και οι αιτήσεις τους δεν παρουσιάζουν τοπικότητα στα αντικείμενα που προέρχονται από την ίδια ιστοσελίδα. Αρα η απόδοση της πολιτικής αυτής μπορεί να μετρηθεί μόνο με προσομοίωση πραγματικών traces. Τέλος η υλοποίηση της πολιτικής LAZY-READS, αν και αποτελεί σχετικά απλή προσθήκη στον υπάρχοντα κώδικα για την STREAM-PACKETIZER, δεν υλοποιήθηκε λόγω έλλειψης χρόνου.

### 5.2.6 ΔΙΑΧΕΙΡΙΣΗ ΜΝΗΜΗΣ

Η διαχείριση δυναμικής μνήμης που παρείχαν οι βιβλιοθήκες του λειτουργικού συστήματος δημιούργησε πολλές φορές προβλήματα κατά τη διάρκεια των πειραμάτων μας. Ειδικότερα τα προβλήματα οφείλονταν τις περισσότερες φορές στην κλήση της realloc() που χρησιμοποιήθηκε για την αύξηση του χώρου δυναμικής μνήμης που καταλάμβαναν ορισμένες δομές δεδομένων, όπως για παράδειγμα οι buckets του hash table στο ευρετήριο. Εφόσον οι προσπάθειες που κάναμε με διάφορα τεχνάσματα απέτυχαν, καταφύγαμε στην υλοποίηση ενός δικού μας υποσυστήματος διαχείρισης δυναμικής μνήμης που χρησιμοποιεί μόνο την κλήση malloc() για να δεσμεύσει αρχικά κάποια μνήμη από το σύστημα την οποία στη συνέχεια διαχειρίζεται αυτόνομα.

Ο κώδικας του υποσυστήματος διαχείρισης δυναμικής μνήμης του FOXY περιέχεται στο αρχείο foxy_mm.c. Η υλοποίησή μας διαχειρίζεται τη μνήμη σε μορφή σταθερού μεγέθους "κελιών μνήμης" (memory cells) που τη θεωρούμε σαν τη μικρότερη μονάδα μνήμης που μπορούμε να διαχειριστούμε και στην υλοποίησή μας είναι μεγέθους 64 bytes. Η διαχείριση του συνόλου αυτού των "κελιών μνήμης" γίνεται μέσω ενός χάρτη από bits (bitmap) που υλοποιήσαμε. Για τη χρήση αυτού του υποσυστήματος διαχείρισης δυναμικής μνήμης από το FOXY χρησιμοποιούνται οι κλήσεις foxy_mem_alloc(), foxy_mem_free() και foxy_mem_realloc() αντί των αντίστοιχων malloc(), free() και realloc() των βιβλιοθηκών του συστήματος.

### 5.2.7 ΕΠΙΛΟΓΕΣ ΧΡΗΣΤΗ ΓΙΑ ΤΗ ΛΕΙΤΟΥΡΓΙΑ ΤΟΥ FOXY

Οι επιλογές του χρήστη για την επιθυμητή λειτουργία του FOXY δηλώνονται σε δύο χρονικές στιγμές, τη στιγμή μετάφρασης του πηγαίου κώδικα όπου διαβάζεται το αρχείο των σταθερών *foxy.h* και τη στιγμή εκτέλεσης του FOXY όπου διαβάζεται το αρχείο επιλογών (configuration file) *foxy.conf*.

Το αρχείο επιλογών, όπως αναφέραμε και παραπάνω ονομάζεται *foxy.conf*, αλλά το όνομά του μπορεί να αλλάξει μεταβάλλοντας την τιμή της σταθεράς *FOXY_CONFIG_FILENAME* στο αρχείο *foxy.h*. Οι επιλογές του χρήστη στο αρχείο αυτό είναι οι παρακάτω:

◇ **foxy_port_num**
Ορίζει τον αριθμό του port όπου το FOXY θα περιμένει TCP συνδέσεις.

◇ **maxim_obj_size**
Ορίζει το μέγιστο όριο μεγέθους των αντικειμένων που θα αποθηκευτούν στην μνήμη του FOXY. Αντικείμενα μεγαλύτερα από αυτό το μέγεθος θα επιστραφούν στον Πελάτη χωρίς την αποθήκευσή τους στη μνήμη του FOXY.

◇ **foxy_mem_size**
Ορίζει το μέγεθος σε bytes του πρωτεύοντος επιπέδου κύριας μνήμης (αν υπάρχει).

◇ **foxy_num_disks**
Ορίζει τον αριθμό των δίσκων που θα χρησιμοποιηθούν για την αποθήκευση αντικειμένων. Ο συγκεκριμένος ορισμός θα έχει μελλοντική χρήση, αφού στην παρούσα έκδοση χρησιμοποιούμε μόνο ένα δίσκο.

◇ **foxy_cache_dir**
Ορίζει το μέγεθος σε bytes του δευτερεύοντος επιπέδου μνήμης δηλαδή του δίσκου, όπως επίσης τον κατάλογο και το όνομα αρχείου στο οποίο θα αποθηκευτούν τα αντικείμενα.

◇ **mem_high_wmark**
Ορίζει σε εκατοστιαίο ποσοστό το "υψηλό όριο πληρότητας" του πρωτεύοντος επιπέδου κύριας μνήμης (αν υπάρχει).

◇ **mem_low_wmark**
Ορίζει σε εκατοστιαίο ποσοστό το "χαμηλό όριο πληρότητας" του πρωτεύοντος επιπέδου κύριας μνήμης (αν υπάρχει).

◇ **disk_hi_wmark**
Ορίζει σε εκατοστιαίο ποσοστό το "υψηλό όριο πληρότητας" του δευτερεύοντος επιπέδου μνήμης δηλαδή του δίσκου.

◇ **disk_low_wmark**
Ορίζει σε εκατοστιαίο ποσοστό το "χαμηλό όριο πληρότητας" του δευτερεύοντος επιπέδου μνήμης δηλαδή του δίσκου.

◇ **foxy_log_state**
Ορίζει αν είναι επιθυμητό το FOXY να τυπώνει διαγνωστικά μυνήματα, και με ποια έξοδο (το τερματικό, ένα αρχείο ή και τα δύο).

◇ **foxy_log_filen**
Ορίζει το όνομα αρχείου σε περίπτωση εκτύπωσης διαγνωστικών μυνήματων σε αρχείο.

# Πειραματικά Αποτελέσματα

The sheer volumne of answers can often stifle insight ... The purpose of computing is insight, not numbers.

—Hamming

## 6.1 ΠΕΙΡΑΜΑΤΙΚΟ ΠΕΡΙΒΑΛΛΟΝ

Για τα πειράματά μας με το Squid και το FOXY δοκιμάσαμε δύο πολύ γνωστά μετροπρογράμματα, το Wisconsin Proxy Benchmark [49] και το Web Polygraph [5]. Τελικά για τους λόγους που θα εξηγήσουμε παρακάτω, επιλέξαμε το δεύτερο για τις τελικές μετρήσεις μας.

### 6.1.1 ΤΟ ΜΕΤΡΟΠΡΟΓΡΑΜΜΑ WISCONSIN PROXY BENCHMARK

Το μετροπρόγραμμα "Wisconsin Proxy Benchmark" [49] δημιουργήθηκε στο Πανεπιστήμιο του Wisconsin από την ερευνητική ομάδα της Pei Cao με σκοπό τη μέτρηση της απόδοσης των Web proxies. Μέχρι την περίοδο εκείνη δεν υπήρχαν αξιόπιστα μετροπρογράμματα και το Wisconsin Proxy Benchmark αποτέλεσε ένα πρώτο βήμα.

Τα δικά μας πειράματα ξεκίνησαν επίσης με το Wisconsin Proxy Benchmark, γιατί όταν ξεκινήσαμε την ενασχόλησή μας με το θέμα των Web proxies δεν υπήρχε άλλος αξιόπιστος τρόπος μέτρησης. Αφότου όμως αρχίσαμε να κάνουμε μετρήσεις, αρχίσαμε να παρατηρούμε τα μειονεκτήματα του Wisconsin Proxy Benchmark. Το πρώτο βασικό πρόβλημα ήταν ότι το benchmark δημιουργούσε υπερβολικό φόρτο στο υπολογιστικό σύστημα που έτρεχε, λόγω του γεγονότος ότι οι Εξυπηρέτες WWW που παρείχε, δημιουργούσαν μία καινούρια θυγατρική διεργασία για κάθε αίτηση που εξυπηρετούσαν χρησιμοποιώντας την κλήση συστήματος *fork()*. Με αυτόν τον τρόπο, ο φόρτος του συστήματος από το benchmark ήταν περισσότερος συνήθως από αυτόν του Web proxy που έπρεπε να μετρήσουμε. Όπως είναι κατανοητό, το γεγονός αυτό εκτός του ότι μείωνε την αξιοπιστία των μετρήσεων, μείωνε σε μεγάλο βαθμό και τις επιδόσεις του Web proxy.

Η λύση μας στο πρόβλημα αυτό ήταν να μετατρέψουμε εμείς το Wisconsin Proxy

Benchmark ώστε να χρησιμοποιεί ξεχωριστά threads αντί για διεργασίες για την εξυπηρέτηση των αιτήσεων. Καταφέραμε έτσι να λύσουμε σε μεγάλο βαθμό το πρόβλημα, παρ' όλα αυτά αντιμετωπίσαμε κι άλλα προβλήματα που είχαν να κάνουν με τους περιορισμούς στο υπολογιστικό και δικτυακό υλικό του πειραματικού μας περιβάλλοντος. Ψάχνοντας για κάποια λύση ανακαλύψαμε την ύπαρξη ενός περισσότερο αξιόπιστου και χρήσιμου benchmark που μας έδινε πολύ περισσότερες επιλογές στα πειράματά μας.

Τα κύρια σημεία που μας ώθησαν στην αντικατάσταση του Wisconsin Proxy Benchmark με το Web Polygraph είναι τα ακόλουθα:

◇ Το Web Polygraph δημιουργούσε πολύ μικρότερο φόρτο στο υπολογιστικό σύστημα αφήνοντας στο Web proxy υπό μέτρηση μεγαλύτερα περιθώρια για χρήση των πόρων του συστήματος. Αυτό ήταν ένα κρίσιμο χαρακτηριστικό, δεδομένου ότι θέλουμε να μετρήσουμε Web proxies υψηλών επιδόσεων.

◇ Το Web Polygraph μας παρείχε πολύ περισσότερα αποτελέσματα στα πειράματα, όπως για παράδειγμα το μέσο ρυθμό εξυπηρέτησης αιτήσεων, τη μέση καθυστέρηση, το μέσο ποσοστό επιτυχημένων αιτήσεων (document hit ratio) και δεδομένων (byte hit ratio), την κατανομή των μεγεθών των αρχείων και πολλά άλλα. Αντίθετα, το Wisconsin Proxy Benchmark τύπωνε σε ένα αρχείο μόνο το χρόνο εκτέλεσης κάθε αίτησης και το μέγεθός της. Φιλτράροντας εμείς (με δικό μας φίλτρο) αυτά τα δεδομένα, μετρούσαμε τα διάφορα μεγέθη που μας ενδιέφεραν με πρώτο το μέσο ρυθμό εξυπηρέτησης αιτήσεων.

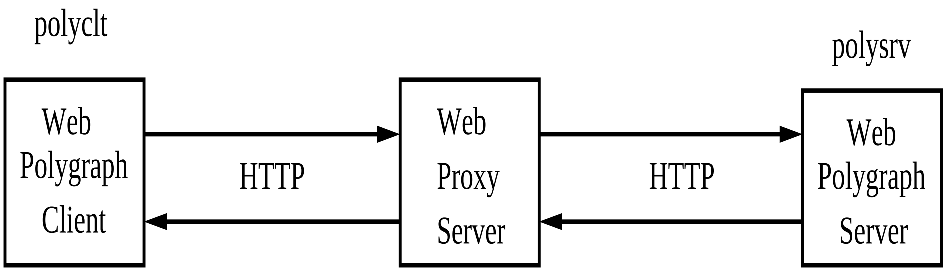
◇ Το Web Polygraph μας επέτρεπε να ορίσουμε ένα δικό μας workload για το πείραμα δίνοντάς μας πάρα πολλές επιλογές που δε μας έδινε το Wisconsin Proxy Benchmark, όπως τη δημιουργία φάσεων του πειράματος με ξεχωριστά στατιστικά για κάθε μία, τον ορισμό της κατανομής του μεγέθους των αρχείων (π.χ. constant,exponential κτλ.), τον ορισμό του μοντέλου επιλογής των δημοφιλών αντικειμένων (zipf,uniform,κτλ.), το μοντέλο μεταβολής των αντικειμένων στο χρόνο και πολλά άλλα μεγέθη.

◇ Τέλος, το Web Polygraph αναπτύχθηκε από την ίδια ομάδα προγραμματιστών που ανέπτυξαν το Squid και έχει ήδη καθιερωθεί σαν εκ-των-πραγμάτων πρότυπο (de-facto standard) για τη μέτρηση κάθε είδους Web proxies, συμπεριλαμβανομένων και αυτών που κατασκευάζονται από μεγάλες εταιρείες [19], [80].

### 6.1.2 ΤΟ ΜΕΤΡΟΠΡΟΓΡΑΜΜΑ WEB POLYGRAPH

Για τους λόγους που αναπτύξαμε αμέσως παραπάνω, χρησιμοποιήσαμε για τα πειράματά μας την έκδοση 2.2.9 του μετροπρογράμματος Web Polygraph. Σε αυτή την παράγραφο περιγράφουμε περιληπτικά τη λειτουργία του μετροπρογράμματος και τις παραμέτρους που χρησιμοποιήσαμε στα πειράματά μας.

Το Web Polygraph μετράει τη σωστή λειτουργία και την απόδοση ενός Web proxy, δημιουργώντας δύο διεργασίες, από τις οποίες η μία προσομοιώνει τη συμπεριφορά των Πελατών που στέλνουν αιτήσεις στο Web proxy, και η άλλη τη συμπεριφορά των Εξυπηρετών WWW από τους οποίους ανακτώνται τα αντικείμενα των αιτήσεων αυτών. Ο τρόπος αυτός λειτουργίας του Web Polygraph απεικονίζεται στο σχήμα 6.1, όπου οι διεργασίες polyclt και polysrv αντιστοιχούν στις διεργασίες των Πελατών και των Εξυπηρετών αντίστοιχα, οι οποίες μπορεί να τρέχουν στα ίδια ή σε διαφορετικά μηχανήματα χρησιμοποιώντας τοπικό δίκτυο για τις αιτήσεις στο Web proxy. Όπως είναι κατανοητό με τον παραπάνω τρόπο, η μέτρηση της λειτουργίας του Web proxy γίνεται σε πραγματικές συνθήκες, παρέχοντάς μας όμως παράλληλα τη δυνατότητα να ρυθμίσουμε εμείς τα χαρακτηριστικά του workload που το benchmark ακολουθεί.




**Σχήμα 6.1** Διάγραμμα λειτουργίας του Web Polygraph.

Τα χαρακτηριστικά αυτά μπορεί να είναι ο αριθμός πελατών, η κατανομή του μεγέθους των αντικειμένων των αιτήσεων, η κατανομή επιλογής των δημοφιλέστερων αντικειμένων, στατικά ή δυναμικά αντικείμενα, ο χρόνος μεταβολής των αντικειμένων κτλ. Για τις ανάγκες των πειραμάτων μας επιλέξαμε τις παραμέτρους που περιγράφονται στην παρακάτω παράγραφο. Πρέπει να αναφέρουμε ότι η επιλογή μας είχε το σκεπτικό της μοντελοποίησης του πραγματικού φόρτου που δέχεται ένα Web proxy σήμερα με τέτοιο τρόπο ώστε να μπορέσουμε να μετρήσουμε τις μέγιστες επιδόσεις του (κυρίως το ρυθμό εξυπηρέτησης αιτήσεων αλλά και τη μέση καθυστέρηση ανά αίτηση).

Μία προϋπόθεση για να μετρηθεί η μέγιστη απόδοση του Web proxy είναι να έχουμε αρκετά γρήγορη δικτυακή υποδομή, γιατί αν ο ρυθμός αιτήσεων επηρεαζόταν από τη χαμηλή ταχύτητα του δικτύου, δεν θα μετρούσαμε το ρυθμό αιτήσεων που εξυπηρετούσε το Web proxy, αλλά το ρυθμό αιτήσεων που μπορούσε να υποστηρίξει το δίκτυό μας. Για το λόγο αυτό, επιλέξαμε να τρέξουμε τα Web proxy υπό μέτρηση στο ίδιο μηχάνημα με το μετροπρόγραμμα Web Polygraph. Παράλληλα όμως έπρεπε να επιλεγεί μία μηχανή με αρκετή υπολογιστική ισχύ, ώστε το μετροπρόγραμμα να μην απορροφά σημαντικούς πόρους από το Web proxy. Πάντως, η συγκεκριμένη προϋπόθεση δεν είναι δύσκολο να υλοποιηθεί αφού όπως έχουμε αναφέρει το σημείο στο οποίο παρουσιάζονται προβλήματα απόδοσης στη λειτουργία των Web proxies είναι το υποσύστημα δίσκων.

### 6.1.3 ΠΑΡΑΜΕΤΡΟΙ ΠΕΙΡΑΜΑΤΩΝ

Για τη διεξαγωγή των πειραμάτων μας επιλέξαμε τις παρακάτω παραμέτρους του μετροπρογράμματος Web Polygraph:

◇ **Αριθμός Πελατών : 100.**
Ένα πλήθος από 100 πελάτες είναι μια φυσιολογική τιμή για ένα Web proxy. Είναι αρκετό πλήθος για να μπορεί το Web proxy να εξυπηρετήσει πολλές αιτήσεις ταυτόχρονα, ενώ παράλληλα δεν είναι τόσο υψηλό ώστε και μόνο ο αριθμός των TCP συνδέσεων να φορτώνει το σύστημα. Ο αριθμός των Εξυπηρετών του benchmark δε μας ενδιέφερε, αφού ακόμα κι ένας γρήγορος Εξυπηρέτης θα μπορούσε να απαντάει όλες τις αιτήσεις από τους 100 Πελάτες.

◇ **Ποσοστό Επιτυχημένων Αιτήσεων 50%.**
Χρησιμοποιήσαμε ποσοστό επιτυχημένων αιτήσεων που είναι ένα έγκυρο ποσοστό σε μετρήσεις πραγματικών Web proxies και είναι παράλληλα αρκετά υψηλό, ώστε να μετρήσουμε την απόδοση που έχει η μνήμη (η κεντρική μνήμη και ο δίσκος) όταν διαβάζουμε ή γράφουμε δεδομένα.

◇ **Κατανομή αιτήσεων πελατών : zipf(0.6).**
Η κατανομή αιτήσεων πελατών αφορά την κατανομή που χρησιμοποιεί το Web polygraph για να επιλέξει ποια αντικείμενα θα ξαναζητηθούν από τους πελάτες, έτσι ώστε να διατηρηθεί το ποσοστό επιτυχημένων αιτήσεων που έχουμε επιλέξει. Η κατανομή zipf(0.6) είναι η κατανομή Zipf με τιμή $\alpha = 0.6$ (βλέπε παράγραφο 3.1.1). Η τιμή αυτή του $\alpha$ συμφωνεί με μετρήσεις ερευνητών [11]. Η ίδια αυτή παράμετρος zipf(0.6) έχει χρησιμοποιηθεί και σε πειράματα με εμπορικά Web proxies από την ομάδα που ανέπτυξε το Web Polygraph [80].

◇ **Στατικά αντικείμενα και Ποσοστό Αποθηκεύσιμων Αντικειμένων 100%.**
Στα πειράματά μας χρησιμοποιήσαμε στατικά αντικείμενα (δηλαδή αντικείμενα που δεν αλλάζουν όσο προχωράει ο χρόνος ή πιο σωστά αλλάζουν πολύ αργά σε σχέση με το χρόνο), γιατί μας ενδιαφέρει να μετρήσουμε τις επιδόσεις που

μπορεί να έχει ένα Web proxy στο μικρό χρονικό διάστημα των μερικών ωρών που διαρκούν τα πειράματά μας. Επίσης όλα τα αντικείμενα των πειραμάτων είναι αποθηκεύσιμα (cacheable) έτσι ώστε να μπορούμε να ελέγξουμε το ποσοστό επιτυχημένων αιτήσεων που επιτυγχάνει κάθε Web proxy σε σχέση με το ποσοστό που προσφέρεται από το benchmark.

◇ **Κατανομή Μεγεθών Αντικειμένων exp(5KB).**
Το μέγεθος αυτό είναι πολύ σημαντικό γιατί καθορίζει πόσο μεγάλα θα είναι τα αντικείμενα που ζητάνε οι Πελάτες του Web proxy και άρα τη συνολική κίνηση δεδομένων που θα εξυπηρετεί το Web proxy που μετράμε. Η κατανομή των μεγεθών που επιλέξαμε ήταν η εκθετική κατανομή με μέσο μέγεθος 5 KBytes, που όπως έχει μετρηθεί είναι το μέσο μέγεθος των αρχείων του Παγκόσμιου Ιστού [58], [60]. Η εκθετική κατανομή έχει επίσης επιλεγεί και για πειράματα εμπορικών Web proxies από την ομάδα που ανέπτυξε το Web Polygraph [80].

◇ **Όριο Πειράματος η συμπλήρωση 1000000 αιτήσεων.**
Θέσαμε σαν όριο τη συμπλήρωση 1000000 αιτήσεων από το Web proxy, γιατί το θεωρούμε αρκετό για την καταγραφή της απόδοσης του. Στη διάρκεια αυτών των αιτήσεων θα έχουν μεταφερθεί στους Πελάτες περίπου 5 GBytes δεδομένων (αφού το μέσο μέγεθος αντικειμένου τέθηκε στα 5 KBytes) και δεδομένου ότι έχουμε ποσοστό επιτυχημένων αιτήσεων 50%, θα γραφούν στην ενδιάμεση μνήμη (cache) του Web proxy περίπου 500000 αντικείμενα που αντιστοιχούν σε 2 GBytes δεδομένων.

Στα πειράματά που έγιναν, ο ρυθμός εξυπηρέτησης αιτήσεων από το Web Polygraph μεταβάλλεται ανάλογα με την απόδοση του Web proxy υπό μέτρηση. Με λίγα λόγια όταν εξυπηρετηθεί μια αίτηση, κάθε πελάτης του μετροπρογράμματος ξαναστέλνει αμέσως την επόμενη.

### 6.1.4 ΥΠΟΛΟΓΙΣΤΙΚΟ ΣΥΣΤΗΜΑ

Τα πειράματά μας έγιναν σε μία μηχανή Sun Ultra Enterprise 450 η οποία διέθετε τα χαρακτηριστικά που περιγράφονται στον πίνακα 6.1.

| | |
|---|---|
| CPU | 2 x Sun UltraSparc II - 400 MHz |
| L2 Cache Memory | 4 MBytes |
| Main Memory | 2 GBytes |

**Πίνακας 6.1** Χαρακτηριστικά Μηχανής Πειραματικού Περιβάλλοντος.

Το λειτουργικό σύστημα που έτρεχε ήταν το Sun Solaris έκδοσης 2.7 και το σύστημα αρχείων UFS που το συνοδεύει. Ο δίσκος που χρησιμοποιήθηκε από τα Web proxies που μετρήσαμε για την αποθήκευση αντικειμένων ήταν ένας Sun 18G Disk Drive με τα χαρακτηριστικά που περιγράφονται στον πίνακα 6.2.

| | |
|---|---|
| Model | Sun 18G Disk Drive |
| Interface | SCA SCSI |
| Capacity | 18 GBytes |
| Rotations per minute | 10000 |
| Rotational latency (average) | 2.99 msec |
| Single track seek (read) | 0.8 msec |
| Single track seek (write) | 1.0 msec |
| Average seek (read) | 7.5 msec |
| Average seek (write) | 8.5 msec |
| Maximum seek (read) | 16 msec |
| Maximum seek (write) | 17 msec |
| Interface transfer rate (synchronous) | 10 to 40 MBytes/sec |
| Interface transfer rate (asynchronous) | 5 to 20 MBytes/sec |

**Πίνακας 6.2** Χαρακτηριστικά Δίσκου Πειραματικού Περιβάλλοντος.

Χρησιμοποιώντας το μετροπρόγραμμα lat_fs από το σύνολο μετροπρογραμμάτων HBench-OS, μετρήσαμε το χρόνο που χρειαζόμαστε για τη δημιουργία και τη διαγραφή αρχείων με τη σειρά που δημιουργήθηκαν στο δίσκο που χρησιμοποιήσαμε. Τα αποτελέσματα φαίνονται στον πίνακα 6.3.

### 6.1.5 ΡΥΘΜΙΣΕΙΣ ΤΟΥ SQUID

Η έκδοση του Squid που χρησιμοποιήθηκε ήταν η 2.2.STABLE4 και οι παράμετροι του ήταν οι ίδιοι που έχουν χρησιμοποιηθεί και σε άλλες μετρήσεις [80] και θεωρούνται οι βέλτιστες για τέτοιου είδους πειράματα. Πιο συγκεκριμένα, εκτός από τον ορισμό του TCP port στο οποίο θα έτρεχε το Squid και τον κατάλογο του δίσκου που θα χρησιμοποιούσε, κάναμε τις εξής ρυθμίσεις στο Squid :

- ♢ cache_mem 128 MB
- ♢ half_closed_clients off
- ♢ memory_pools off

| File Size (Bytes) | Number of Files | Χρόνος ανά Αρχείο (msec) |
|---|---|---|
| 4096 | 64 | 11972.8125 |
| 4096 | 128 | 12076.1484 |
| 4096 | 512 | 8764.2910 |
| 4096 | 1024 | 8714.0635 |
| 5120 | 64 | 6228.2969 |
| 5120 | 128 | 12029.9180 |
| 5120 | 512 | 10638.953 |
| 5120 | 1024 | 8502.1104 |
| 8192 | 64 | 12066.7500 |
| 8192 | 128 | 12867.4844 |
| 8192 | 512 | 12249.8887 |
| 8192 | 1024 | 11823.4102 |

**Πίνακας 6.3** Δημιουργία αρχείων στο δίσκο του Πειραματικού Περιβάλλοντος.

Στην ιεραρχία δύο επιπέδων καταλόγων που χρησιμοποιεί το Squid για την αποθήκευση αντικειμένων στο δίσκο, έμεινε η αρχική τιμή του, που ήταν 16 κατάλογοι για το πρώτο επίπεδο και 256 για το δεύτερο. Όλες οι υπόλοιπες ρυθμίσεις του Squid ήταν οι αρχικές της εγκατάστασής του, δηλαδή αυτές που είχαν διαμορφωθεί από την ομάδα που το αναπτύσσει [102].

### 6.1.6 ΡΥΘΜΙΣΕΙΣ ΤΟΥ FOXY

Στο FOXY χρησιμοποιήσαμε μόνο ένα (δευτερεύον) επίπεδο μνήμης (δηλαδή δεν χρησιμοποιήσαμε κύρια μνήμη για την αποθήκευση αντικειμένων αλλά μόνο το δίσκο). Βέβαια το λειτουργικό σύστημα χρησιμοποιούσε την κρυφή μνήμη του συστήματος αρχείων για τα δεδομένα που γράφαμε στο δίσκο, όπως έγινε και με το Squid. Το "υψηλό όριο πληρότητας" στα πειράματά μας ήταν 70% (που σημαίνει ότι καλούσαμε την πολιτική αντικατάστασης αντικειμένων όταν ο δίσκος γέμιζε κατά 70%) και το "χαμηλό όριο πληρότητας" ήταν 50% (δηλαδή σβήναμε αντικείμενα από το δίσκο μέχρι να είναι 50% γεμάτος). Τέλος, το μέγιστο όριο μεγέθους των αντικειμένων που θα αποθηκευτούν στην μνήμη του FOXY ήταν 64 KBytes.

## 6.2 ΠΕΡΙΓΡΑΦΗ ΠΕΙΡΑΜΑΤΩΝ

Στα πειράματα μας επιλέξαμε να συγκρίνουμε το Web proxy που υλοποιήσαμε (το FOXY) με το Squid που όπως έχουμε αναφέρει είναι το πιο εξελιγμένο Web proxy ανοιχτού κώδικα σήμερα.

Στα πειράματά μας δοκιμάσαμε και τα δύο Web proxies (FOXY και Squid) χρησιμοποιώντας το μετροπρόγραμμα Web Polygraph με τις παραμέτρους που αναφέραμε παραπάνω και για τρία διαφορετικά μεγέθη χώρου αποθήκευσης αντικειμένων στο δίσκο : 1 GByte, 2 GBytes, 4 GBytes. Με λίγα λόγια κάναμε τα εξής έξι πειράματα:

- ◇ FOXY με (1 GByte, 2 GBytes, 4 GBytes) Disk Cache.
- ◇ Squid με (1 GByte, 2 GBytes, 4 GBytes) Disk Cache.

Τα μεγέθη χώρου που δώσαμε είναι χαρακτηριστικά για να μετρήσουμε τη συμπεριφορά των Web proxies υπό διαφορετικές συνθήκες. Αν υπολογίσουμε ότι 1000000 αιτήσεις με 5 KBytes μέσο μέγεθος αντικειμένου μας δίνουν 5 GBytes δεδομένων και δεδομένου ότι έχουμε ποσοστό επιτυχημένων αιτήσεων 50%, θα έχουμε όγκο δεδομένων που θα γραφεί στην μνήμη του Web proxy περίπου 2.5 GBytes (προσθέτοντας και τους HTTP headers των αντικειμένων που είναι τουλάχιστον 100 bytes για κάθε αντικείμενο έχουμε περίπου 100 MBytes παραπάνω). Σκεπτόμενοι την κύρια μνήμη που έχει το υπολογιστικό σύστημα στο οποίο τρέξαμε τα πειράματα, είναι φανερό ότι δεν χωράει όλα τα δεδομένα που πρέπει να αποθηκεύσει το Web proxy, άρα θα πρέπει υποχρεωτικά να καταφύγουμε στο δίσκο. Τα πειράματά μας λοιπόν μελετάνε τη συμπεριφορά των Web proxies υπό τέτοιες συνθήκες.

Με τα μεγέθη 1, 2, και 4 GBytes χώρου στο δίσκο που δώσαμε μπορούμε να μελετήσουμε την συμπεριφορά του Web proxy σε τρεις διαφορετικές περιπτώσεις:

- ◇ **1 GByte Disk Cache**
  Στην περίπτωση αυτή, εφόσον γράφουμε περίπου 2.5 GBytes στο δίσκο, θα χρειαστεί να καλέσουμε 2-3 φορές την πολιτική αντικατάστασης αντικειμένων για να αντικαταστήσουμε αντικείμενα αφού ο χώρος δεν είναι αρκετός. Επίσης θα φανεί πόσο αποδοτική είναι η πολιτική αντικατάστασης αντικειμένων εφόσον δεν υπάρχει αρκετός χώρος για όλα τα αντικείμενα.

- ◇ **2 GBytes Disk Cache**
  Στην περίπτωση των 2 GBytes χώρου στο δίσκο, θα χρειαστεί επίσης να σβήσουμε από το δίσκο καλώντας την πολιτική αντικατάστασης αντικειμένων, αλλά αυτό θα γίνει 1 ή το πολύ δύο φορές. Επίσης σκεπτόμενοι την κύρια μνήμη που έχει το υπολογιστικό σύστημα στο οποίο τρέξαμε τα πειράματα, σε αυτή την περίπτωση δεν θα χωράνε όλα τα δεδομένα σε αυτήν άρα θα πρέπει να γράψουμε υποχρεωτικά στο δίσκο.

◊ **4 GBytes Disk Cache**

Τέλος όταν έχουμε 4 GBytes χώρου στο δίσκο, δεν θα χρειαστεί να σβήσουμε αντικείμενα από το δίσκο και έτσι λογικά θα έχουμε το βέλτιστο ποσοστό επιτυχημένων αιτήσεων. Παρ' όλα αυτά, επειδή θα έχουμε μεγαλύτερο χώρο να διαχειριστούμε, είναι αναμενόμενο η απόδοση να πέσει λόγων των περισσότερων αναγνώσεων από το δίσκο.

Αφού λοιπόν εκτελέσαμε τα παραπάνω πειράματα, δίνουμε τα αποτελέσματά μας στην επόμενη παράγραφο 6.3.

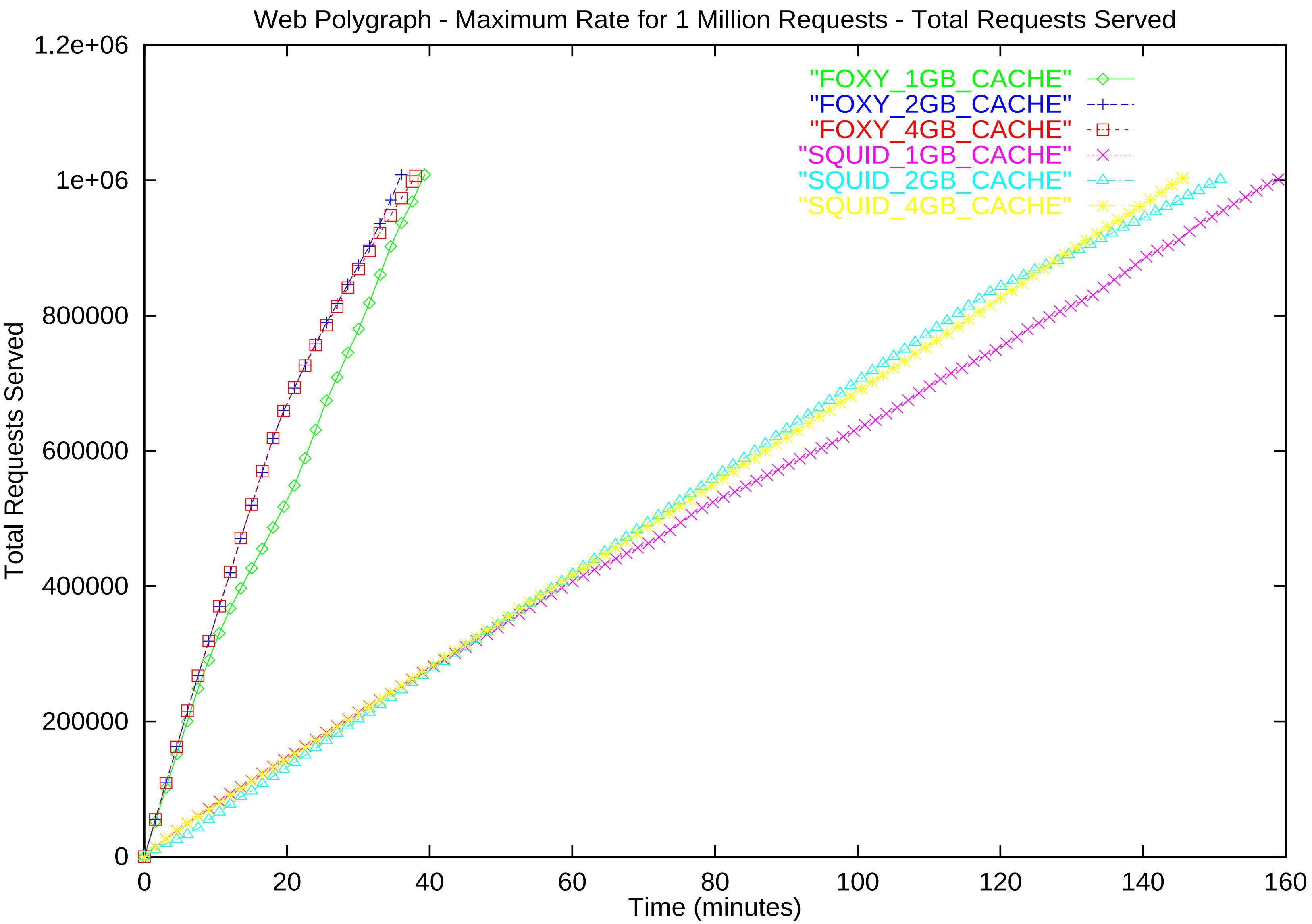
## 6.3 ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

Τα αποτελέσματα των πειραμάτων μας καθώς και μετρήσεις παραμέτρων του υπολογιστικού συστήματος (π.χ. πόση μνήμη ήταν ελεύθερη, πόση υπολογιστική ισχύς χρησιμοποιόταν κτλ.) κατά τη διάρκεια των πειραμάτων, φαίνονται στα σχήματα παρακάτω. Όπως θα δούμε οι παράμετροι του συστήματος θα μας βοηθήσουν πολύ στην ερμηνεία των αποτελεσμάτων μας.

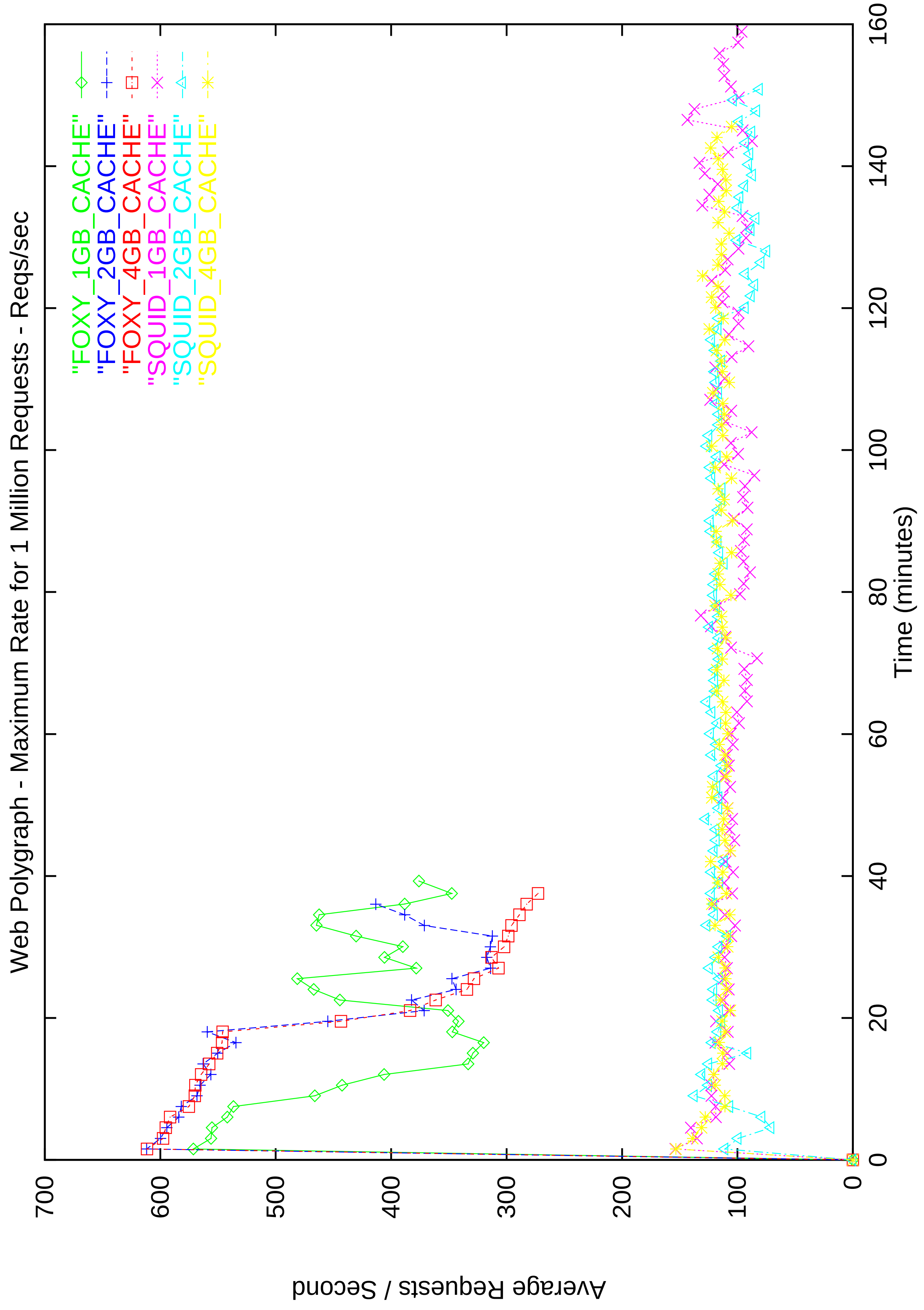
Ξεκινώντας από το σχήμα 6.2 παρατηρούμε την εξυπηρέτηση των 1000000 αιτήσεων των πειραμάτων από το Squid και το FOXY σε σχέση με τη χρονική στιγμή έναρξης κάθε πειράματος. Παρατηρούμε ότι το FOXY αποδίδει πολύ καλύτερα και καταφέρνει να τελειώσει το πείραμα σε χρόνο μικρότερο των 40 λεπτών για όλα τα μεγέθη χώρου στο δίσκο που χρησιμοποιήσαμε. Αντίθετα, το Squid χρειάζεται περίπου 150-160 λεπτά σε όλες τις περιπτώσεις.

Στο σχήμα 6.3 παρατηρούμε το ρυθμό εξυπηρέτησης αιτήσεων στη μέγιστη χρονική διάρκεια των πειραμάτων. Ο άξονας x φτάνει μέχρι τα 160 λεπτά επειδή η χρονική διάρκεια που χρειάστηκε το Squid για να ολοκληρώσει το πείραμα ήταν μεγαλύτερη από το FOXY. Όπως φαίνεται ο ρυθμός εξυπηρέτησης αιτήσεων για το FOXY είναι αρκετά υψηλότερος από το Squid, αλλά παρουσιάζει και μεγαλύτερες διακυμάνσεις. Αυτές οφείλονται όπως θα δούμε στα επόμενα σχήματα στην απόδοση της μνήμης και του δίσκου.

Στο σχήμα 6.5 φαίνεται η διακύμανση του ποσοστού επιτυχημένων αιτήσεων κατά τη διάρκεια των πειραμάτων. Όπως παρατηρούμε σε αυτό το σχήμα, στην εξέλιξη του πειράματος το Squid έχει χαμηλότερο ποσοστό επιτυχημένων αιτήσεων από ότι το FOXY. Επίσης παρατηρούμε ότι για μεγαλύτερο χώρο στο δίσκο, το ποσοστό επιτυχημένων αιτήσεων του FOXY ανεβαίνει, ενώ για χώρο 4 GBytes όπου δεν σβήνονται αντικείμενα από το δίσκο, το ποσοστό επιτυχημένων αιτήσεων διατηρείται στο 50%, όπως δηλαδή έχει τεθεί σαν παράμετρος στο Web Polygraph. Αντίθετα για 1 GByte χώρο στο δίσκο, το ποσοστό επιτυχημένων αιτήσεων είναι αρκετά χαμηλό,




**Σχήμα 6.2** Εξυπηρέτηση αιτήσεων από το Squid και το FOXY σε σχέση με το χρόνο.




**Σχήμα 6.3** Μεταβολή του ρυθμού επιτυχημένων αιτήσεων στη διάρκεια των πειραμάτων.

διότι ο διαθέσιμος χώρος δεν φτάνει για να μείνει στο 50%. Το Squid παρουσιάζει αρκετά χαμηλότερο ποσοστό επιτυχημένων αιτήσεων, ένα μεγάλο μέρος από το οποίο οφείλεται στις λανθασμένες αποκρίσεις ((βλέπε σχήμα 6.6 παρακάτω).

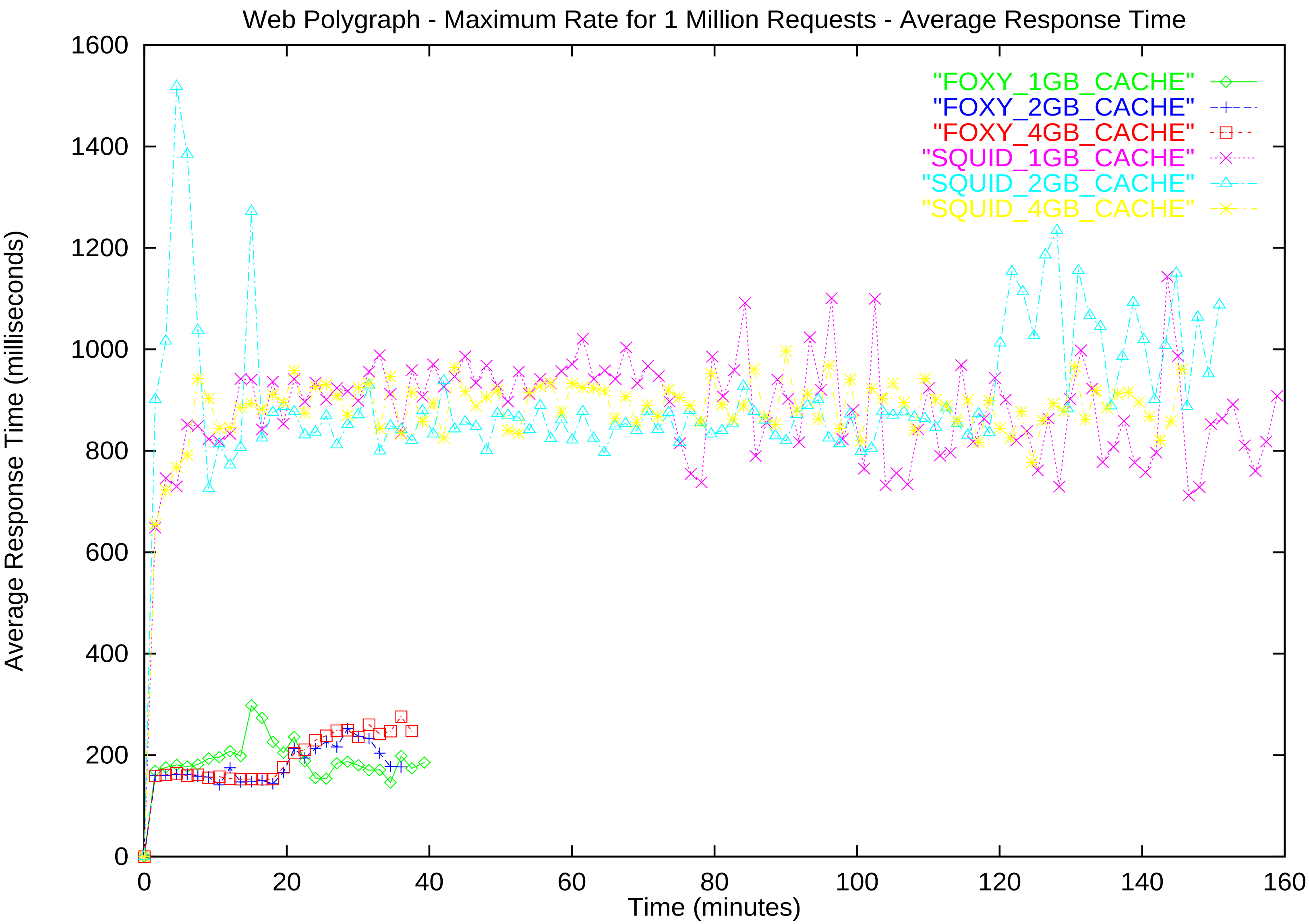
Στο σχήμα 6.4 φαίνεται η διακύμανση του μέσου χρόνου εξυπηρέτησης ανά αίτηση κατά τη διάρκεια των πειραμάτων. Όπως παρατηρούμε, το FOXY έχει μέσο χρόνο εξυπηρέτησης ανά αίτηση που διατηρείται γύρω στα 200 milliseconds, ενώ αντίθετα ο μέσος χρόνος εξυπηρέτησης ανά αίτηση είναι στο διάστημα 800 ως 1000 milliseconds.

Στο σχήμα 6.6 παρατηρούμε το συνολικό αριθμό λαθών κατά τη διάρκεια των πειραμάτων (δηλαδή των αιτήσεων κατά τη διάρκεια των οποίων παρουσιάστηκαν λάθη). Όπως βλέπουμε από τα 60 λεπτά και μετά το Squid με 1 GByte χώρο στο δίσκο παρουσιάζει μεγάλο αριθμό λανθασμένων αιτήσεων, το οποίο αποτελεί και ένα λόγο για τη δραστική μείωση του ποσοστού επιτυχημένων αιτήσεων (βλέπε σχήμα 6.5).

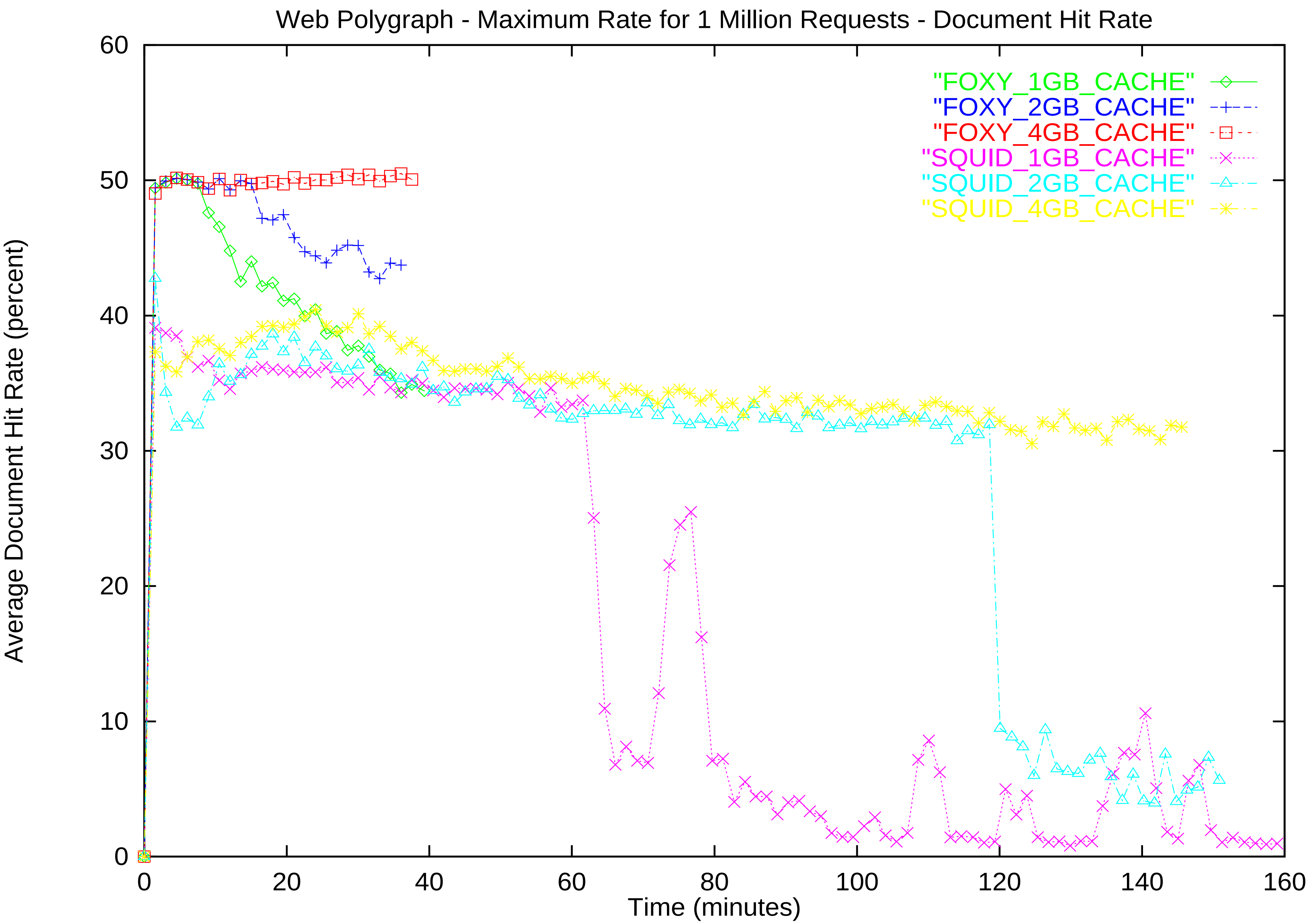
Τα επόμενα πέντε σχήματα περιγράφουν με ακρίβεια τι συμβαίνει κατά τη διάρκεια των πειραμάτων του FOXY και γιατί υπάρχουν οι διακυμάνσεις στο μέσο ρυθμό εξυπηρέτησης αιτήσεων. Όπως βλέπουμε στο σχήμα 6.7 περίπου στα 10 λεπτά, όπου πέφτει ο μέσος ρυθμός εξυπηρέτησης για 1 GByte disk cache ανεβαίνει ο χρόνος που η CPU δεν χρησιμοποιείται (idle time). Αυτό γίνεται γιατί όπως βλέπουμε στο σχήμα 6.10 αυξάνονται οι αναγνώσεις και οι εγγραφές στο δίσκο.

Αντίστοιχα στην περίπτωση των 2 και 4 GBytes disk cache, στα 20 περίπου λεπτά αυξάνεται ο αριθμός των αναγνώσεων και μειώνεται ο αριθμός των εγγραφών (σχήμα 6.10), ενώ παράλληλα μειώνεται η χρήση της CPU (σχήματα 6.8 και 6.9) διότι το FOXY περιμένει τις αναγνώσεις από το δίσκο. Η κίνηση λοιπόν της κεφαλής του δίσκου για τις αναγνώσεις καθυστερεί την εξυπηρέτηση των αιτήσεων και μειώνει το μέσο ρυθμό εξυπηρέτησής τους.

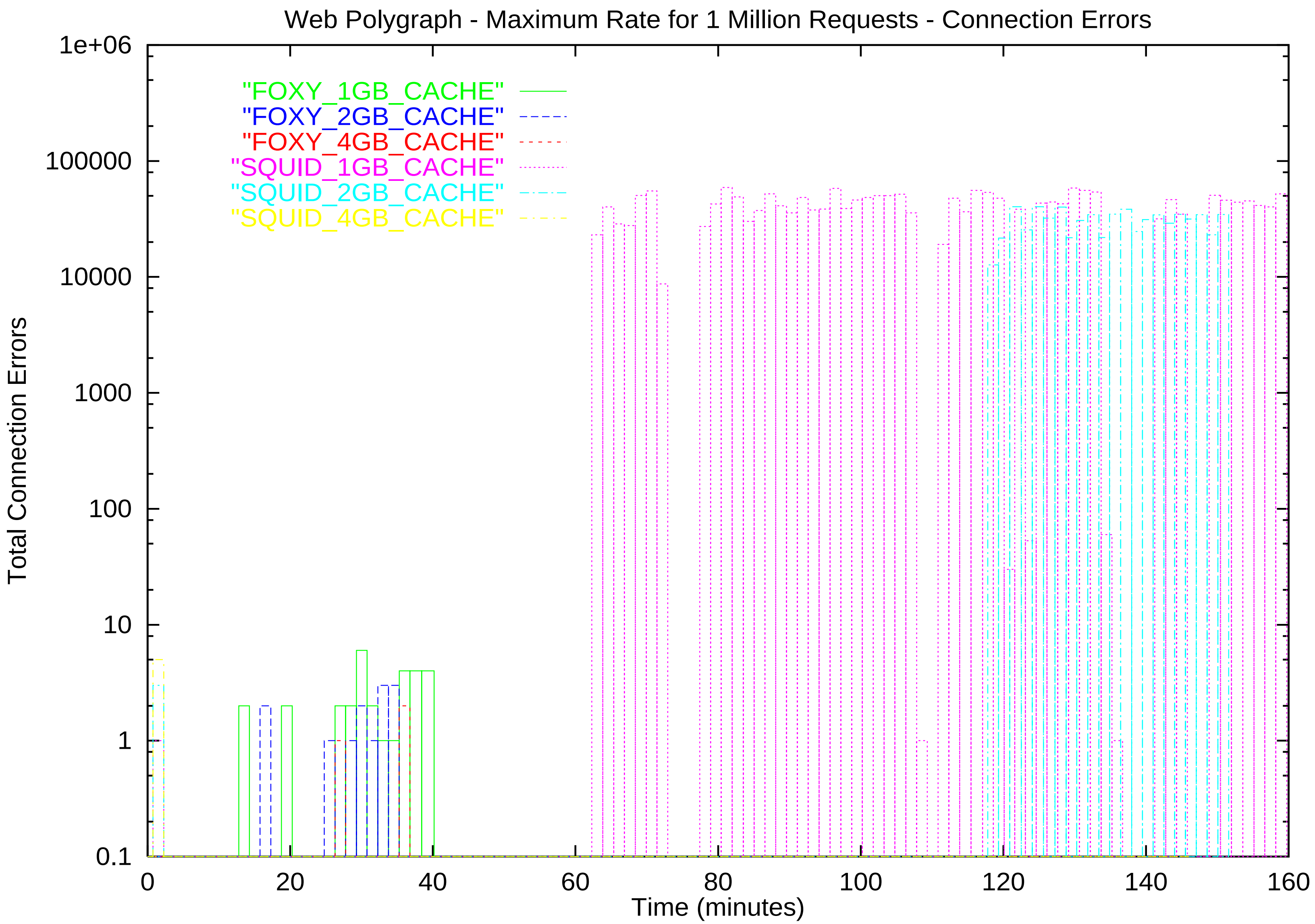
Τέλος βλέποντας το σχήμα 6.11 μπορούμε να αιτιολογήσουμε τον αρχικό υψηλό μέσο ρυθμό εξυπηρέτησης αιτήσεων παρατηρώντας ότι αρχικά η μνήμη της μηχανής είναι άδεια και τα δεδομένα προς το δίσκο γράφονται εκεί, ενώ στη συνέχεια γεμίζει και η πρόσβαση στο δίσκο μειώνει τις επιδόσεις.



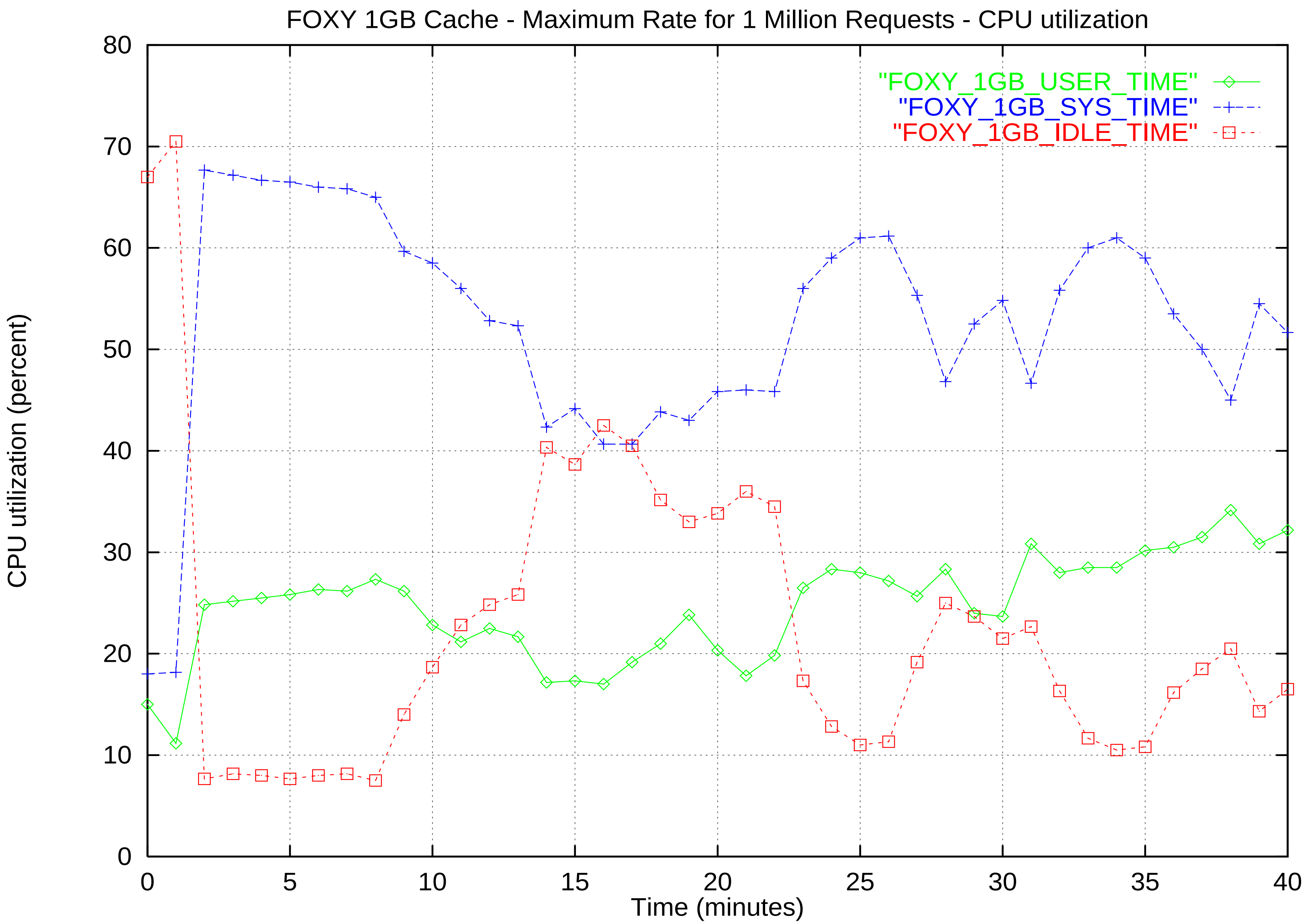

**Σχήμα 6.4** Μεταβολή του μέσου χρόνου εξυπηρέτησης ανά αίτηση στη διάρκεια των πειραμάτων.



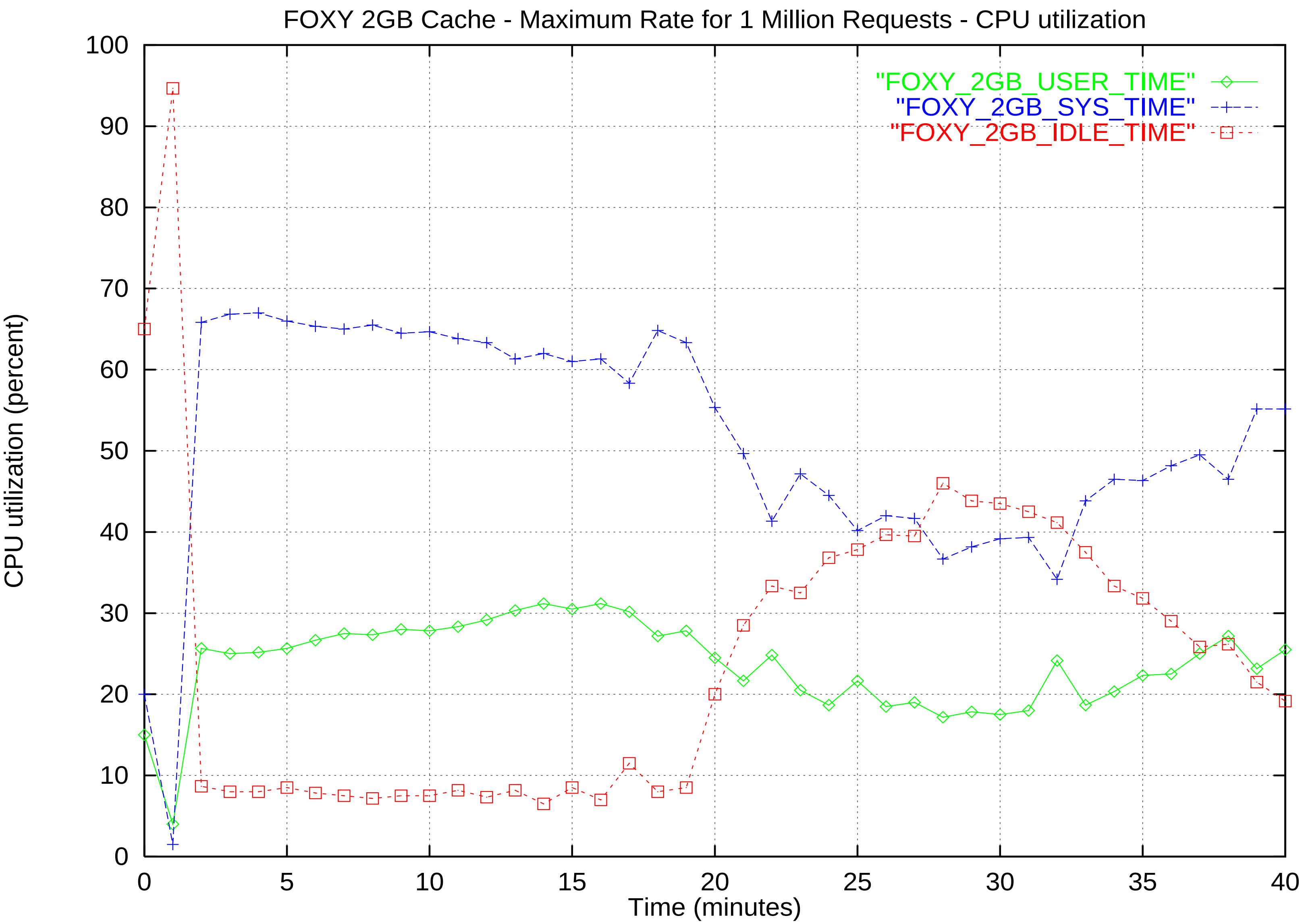

**Σχήμα 6.5** Μέτρηση του ποσοστού επιτυχημένων αιτήσεων στη διάρκεια των πειραμάτων.



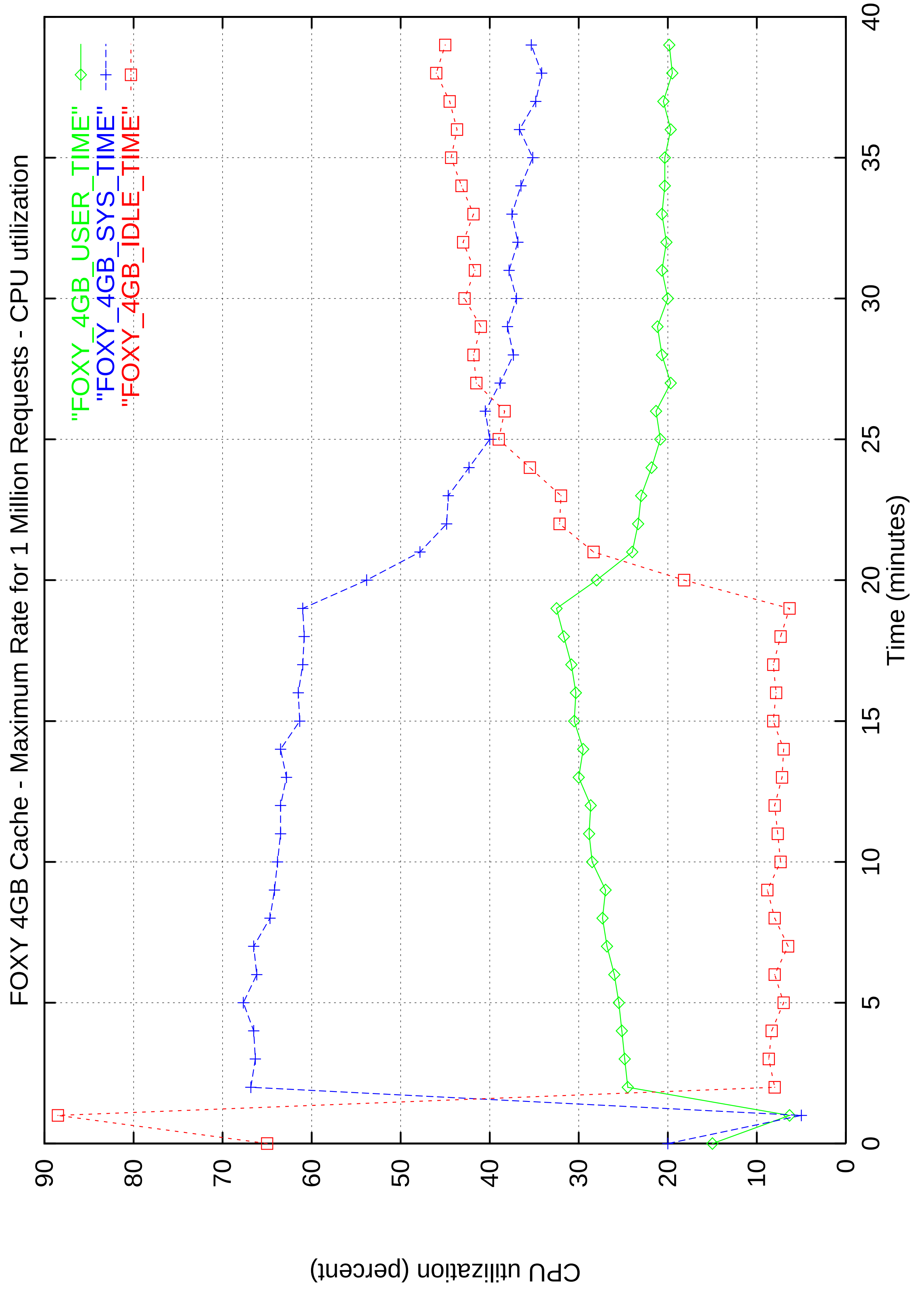

**Σχήμα 6.6** Μέτρηση του αριθμού των λανθασμένων αιτήσεων στην διάρκεια των πειραμάτων.



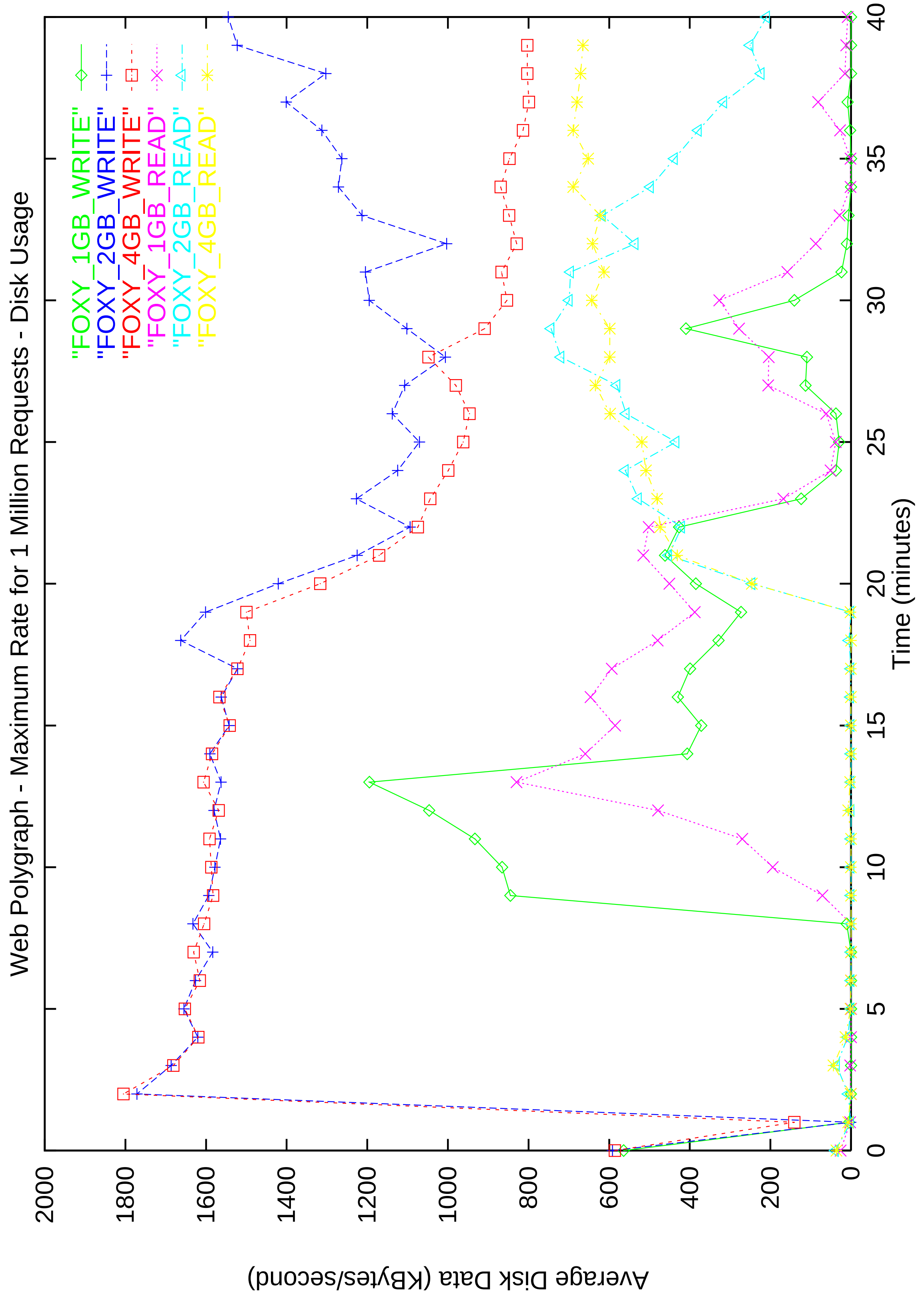

**Σχήμα 6.7** Μέτρηση της χρήσης της CPU στην διάρκεια των πειραμάτων του FOXY με 1 GByte disk cache.



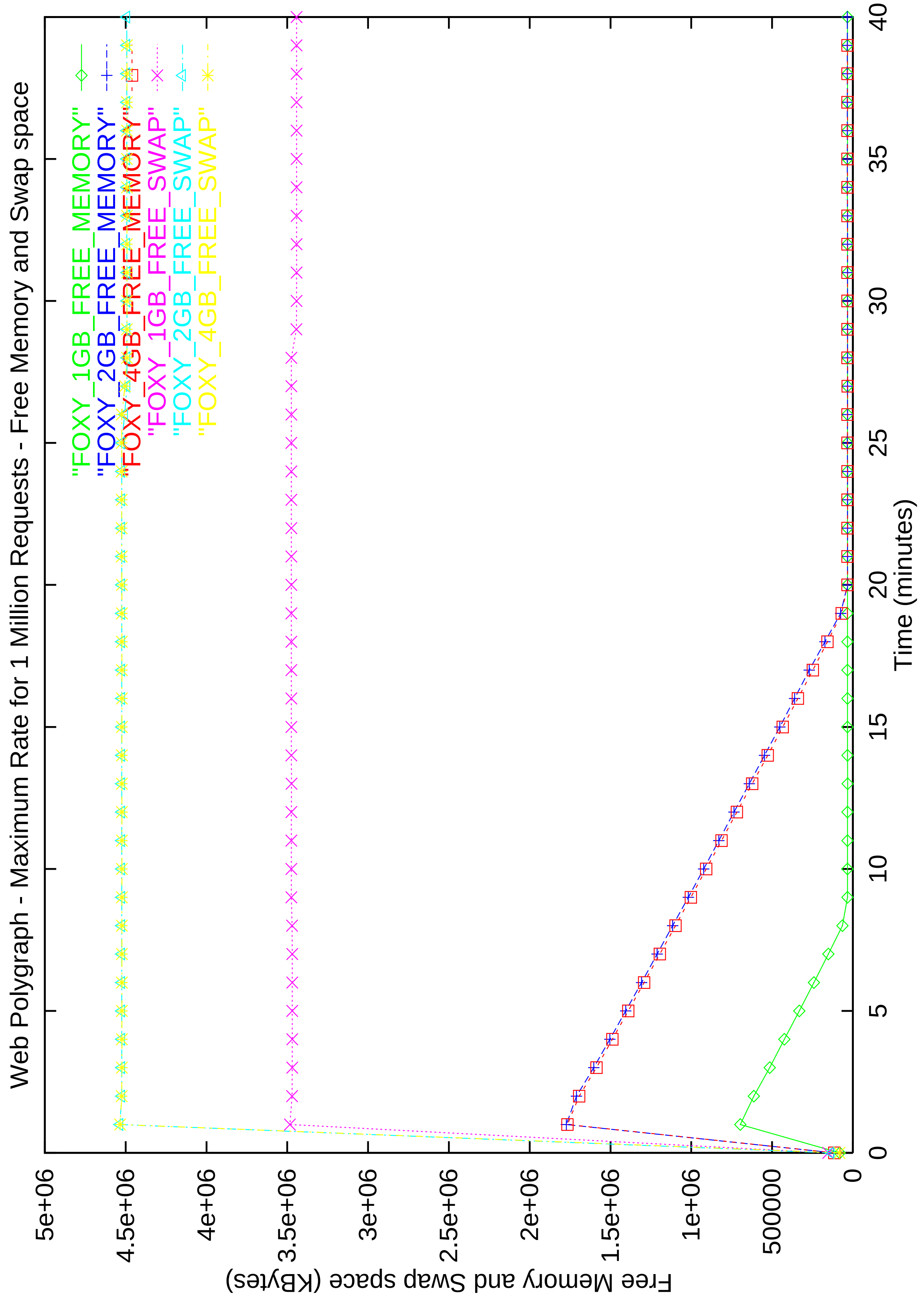

**Σχήμα 6.8** Μέτρηση της χρήσης της CPU στην διάρκεια των πειραμάτων του FOXY με 2 GBytes disk cache.




**Σχήμα 6.9** Μέτρηση της χρήσης της CPU στην διάρκεια των πειραμάτων του FOXY με 4 GBytes disk cache.




**Σχήμα 6.10** Μέτρηση της χρήσης του δίσκου κατά την διάρκεια των πειραμάτων του FOXY.




**Σχήμα 6.11** Μέτρηση της ελεύθερης κεντρικής μνήμης κατά την διάρκεια των πειραμάτων του FOXY.

# Επίλογος

Life would be so much easier if we could just look at the source code.
—Anon.

Στην παρούσα γραπτή εργασία ασχοληθήκαμε με το ερευνητικό θέμα της απόδοσης των Web proxies και εστιάσαμε ειδικότερα στο σημείο της αποθήκευσης των αντικειμένων WWW στους δίσκους που αποτελεί το μεγαλύτερο παράγοντα μείωσης των επιδόσεων. Η συνεισφορά της παρούσας μεταπτυχιακής εργασίας αφορά:

◇ Την περιληπτική παρουσίαση μεθόδων για τη βελτίωση του υποσυστήματος δίσκων ενός Web proxy. Μέσα από μελέτη της χρήσης των δίσκων και των συστημάτων αρχείων παρουσιάζουμε αποτελεσματικούς αλγορίθμους για την αύξηση της ταχύτητας εγγραφής και ανάγνωσης των δεδομένων από τους δίσκους. Οι αλγόριθμοι αυτοί αναπτύχθηκαν από μια ομάδα ανθρώπων στην οποία συμμετείχε και ο συγγραφέας και παρουσιάστηκαν εκτενώς στο [62].

◇ Την ανάπτυξη ενός πρότυπου Web proxy (το οποίο ονομάσαμε FOXY) σχεδιασμένου ώστε να προσφέρει υψηλές επιδόσεις υλοποιώντας τους παραπάνω μηχανισμούς για τη βελτίωση της ταχύτητας εγγραφής και ανάγνωσης των δεδομένων από τους δίσκους. Το Web proxy που αναπτύξαμε, χρησιμοποιεί διαδεδομένο υλικό (hardware) και γενικού σκοπού λειτουργικά συστήματα.

◇ Τον πειραματισμό και τη σύγκριση της υλοποίησής μας με το Squid [102], που αποτελεί το πιο δημοφιλές και προηγμένο Web proxy ανοιχτού κώδικα που τρέχει σε γενικού σκοπού λειτουργικά συστήματα σήμερα. Όπως αποδεικνύεται από τα πειράματά μας, η υλοποίησή μας έχει απόδοση τουλάχιστον τρεις φορές υψηλότερη από το Squid.

Αν και πιστεύουμε ότι η συνεισφορά μας αυτή προάγει την τεχνολογία των Web proxies και τις ευεργετικές συνέπειες της στην καθημερινή χρήση του Παγκόσμιου Ιστού, θεωρούμε ότι υπάρχουν ακόμα σημαντικά περιθώρια βελτίωσης των αλγορίθμων διαχείρισης του δίσκου, αλλά και της υλοποίησης του FOXY. Αυτές οι βελτιώσεις μπορούν να αποτελέσουν μέρος μιας μελλοντικής ερευνητικής εργασίας. Μερικές κατευθύνσεις που θα μπορούσαμε να κινηθούμε μελλοντικά είναι:

◇ Η βελτίωση της υλοποίησης του FOXY με μελετημένη χρήση threads ώστε να έχουμε παράλληλη χρήση πολλών επεξεργαστών. Αντίθετα, στην παρούσα υλοποίηση που χρησιμοποιούμε μόνο μία διεργασία χωρίς threads μπορούμε να εκμεταλλευτούμε την υπολογιστική ισχύ μόνο ενός επεξεργαστή. Όπως είναι αντιληπτό, η αποτελεσματική χρήση πολλών επεξεργαστών σε μηχανές που παρέχουν τέτοιες δυνατότητες θα βελτιώσει τις επιδόσεις του FOXY. Πάντως η αποτελεσματική χρήση threads πρέπει να γίνει μετά από προσεκτική μελέτη και αποφεύγοντας απλουστεύσεις όπως π.χ. τη δημιουργία νέου thread για κάθε αίτηση, κάτι που θα δημιουργούσε περιττό υπολογιστικό φόρτο.

◇ Η σχεδίαση αλγορίθμων για περισσότερους του ενός δίσκους παράλληλα που να συνδυάζει αποτελεσματικά τις πολιτικές STREAM-PACKETIZER και LAZY-READS με σκοπό ακόμα υψηλότερες επιδόσεις. Επίσης θα ήταν επιθυμητή η υλοποίηση τέτοιων αλγορίθμων πολλαπλών δίσκων στο FOXY και η πειραματική αποτίμησή τους.

◇ Μία παράμετρος που επηρεάζει σημαντικά τις επιδόσεις όπως είδαμε και από τα πειράματά μας στην παράγραφο 6.3 (σχήμα 6.10) είναι οι αναγνώσεις αντικειμένων από το δίσκο. Εφόσον λοιπόν μία βελτίωση στην περιοχή αυτή θα προσέφερε σημαντικά καλύτερες επιδόσεις, είναι θεμιτό να εξερευνηθεί βαθύτερα και να προταθούν νέοι καλύτεροι αλγόριθμοι που θα βελτιώνουν τις αναγνώσεις των αντικειμένων από το δίσκο.

◇ Τέλος, θα μπορούσαμε να μελετήσουμε αποτελεσματικούς τρόπους για τη μετατροπή του FOXY σε ένα “έξυπνο Web Proxy”, δηλαδή ένα Web Proxy που δεν θα λειτουργεί μόνο παθητικά εξυπηρετώντας τις αιτήσεις των χρηστών, αλλά έχοντας κάποια εσωτερική “νοημοσύνη” θα παίρνει και πρωτοβουλίες με στόχο την καλύτερη ποιότητα υπηρεσίας προς τους χρήστες, αλλά και την καλύτερη εξοικονόμηση Διαδικτυακών πόρων. Σε αυτή την περιοχή θα μπορούσαμε να μελετήσουμε ιδέες prefetching αντικειμένων WWW, εναλλαγή εκδόσεων πρωτοκόλλου HTTP (1.0 σε 1.1), συμπίεση δεδομένων είτε στην cache είτε πάνω από το δίκτυο, μετατροπή της ψηφιακής μορφής των στατικών εικόνων σε άλλη μικρότερου μεγέθους (π.χ. gif ή tif σε jpeg), WWW content filtering, dynamic URL redirection κτλ.

# Βιβλιογραφία


[1] Ghaleb Abdulla, Edward A. Fox, and Marc Abrams. Shared User Behavior on the World Wide Web. In *Proceedings of WebNet97, Toronto*, October 1997.

[2] Ghaleb Abdulla, Binzhang Liu, and Edward A. Fox. Searching the World-Wide Web: Implications from Studying Different User Behavior. In *Proceedings of WebNet98, Orlando*, November 1998.

[3] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, and Edward A. Fox. Caching Proxies: Limitations and Potentials. In *Proceedings of 4th International World-Wide Web Conference*, pages 119–133, December 1995.

[4] Charu C. Aggarwal and Philip S. Yu. On disk caching of web objects in proxy servers. In Forouzan Golshani and Kia Makki, editors, *Proceedings of the 6th International Conference on Information and Knowledge Management (CIKM-97)*, pages 238–245, New York, November 10–14 1997. ACM Press.

[5] Alex Rousskov. Web Polygraph : Proxy Performance Benchmark. URL: http://polygraph.ircache.net/.

[6] Jussara Almeida and Pei Cao. Measuring proxy performance with the Wisconsin Proxy Benchmark. *Computer Networks And ISDN Systems*, 30(22-23):2179–2192, November 1998. URL : http://www.elsevier.nl/cas/tree/store/comnet/sub/1998/30/22-23/2053.pdf.

[7] Apache Software Foundation. The Apache Web Server. URL: http://www.apache.org/.

[8] M. Arlitt, R. Friedrich, and T. Jin. Performance Evaluation of Web Proxy Cache Replacement Policies. *Lecture Notes in Computer Science*, 1469:193–204, 1998.

[9] Martin Arlitt, Ludmilla Cherkasova, John Dilley, Rich Friedrich, and Tai Jin. Evaluating Content Management Techniques for Web Proxy Caches. In *Proceedings of the Workshop on Internet Server Performance (WISP99)*, May 1999. URL : http://www.cc.gatech.edu/fac/Ellen.Zegura/wisp99/papers/arlitt.ps.

[10] D. J. Barnes and N. G. Smith. An Analysis of World-Wide Web Proxy Cache Performance and its Application to the Modelling and Simulation of Network Traffic. In *Proceedings of the 4th International Conference on Telecommunication Systems Modeling and Analysis*, page 9, March 1996. URL : http://www.cs.ukc.ac.uk/pubs/1996/184.

[11] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proceedings of the INFOCOM '99 conference*, March 1999. URL : http://www.cs.wisc.edu/ cao/papers/zipf-like.ps.gz.

[12] A. Brown and Margo Seltzer. Operating System Benchmarking in the Wake of Lmbench: A Case Study of the Performance of NetBSD on the Intel x86 Architecture. In *1997 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 214–224, Seattle, WA, June 1997. URL : http://www.eecs.harvard.edu/vino/perf/hbench/.

[13] CacheFlow Inc. Url: http://www.cacheflow.com/, 1998.

[14] Robert Cailliau. A Short History of the Web, November 1995. URL: http://www.inria.fr/Actualites/Cailliau-fra.html.

[15] Pei Cao and Sandy Irani. Cost-Aware WWW Proxy Caching Algorithms. In USENIX, editor, *USENIX Symposium on Internet Technologies and Systems Proceedings, Monterey, California, December 8–11, 1997*, pages 193–206, Berkeley, CA, USA, 1997. USENIX. URL : http://www.usenix.org/publications/library/proceedings/usits97/cao.html.

[16] Pei Cao, Jin Zhang, and Kevin Beach. Active Cache: Caching Dynamic Contents on the Web. In *Proceedings of the 1998 Middleware conference*, September 1998. URL : http://www.cs.wisc.edu/ cao/papers/active-cache.html.

[17] Internet Research Group Internet Caching Resource Center. Caching 101 - An overview of Web Caching, 1999. URL: http://www.caching.com/caching101.htm.

[18] Internet Research Group Internet Caching Resource Center. Caching Success Stories : Caching with CacheFlow + SkyCaching = Manageable Bandwidth Costs , September 1999. URL: http://www.caching.com/casestudies.htm.

[19] Internet Research Group Internet Caching Resource Center. The Internet Caching Bake-Off: Lose a Battle, Win the War, 2000. URL: http://www.caching.com/internet_caching_bakeoff.htm.

[20] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, June 1994.

[21] P. M. Chen and D. A. Patterson. Maximizing performance in a striped disk array. In *Proc. 17th Annual Int'l Symp. on Computer Architecture, ACM SIGARCH Computer Architecture News*, page 322, June 1990. Published as Proc. 17th Annual Int'l Symp. on Computer Architecture, ACM SIGARCH Computer Architecture News, volume 18, number 2.

[22] Peter Christy. How Do Caching and Content Delivery Really Differ ?, December 1999. URL: http://www.caching.com/cachingvcontent.htm.

[23] Inc. Cisco Systems. Cisco's Cache Engine. URL: http://www.cisco.com/warp/public/cc/cisco/mkt/scale/cache/index.shtml.

[24] IBM Corporation. Ibm web cache. URL: http://www.ibm.com/.

[25] Inktomi Corporation. Inktomi WebMap. URL: http://www.inktomi.com/webmap/.

[26] Paul Gauthier (Inktomi Corporation), Josh Cohen (Microsoft Corporation), Inc.) Martin Dunsmuir (RealNetworks, and Inc.) Charles Perkins (Sun Microsystems. Web Proxy Auto-Discovery Protocol (Internet Draft), 1999. URL: http://www.ietf.org/internet-drafts/draft-ietf-wrec-wpad-01.txt.

[27] Michael Donald Dahlin. Serverless network file systems. Technical Report CSD-96-900, University of California, Berkeley, April 1996.

[28] Peter B. Danzig, Richard S. Hall, and Michael F. Schwartz. A Case for Caching File Objects inside Internetworks. In *Proceedings of the ACM SIGCOMM '93 Conference*, San Francisco, CA., September 1993. ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/FTP.Caching-PS/.

[29] John Dilley, Martin Arlitt, and Stephane Perret. Enhancement and Validation of the Squid Cache Replacement Policy. In *Proceedings of the 4th International Web Caching Workshop*, April 1999. URL : http://www.ircache.net/Cache/Workshop99/Papers/dilley-0.ps.gz.

[30] John Douceur and William Bolosky. A large-scale study of file-system contents. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computing Systems (SIGMETRICS-99)*, volume 27,1 of *SIGMETRICS Performance Evaluation Review*, pages 59–70, New York, May 1999. ACM Press.

[31] Fred Douglis and Thomas Ball. Tracking and Viewing Changes on the Web. In USENIX Association, editor, *Proceedings of the USENIX 1996 annual technical conference: January 22–26, 1996, San Diego, California, USA*, USENIX Conference Proceedings 1996, pages 165–176, Berkeley, CA, USA, January 1996. USENIX.

[32] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey Mogul. Rate of Change and other Metrics: a Live Study of the World Wide Web. In USENIX, editor, *USENIX Symposium on Internet Technologies and Systems Proceedings, Monterey, California, December 8–11, 1997*, pages 147–158, Berkeley, CA, USA, 1997. USENIX. URL : http://www.usenix.org/publications/library/proceedings/usits97/douglis_rate.html.

[33] Fred Douglis, Antonio Haro, and Michael Rabinovich. HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching. In USENIX, editor, *USENIX Symposium on Internet Technologies and Systems Proceedings, Monterey, California, December 8–11, 1997*, pages 83–94, Berkeley, CA, USA, 1997. USENIX. URL : http://www.usenix.org/publications/library/proceedings/usits97/douglis_hpp.html.

[34] Bradley M. Duska, David Marwood, and Michael J. Feeley. The Measured Access Characteristics of World-Wide-Web Client Proxy Caches. Technical Report TR-97-16, Department of Computer Science, University of British Columbia, October 1997. URL : ftp://ftp.cs.ubc.ca/pub/local/techreports/1997/TR-97-16.ps.gz.

[35] Inc. F5 Networks. Building The Right Infrastructure for E-Commerce, 1999. URL: http://www.f5.com/solutions/whitepapers/ecommerce.html.

[36] Li Fan, Pei Cao, Wei Lin, and Quinn Jacobson. Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computing Systems (SIGMETRICS-99)*, volume 27,1 of *SIGMETRICS Performance Evaluation Review*, pages 178–187, New York, May 1–4 1999. ACM Press.

[37] Shahrooz Feizabadi. History of the World Wide Web. URL: http://ei.cs.vt.edu/ wwwbtb/book/chap1/index.html.

[38] Anja Feldmann, Ramon Caceres, Fred Douglis, Gideon Glass, and Michael Rabinovich. Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments. In *Proceedings of the INFOCOM '99 conference*, March 1999. URL : http://www.research.att.com/ anja/feldmann/papers/infocom99_proxim.ps.gz.

[39] The National Center for Supercomputing Applications : University of Illinois at Urbana-Champaign. NCSA Web Server Activity : Total Connections, June 1997. URL: http://www.ncsa.uiuc.edu/SDG/Presentations/Stats/WebServer.html.

[40] The National Center for Supercomputing Applications : University of Illinois at Urbana-Champaign. NSFnet Backbone Byte Traffic, June 1997. URL: http://www.ncsa.uiuc.edu/SDG/Presentations/Stats/NSFnetStats.html.

[41] Syam Gadde, Jeff Chase, and Michael Rabinovich. A Taste of Crispy Squid. In *Proceedings of the Workshop on Internet Server Performance (WISP'98)*, June 1998. URL : http://www.cs.duke.edu/ari/cisi/crisp/crisp-wisp.ps.gz.

[42] Michael A. Goulde. Network Caching Guide : Optimizing Web Content Delivery, March 1999. URL: http://www.inktomi.com/products/network/traffic/tech/CachingGuide.pdf, Prepared for Inktomi Corporation.

[43] CacheFlow Inc. High-Performance Web Caching White Paper, 1998. URL: http://www.cacheflow.com/technology/wp/hp_cache.html.

[44] CacheFlow Inc. Active Web Caching Technology, 1999. URL: http://www.cacheflow.com/technology/wp/activecaching.html.

[45] Inc. Infolibria. Infolibria Dynacache. URL: http://www.infolibria.com/products/dynacache_overview.html.

[46] Inktomi and NEC Research Institute. Web Surpasses One Billion Documents, January 2000. URL: http://www.inktomi.com/new/press/billion.html.

[47] Inc. Inktomi. Inktomi Traffic Server. URL: http://www.inktomi.com/products/network/traffic/.

[48] Shudong Jin and Azer Bestavros. Popularity-Aware GreedyDual-Size Web Proxy Caching Algorithms. Technical Report 1999-009, Boston University, August 1999. URL : http://www.cs.bu.edu/techreports/99-009-gdsp-web-proxy-caching.ps.Z.

[49] Jussara Almeida and Pei Cao. Wisconsin Proxy Benchmark 1.0. URL: http://www.cs.wisc.edu/ cao/wpb1.0.html.

[50] Eric Dean Katz, Michelle Butler, and Robert McGrath. A scalable HTTP server: The NCSA prototype. *Computer Networks and ISDN Systems*, 27(2):155–164, November 1994. URL: http://www.elsevier.com/cgi-bin/cas/tree/store/comnet/cas_sub/browse/browse.cgi?year=1994&;volume=27&issue=2&aid=1384.

[51] Ken Starr. The Starr Report. URL: http://www.cnn.com/starr.report/.

[52] Balachander Krishnamurthy and Craig Wills. Proxy Cache Coherency and Replacement – Towards a More Complete Picture. In *Proceedings of the ICDCS conference*, June 1999. URL : http://www.research.att.com/ bala/papers/ccrcp.ps.gz.

[53] Thomas T. Kwan, Robert E. McGrath, and Daniel A. Reed. NCSA's World Wide Web Server: Design and Performance. *Computer*, 28(11):68–74, November 1995.

[54] Thomas T. Kwan, Robert E. McGrath, and Daniel A. Reed. User access patterns to NCSA's World Wide Web server. Technical Report UIUCDCS-R-95-1934, University of Illinois at Urbana-Champaign, February 1995.

[55] Ron Lee and Gary Tomlinson. Workload Characteristics for a Very High-Capacity Proxy Cache Design.

[56] Ari Luotonen and Kevin Altis. World-Wide Web Proxies. *Computer Networks and ISDN Systems*, 27(2):147–154, November 1994. http://www.elsevier.com/cgi-bin/cas/tree/store/comnet/cas_sub/browse/-browse.cgi?year=1994&;volume=27&issue=2&aid=1380.

[57] Ari Luotonen, Henrik Frystyk Nielsen, and Tim Berners-Lee. CERN httpd : Version 3.0A, July 1996. URL: http://www.w3.org/Daemon/.

[58] Carlos Maltzahn, Kathy Richardson, and Dirk Grunwald. Reducing the Disk I/O of Web Proxy Server Caches. In *Proceedings if the USENIX Annual Technical Conference*, June 1999.

[59] Carlos Maltzahn, Kathy Richardson, Dirk Grunwald, and James Martin. A Feasibility Study of Bandwidth Smoothing on the World-Wide Web Using Machine Learning. Technical Report CU-CS-879-99, Dept. of Computer Science, University of Colorado at Boulder, January 1999. URL : http://www.cs.colorado.edu/ carlosm/CU-CS-879-99.ps.gz.

[60] Carlos Maltzahn, Kathy J. Richardson, and Dirk Grunwald. Performance Issues of Enterprise Level Web Proxies. In *Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, volume 25,1 of *Performance Evaluation Review*, pages 13–23, New York, June 15–18 1997. ACM Press.

[61] Carlos Maltzahn, Kathy J. Richardson, Dirk Grunwald, and James H. Martin. On Bandwidth Smoothing. In *Proceedings of the 4th International Web Caching Workshop*, April 1999. URL : http://www.ircache.net/Cache/Workshop99/Papers/maltzahn-final.ps.gz.

[62] Evangelos P. Markatos, Manolis G.H. Katevenis, Dionisis Pnevmatikatos, and Michail D. Flouris. Secondary Storage Management for Web Proxies. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems (USITS'99)*, pages 93–104, October 1999. URL : http://archvlsi.ics.forth.gr/papers/1999.USITS99.web_proxy_storage.ps.gz.

[63] Larry McVoy and Carl Staelin. lmbench: Portable Tools for Performance Analysis. In USENIX, editor, *USENIX 1996 Annual Technical Conference, January 22–26, 1996. San Diego, CA*, pages 279–294, Berkeley, CA, USA, January 1996. USENIX. URL : http://bitmover.com/lmbench/.

[64] Jeffrey C. Mogul. Speedier Squid : A Case Study of an Internet Server Performance Problem. *;login : The USENIX Association Magazine*, 24(1):50–58, 1999.

[65] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta-encoding and data compression for

HTTP. In *Proceedings of the ACM SIGCOMM Conference : Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM-97)*, volume 27,4 of *Computer Communication Review*, pages 181–196, New York, September 14–18 1997. ACM Press.

[66] Netscape, Inc. Netscape Proxy Server. URL: http://home.netscape.com/proxy/v3.5/datasheet/.

[67] Inc. Network Appliance. NetCache Appliances. URL: http://www.netapp.com/products/netcache/.

[68] Inc. Novell. Novell BorderManager. URL: http://www.novell.com/bordermanager/.

[69] J. Ousterhout, M. Seltzer, and K. Smith. Ousterhout's critiques of LFS and response by Seltzer and Smith, and reply by Ousterhout. URLS: http://www.sunlabs.com/people/john.ousterhout/seltzer93.html, http://www.sunlabs.com/people/john.ousterhout/seltzer.html and http://www.eecs.harvard.edu/~margo/usenix.195/ouster.html, 1995.

[70] J. K Ousterhout, H. Da Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson. A Trace-Driven Analysis of the UNIX 4.2 BSD File System. Technical Report UCB/CSD 85/230, University of California, April 1985.

[71] D. Patterson et al. RAID: Redundant arrays of inexpensive disks. In *19 ACM SIGMOD Conf. on the Management of Data, Chicago*, June 1988.

[72] Vern Paxson and Sally Floyd. Wide-area traffic: the failure of Poisson modeling. *ACM SIGCOMM 94*, pages 257–268, August 1994. URL: ftp://ftp.ee.lbl.gov/papers/poisson.ps.Z.

[73] Oskar Pearson and Julian Anderson. Squid User's Guide : Bottlenecks and performance issues, 2000. URL: http://squid.nlanr.net/Doc/Users-Guide/opt/performance.html.

[74] The Censorware Project. Size of the web: A dynamic essay for a dynamic medium, February 2000. URL: http://censorware.org/web_size/.

[75] Michael Rabinovich, Jeff Chase, and Syam Gadde. Not all hits are created equal: Cooperative Proxy Caching over a Wide-Area Network. *Computer Networks and ISDN Systems*, 30(22–23):2253–2259, November 1998.

URL : http://www.elsevier.com/cas/tree/store/comnet/sub/1998/30/22-23/2055.pdf.

[76] Luigi Rizzo and Lorenzo Vicisano. Replacement policies for a proxy cache. Technical Report RN/98/13, UCL-CS, 1998. URL : http://www.iet.unipi.it/ luigi/lrv98.ps.gz.

[77] Alex Rousskov and Valery Soloviev. Caching Policies for Reducing Disk I/Os in a Web Proxy, March 1998. URL : http://www.cs.ndsu.nodak.edu/ soloviev/paperSavingIO.ps.

[78] Alex Rousskov and Valery Soloviev. Eliminating the I/O Bottleneck in Large Web Caches. In *Proceedings of the 6th NASA/IEEE Conference on Mass Storage Systems, College Park, Maryland*, March 1998.

[79] Alex Rousskov and Valery Soloviev. On Peformance of Caching Proxies. In *Proceedings of the 1998 ACM SIGMETRICS International Conference on the Performance of Computer Systems*, pages 272–273, 1998.

[80] Alex Rousskov, Duane Wessels, and Glenn Chisholm. The First IR-Cache Web Cache Bake-off : The Official Report, April 1999. URL: http://bakeoff.ircache.net/.

[81] Richard Rubenstein, Harry M. Hersch, and Henry F. Ledgard. *The Human Factor: Designing Computer Systems for People*. Digital Press, Burlington, MA, 1984.

[82] Margo Seltzer. Clustering, Logging and Journaling: approaches to high performance file system design. Seminar Slides, available at ftp://virtual.harvard.edu/pub/margo/mar17.95.ps, March 17 1995.

[83] Margo Seltzer, Keith Bostic, Marshall Kirk McKusick, and Carl Staelin. An Implementation of a Log-Structured File System for UNIX. In *Usenix Conference*, pages 307–326, Winter 1993.

[84] Margo Seltzer, Keith A. Smith, Hari Balakrishnan, Jacqueline Chang, Sara McMains, and Venkata Padmanabhan. File System Logging Versus Clustering: A Performance Comparison. In *Usenix Annual Technical Conference*, 1995.

[85] The National Janet Web Cache Service. Toplevel Cache Server Survey, 1999. URL: http://wwwcache.ja.net/Surveys/TopLevelCacheServerSurvey.html.

[86] Junho Shim, Peter Scheuermann, and Radek Vingralek. Proxy Cache Design: Algorithms, Implementation and Performance. *IEEE Transactions on Knowledge and Data Engineering*, 1999. URL : http://www.ece.nwu.edu/ shimjh/publication/tkde98.ps.

[87] Z. Smith. The Truth About the Web Crawling Through Eternity. *Web Techniques Magazine*, 2(5), May 1997. URL: http://www.alexa.com/company/internet_stats.html.

[88] The Internet Society. Hypertext Transfer Protocol - HTTP/1.1 RFC, 1999. URL: ftp://ftp.isi.edu/in-notes/rfc2616.txt.

[89] Valery Soloviev. Prefetching in Segmented Disk Cache for Multi-Disk Systems. In *Proceedings of the 4th ACM-IEEE Annual Workshop on I/Os in Parallel and Distributed Systems (IOPADS'96), Philadelphia, PA*, pages 69–82, May 1996. URL : http://www.cs.ndsu.nodak.edu/ soloviev/paperCacheDeclNew.ps.

[90] Valery Soloviev and Andrew Yahin. File Placement in a Web Cache Server. In *Proceedings of the 10th ACM Symposium on Parallel Algorithms and Architectures, Puerto Vallarta, Mexico*, July 1998. URL : http://www.cs.ndsu.nodak.edu/ soloviev/paperProxyDisk.ps.

[91] Alteon Web Systems. New Options in Optimizing Web Cache Deployment (White Paper), 1999. URL: http://www.alteonwebsystems.com/products/white_papers/WCR/index.shtml.

[92] Igor Tatarinov, Alex Rousskov, and Valery Soloviev. Static Caching in Web Servers. In *Proceedings of the 6th IEEE International Conference on Computer Communications and Networks (IC3N'97), Las Vegas, Nevada*, pages 410–417, September 1997.

[93] Renu Tewari, Michael Dahlin, Harrick Vin, and John Kay. Beyond hierarchies: Design considerations for distributed caching on the Internet. Technical Report TR98-0, University of Texas, February 1998. URL: http://www.cs.utexas.EDU/users/dahlin/papers/tr98-04.ps.

[94] The ATM Forum. Asynchronous Trasfer Mode (ATM). URL: http://www.atmforum.org/.

[95] The Gigabit Ethernet Alliance. Gigabit Ethernet. URL: http://www.gigabit-ethernet.org/.

[96] Graphics Visualization, Usability (GVU) Center, College of Computing, and Georgia Institute of Technology Atlanta. GVU's WWW User Surveys. URL: http://www.cc.gatech.edu/gvu/user_surveys/.

[97] WWW Consortium (W3C). A Little History of the World Wide Web. URL: http://www.w3.org/History.html.

[98] WWW Consortium (W3C). Extensible Markup Language (XML), April 1997. URL: http://www.w3.org/XML/.

[99] WWW Consortium (W3C). HTTP - Hypertext Transfer Protocol, April 1997. URL: http://www.w3.org/Protocols/.

[100] WWW Consortium (W3C). HTML 4.01 Specification (W3C Recommendation), December 1999. URL: http://www.w3.org/TR/html401/.

[101] D. Wessels and K. Claffy. Internet Cache Protocol (ICP), version 2 (Internet Draft), September 1997. URL: http://ircache.nlanr.net/Cache/ICP/ICP-id.txt.

[102] Duanne Wessels. Squid Internet Object Cache, 1996. URL: http://squid.nlanr.net/Squid/.

[103] Duanne Wessels. Squid version 2.3, 2000. URL: http://squid.nlanr.net/Versions/v2/2.3/.

[104] Stephen Williams, Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, and Edward A. Fox. Removal Policies in Network Caches for World-Wide Web Documents. In *Proceedings of ACM SIGCOMM, Stanford, CA*, pages 293–305, August 1997.

[105] Roland P. Wooster and Marc Abrams. Proxy Caching that Estimates Page Load Delays. *Computer Networks and ISDN Systems*, 29:1497–1505, 1997.

[106] Neal E. Young. On-Line File Caching as Cache Size Varies. In *Proceedings of the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 241–250, New York, January 1991. ACM Press.

[107] Neal E. Young. On-Line File Caching. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-98)*, pages 82–86, New York, January 25–27 1998. ACM Press.

[108] Robert H. Zakon. Hobbes' Internet Timeline. URL: http://info.isoc.org/guest/zakon/Internet/History/HIT.html.

# Ευρετήριο