ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μελέτη μηχανισμού διασφάλισης συνέπειας εξυπηρετητών κρυφής μνήμης Παγκόσμιου Ιστού, με εφαρμογή της θεωρίας Βέλτιστης Παύσης

Βασίλειος Α. Τσιρώνης

Επιβλέποντες: Ευστάθιος Χατζηευθυμιάδης, Αναπληρωτής Καθηγητής

ΑΘΗΝΑ

ΜΑΡΤΙΟΣ 2013

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μελέτη μηχανισμού διασφάλισης συνέπειας εξυπηρετητών κρυφής μνήμης Παγκόσμιου Ιστού, με εφαρμογή της θεωρίας Βέλτιστης Παύσης

Βασίλειος Α. Τσιρώνης

Α.Μ.: 993

Επιβλέποντες: **Ευστάθιος Χατζηευθυμιάδης**, Αναπληρωτής Καθηγητής

Εξεταστική Επιτροπή: **Ευστάθιος Χατζηευθυμιάδης**, Αναπληρωτής Καθηγητής
Λάζαρος Μεράκος, Καθηγητής
Ευάγγελος Ζέρβας, Καθηγητής

ΑΘΗΝΑ

Μάρτιος 2013

ΠΕΡΙΛΗΨΗ

Η κρυφή μνήμη (cache) και ο μηχανισμός διαχειρισής της (caching) είναι απο τα πλέον σημαντικά ζητήματα στην Πληροφορική επιστήμη (Computer science). Το πρόβλημα που ανακύπτει απο την χρήση ενός τέτοιου μηχανισμού είναι γνωστό ως πρόβλημα συνέπειας κρυφής μνήμης (cache consistency problem). Η χρήση ενός caching μηχανισμού στον Παγκόσμιο Ιστό, ΠΙ (World Wide Web, WWW ή Web) ονομάζεται Web caching και το αντίστοιχο πρόβλημα Web caching consistency problem. Μέσω Web caching ικανοποιείται η επιτακτική ανάγκη για έναν ΠΙ γρήγορο και εύρωστο. Η πρόσφατη εξάπλωση του ΠΙ θέτει εκ νέου το πρόβλημα της επεκτασιμότητας (scalability), με άμεση συνέπεια τη μεγαλύτερη ανάγκη για αποδοτικότερο Web caching που θα διασφαλίζει αποσυμφόρηση δικτύου, μείωση φόρτου στους εξυπηρετητές (servers), εξοικονόμηση εύρους ζώνης (bandwidth) και μείωση καθυστέρησης απόκρισης (latency), ενώ ταυτόχρονα θα προσφέρει υψηλή συνέπεια (consistency). Στη παρούσα διπλωματική εργασία μελετάται το Web caching consistency problem απο τη πλευρά των εξυπηρετητών κρυφής μνήμης (cache servers) και απο μια νέα σκοπιά, αυτή της θεωρίας Βέλτιστης Παύσης (Optimal Stopping). Συγκεκριμένα, μελετάται η δυνατότητα υλοποίησης ενός μηχανισμού διασφάλισης συνέπειας στους εξυπηρετητές κρυφής μνήμης ΠΙ, εφαρμόζοντας αρχές της θεωρίας Βέλτιστης Παύσης. Για το λόγο αυτό υλοποιείται ένας προσομοιωτής (simulator) βάσει της θεωρίας αυτής, στον οποίο εισάγονται δεδομένα απο web traces και στη συνέχεια εκτελούνται κατάλληλες προσομοιώσεις οι οποίες βοηθούν στην εξαγωγή συμπερασμάτων.

Θεματική Περιοχή: Κρυφή Μνήμη Παγκόσμιου Ιστού.

Λέξεις Κλειδιά: κρυφή μνήμη, συνέπεια, Παγκόσμιος Ιστός, εξυπηρετητές κρυφής μνήμης, βέλτιστη παύση.

ABSTRACT

The cache memory along with its management mechanism are amongst the most significant issues in the Computer science. The problem which is related with caching is known as cache consistency problem. The use of a caching mechanism in the World Wide Web (WWW or Web) is called Web caching and the relative problem is called Web caching consistency problem. The urgent need for a robust and fast Web is satisfied through Web caching. The recent wide spreading of Web has posed once again the scalability problem, with direct consequence the even bigger need for a more efficient Web caching which will reduce the network congestion, the load in servers, the bandwidth consumption and the latency, while at the same time will provide high consistency. In the current master thesis, the Web caching consistency problem is studied from the cache servers point of view and from a new perspective which is related with the Optimal Stopping theory. More specifically, it is being studied the potential of a consistency mechanism implementation for the Web caching servers, applying the principles of the Optimal Stopping theory. For that reason, a simulator based on that theory is being implemented, and is loaded with data taken from web traces, for the execution of the appropriate simulations which will draw the necessary conclusions.

Subject Area: Web caching.

Keywords: cache, consistency, World Wide Web, cache servers, optimal stopping.

Θα ήθελα να αφιερώσω την εργασία αυτή στην μνήμη του Κώστα και του Πελοπίδα

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να εκφράσω τις ευχαριστίες μου προς τον επιβλέποντα της εργασίας κ. Ευστάθιο Χατζηευθυμιάδη (Αναπληρωτής Καθηγητής, Τμήμα Πληροφορικής και Τηλεπικοινωνιών, ΕΚΠΑ), που μου πρότεινε και μου εμπιστεύτηκε τη διεκπεραίωση της παρούσας διπλωματικής εργασίας, μιας εργασίας που με έφερε σε επαφή με σημαντικούς τομείς της Πληροφορικής επιστήμης, καθώς επίσης και της Στατιστικής επιστήμης, συμβάλλοντας έτσι στην περαιτέρω διεύρυνση των πνευματικών μου οριζόντων. Θα ήθελα ακόμα να ευχαριστήσω τον προσωπικό μου φίλο Δημήτριο Παπαβασιλείου, για την ηθική υποστήριξη και τις πολύτιμες τεχνικές συμβουλές που μου παρείχε κατα τη διάρκεια της υλοποίησης της παρούσας εργασίας.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

ΠΡΟΛΟΓΟΣ

Η παρούσα διπλωματική εργασία εκπονήθηκε στην Αθήνα κατα την περίοδο 2012 - 2013 υπο την επίβλεψη του Αναπληρωτή Καθηγητή του τμήματος Πληροφορικής και Τηλεπικοινωνιών, του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών, κ. Ευστάθιο Χατζηευθυμιάδη. Στόχος αυτής της διπλωματικής εργασίας είναι η μελέτη του προβλήματος της διασφάλισης συνέπειας στα δεδομένα των κρυφών μνημών του ΠΙ (Web caches) απο μια καινούργια σκοπιά, αυτή της στατιστικής ακολουθιακής ανάλυσης και πιο συγκεκριμένα της θεωρίας της Βέλτιστης Παύσης.

Σε αυτά τα πλαίσια, λοιπόν, η εκπόνηση της διπλωματικής εργασίας εξελίχθηκε ως εξής:

1. Συγκέντρωση και μελέτη της βιβλιογραφίας που σχετίζεται με τον μηχανισμό κρυφής μνήμης στον ΠΙ και πιο συγκεκριμένα με το πρόβλημα της συνέπειας σε αυτόν.

2. Συγκέντρωση και μελέτη της βιβλιογραφίας που σχετίζεται με την θεωρία της Βέλτιστης Παύσης.

3. Μοντελοποίηση του προβήματος της συνέπειας στις κρυφές μνήμες ΠΙ, ως ένα πρόβλημα της θεωρίας Βέλτιστης Παύσης.

4. Σχεδιασμός και υλοποίηση προσομοιωτή μιας κρυφής μνήμης ΠΙ, που θα εφαρμόζει την μοντελοποίηση της προηγούμενης φάσης.

5. Διεξαγωγή κατάλληλων προσομοιώσεων απαραίτητων για την εξαγωγή συμπερασμάτων σχετικών με την μελέτη του μηχανισμού συνέπειας που μοντελοποιείται στην παρούσα εργασία.

6. Σύνταξη του παρόντος κειμένου στο οποίο παρουσιάζονται συνολικά και με πληρότητα τα αποτελέσματα της παρούσας μελέτης.

Επιπλέον είναι σημαντικό να αναφερθεί, πως απο αυτό το σημείο του κειμένου και έπειτα, θα προτιμάται η χρήση του ξενόγλωσσου όρου έναντι του ελληνικού όπου αυτή κρίνεται καταλληλότερη προς χάριν της συνεκτικότερης και ακριβέστερης ορολογίας. Για παράδειγμα αντί του όρου Παγκόσμιος Ιστός θα προτιμάται ο όρος Web, αντιστοίχως αντί του όρου κρυφή μνήμη Παγκοσμίου Ιστού θα προτιμάται ο όρος Web cache, όπως επίσης ο όρος Web caching κ.ο.κ.

Τέλος, παραδόθηκε ένας οπτικός δίσκος (CD) στον οποίο περιλαμβάνεται ο πηγαίος κώδικας του προσομοιωτή, τα εκτελέσιμα αρχεία και όλα τα ψηφιακά αρχεία τα οποία είναι απαραίτητα για την πλήρη τεκμηρίωση της εργασίας.

1. ΕΙΣΑΓΩΓΗ

Το Web είναι ένα ταχέως εξελισσόμενο και ευρέως διαδεδομένο **πληροφοριακό σύστημα, ΠΣ** (information system, IS) αποτελούμενο απο διασυνδεδεμένα έγγραφα υπερκειμένου τα οποία προσπελαύνονται μέσω του Διαδικτύου. Η εξάπλωση του Web, καθώς και του Διαδικτύου είναι ραγδαία στις μέρες μας. Αυτή η εξάπλωση έχει δύο όψεις: απο τη μία εκφράζεται μέσω της τεράστιας αύξησης των χρηστών του Διαδικτύου[1] [1, 2], και απο την άλλη μέσω της ολοένα και μεγαλύτερης ανάπτυξης τεχνολογιών γύρω απο το Διαδίκτυο και το Web ειδικότερα. Μέσα στα πλαίσια αυτά, τίθονται εκ νέου προβλήματα όσον αφορά την επεκτασιμότητα του Web, δηλαδή την ικανοτητά του να συνεχίζει να υποστηρίζει αποδοτικά τις ολοένα και αυξανόμενες απαιτήσεις. Για την επίλυση αυτών των προβλημάτων έχουν προταθεί και συνεχίζουν να προτείνονται διάφορες λύσεις. Μια απο τις πλέον εδραιωμένες λύσεις είναι η χρήση Web caches.

Η λογική των Web caches είναι απλή και ανάλογη με αυτή στα συστήματα μνήμης των προσωπικών υπολογιστών. Μια κρυφή μνήμη[2] είναι μια μνήμη η οποία βρίσκεται πολύ κοντά στην **Κεντρική Μονάδα Επεξεργασίας, ΚΜΕ** (Central Processor Unit, CPU) προσφέροντας γρήγορη προσπέλαση δεδομένων. Παρομοίως, μια κρυφή μνήμη που βρίσκεται κοντά στον σκληρό δίσκο είναι μια μνήμη η οποία χρησιμοποιείται για να αποθηκευθούν δεδομένα του σκληρού δίσκου που προσπελαύνονται συχνά και έτσι να αυξηθεί η ταχύτητα προσπελασής τους. Κατα την ίδια λογική ο Web caching μηχανισμός είναι ένας μηχανισμός αποθήκευσης Web αντικειμένων[3] κοντά στον χρήστη, ούτως ώστε να επιτυγχάνεται ταχεία προσπέλαση αυτών, βελτιώνοντας την αντίληψη και την εμπειρία περιήγησης του χρήση στο Web. Για την υποστήριξη του Web caching μηχανισμού, υπάρχουν διάφοροι τύποι Web caches[4], μερικές εκ των οποίων βρίσκονται στον ίδιο τον υπολογιστή του χρήστη και άλλες σε κάποιο Web server, ο οποίος εναλλακτικά ονομάζεται cache server. Σε κάθε περίπτωση, ο Web caching μηχανισμός παρέχει πολλά πλεονεκτήματα, αλλά θέτει και σημαντικά προβλήματα στην εύρυθμη λειτουργία του Web.

Ένα απο τα σημαντικότερα προβλήματα που ανακύπτει απο την χρήση του μηχανισμού αυτού και το οποίο θα διερευνηθεί στα πλαίσια της παρούσας εργασίας είναι η συνέπεια του συστήματος (σ.σ. του Web). Όπως γίνεται αντιληπτό, μια Web cache διαχειρίζεται αντίγραφα αντικειμένων και όχι τα ίδια τα αντικείμενα. Εάν, λοιπόν, ένα αντικείμενο αλλάξει στην πηγή και αυτή η αλλαγή δεν κοινοποιηθεί στις Web caches που κατέχουν αντίγραφα του αντικειμένου αυτού, τότε το σύστημα θα βρεθεί σε μη συνεπή κατάσταση. Ακόμα χειρότερα εάν ένας χρήστης στείλει μια **αίτηση** (request) γι αυτό το αντικείμενο και η αίτηση αυτή ικανοποιηθεί απο μια Web cache που δεν έχει ενημερωθεί για την αλλαγή του πηγαίου αντικειμένου, ο χρήστης θα

[1] Σύμφωνα με στοιχεία του Internet World Statistics, οι χρήστες του Διαδικτύου έχουν αυξηθεί πάνω απο πέντε φορές κατα την περίοδο 2000 - 2012.

[2] Στο σημείο αυτό υπενθυμίζεται οτι ο ξενόγλωσσος όρος για την κρυφή μνήμη είναι cache.

[3] Μερικές φορές αντι του όρου Web αντικείμενα χρησιμοποιείται ο όρος Web έγγραφα. Αυτοί οι δύο όροι είναι ισοδύναμοι και χρησιμοποιούνται για να εκφράσουν το σύνολο των αντικειμένων που διακινούνται μέσω του Web (π.χ. HTML σελίδες, εικόνες κ.λπ.). Στα πλαίσια αυτού του κειμένου θα προτιμηθεί ο όρος Web αντικείμενα.

[4] Όλοι αυτοί οι τύποι Web caches θα περιγραφούν αναλυτικά στο επόμενο κεφάλαιο.

λάβει μια **απάντηση** (response) που θα περιέχει **έωλα** (stale) δεδομένα.

Αυτή τη στιγμή υπάρχουν αρκετοί Web caching μηχανισμοί οι οποίοι προσπαθούν να επιλύσουν το πρόβλημα της συνέπειας ή τουλάχιστον να περιορίσουν τις επιπτώσεις του στην λειτουργία του Web. Οι μηχανισμοί αυτοί χωρίζονται σε δύο μεγάλες κατηγορίες: σε αυτούς που παρέχουν ισχυρή συνέπεια και σε αυτούς που παρέχουν ασθενή. Την παρούσα εργασία θα απασχολήσουν οι μηχανισμοί που ανήκουν στη δεύτερη κατηγορία, καθότι το **Πρωτόκολλο Μεταφοράς Υπερκειμένου** (Hypertext Transfer Protocol, HTTP) [3], που είναι το πρωτόκολλο εκείνο το οποίο χρησιμοποιεί το Web, παρέχει ένα σύνολο κανόνων και λειτουργιών που μπορούν να υποστηρίξουν κυρίως αυτούς του μηχανισμούς.

Στα πλαίσια, λοιπόν, της εργασίας αυτής θα μελετηθεί εκ νέου το πρόβλημα της συνέπειας του Web caching, όπως αυτό τίθεται απο τους περιορισμούς που επιβάλει το HTTP, καθώς και η ανάγκη για επεκτασιμότητα αλλά και ευελιξία του Web. Θα μελετηθεί μάλιστα το πρόβλημα αυτό απο μια καινούργια σκοπία, καθώς θα γίνει εφαρμογή της θεωρίας Βέλτιστης Παύσης [4], μιας ραγδαία εξελισσόμενης θεωρίας τα τελευταία πενήντα χρόνια, που ανήκει στην Στατιστική επιστήμη και πιο συγκεκριμένα στον τομέα της Ακολουθιακής Στατιστικής Ανάλυσης. Συνεπώς θα προταθεί ένας νέος μηχανισμός ασθενούς συνέπειας, ο οποίος θα προσομοιωθεί με την υλοποίηση κατάλληλου λογισμικού και την είσοδο σε αυτό δεδομένων που προέρχονται απο Web traces, ώστε να εξαχθούν τα κατάλληλα συμπεράσματα όσον αφορά την απόδοση του.

1.1 Αντικείμενο της Εργασίας

Αντικείμενο της εργασίας είναι το Web caching consistency problem. Πιο συγκεκριμένα, γίνεται μια προσπάθεια να εφαρμοστεί η θεωρία Βέλτιστης Παύσης στην επίλυση του προβλήματος αυτού. Ουσιαστικά, μελετάται μέσω της σχεδίασης και υλοποίησης κατάλληλου λογισμικού προσομοιωτή, η δυνατότητα υλοποίησης ενός Web caching μηχανισμού ασθενούς συνέπειας, στον οποίο να γίνεται εφαρμογή της θεωρίας Βέλτιστης Παύσης. Επιπλέον, μελετάται η απόδοση ενός τέτοιου μηχανισμού, βάσει των αποτελεσμάτων του προσομοιωτή, και γίνεται μια συγκριτική αποτίμηση με Web caching μηχανισμούς ασθενούς συνέπειας, που είναι ευρέως διαδεδομένοι.

1.2 Στόχοι της Εργασίας

Στόχος της εργασίας αυτής είναι η σχεδίαση, υλοποίηση και αποτίμηση ενός Web caching μηχανισμού ασθενούς συνέπειας, στον οποίο θα γίνεται εφαρμογή της θεωρίας Βέλτιστης Παύσης και ο οποίος θα επιτυγχάνει υψηλότερο ποσοστό συνέπειας, σε σύγκριση με τους ήδη υπάρχοντες. Βέβαια, δεν υλοποιείται ένα πλήρες σύστημα Web caching, αλλά ο προσομοιωτής ενός τέτοιου συστήματος. Καθώς επίσης, η απόδοση δεν μετράται συνολικά, αλλά μονάχα στα πλαίσια της συνέπειας των δεδομένων. Γι αυτό άλλωστε και δε γίνεται λόγος για υλοποίηση ενός τέτοιου μηχανισμού, αλλά για μελέτη. Τέλος, στόχος της παρούσας εργασίας είναι και η κριτική παρουσίαση τεχνολογιών που συνέβαλαν και συμβάλουν στην ανάπτυξη του Web, καθώς και των σημαντικών προόδων που έχουν γίνει στη θεωρία Βέλτιστης Παύσης.

1.3 Διάρθρωση της Εργασίας

Η παρούσα εργασία αποτελείται από έξι κεφάλαια.

- Στο Κεφάλαιο 2 παρουσιάζονται τα κυριότερα ζητήματα γύρω απο το Web caching και ειδικότερα απο το πρόβλημα της συνέπειας. Παρουσιάζονται οι διάφοροι τύποι Web caches, οι κυριότεροι μηχανισμοί ισχυρούς και ασθενούς συνέπειας, τα πλεονεκτήματα και τα προβλήματα αυτών. Δίδεται ουσιαστικά το θεωρητικό υπόβαθρο που χρειάζεται κάποιος για να κατανοήσει το Web caching consistency problem.

- Στο Κεφάλαιο 3 γίνεται μια εισαγωγή στη θεωρία Βέλτιστης Παύσης. Παρουσιάζεται η βιβλιογραφία που είναι σχετική με αυτή, δίδεται η αξιωματική της θεμελίωση και περιγράφεται ο αλγόριθμος *Sum the odds to one and stop*, που είναι καίριας σημασίας για τον μηχανισμό που μελετάται σε αυτή τη διπλωματική εργασία.

- Στο κεφάλαιο 4 παρουσιάζεται ο μηχανισμός σε θεωρητικό επίπεδο. Περιγράφονται οι μετρικές και οι κανόνες διακοπής που υλοποιούνται και η θεωρητική σημασία αυτών.

- Στο κεφάλαιο 5 παρουσιάζεται ο προσομοιωτής που αποτελεί την τεχνική υλοποίηση των θεωρητικών ευρημάτων του προηγούμενου κεφαλαίου. Δίδονται, επίσης, τα αποτελέσματα απο διάφορες προσομοιώσεις, στα οποία απεικονίζεται η απόδοση του μηχανισμού.

- Στο κεφάλαιο 6 γίνεται μια σύνοψη όσων παρουσιάστηκαν, καθώς επίσης αποτυπώνονται τα συμπεράσματα που προέκυψαν στα πλαίσια της διπλωματικής εργασίας και τα οποία σκιαγραφούν τις δυνατότητες, συνάμα τις κατευθύνσεις μελλοντικής επέκτασης της.

Τέλος, στο Παράρτημα ακολουθεί η ορολογία που χρησιμοποιήθηκε, ο πίνακας συντμήσεων-αρκτικόλεξων και η σχετική βιβλιογραφία.

2. ΚΡΥΦΗ ΜΝΗΜΗ ΠΑΓΚΟΣΜΙΟΥ ΙΣΤΟΥ

Στο κεφάλαιο αυτό θα περιγραφεί η λειτουργία του Web caching, με κύριο στόχο να αναλυθεί το Web caching consistency πρόβλημα. Για να καταστεί επιτυχής η προσπάθεια αυτή, θα παρουσιαστεί αρχικά η έννοια της κρυφής μνήμης, όπως αυτή πραγματώνεται στα συστήματα μνήμης των προσωπικών υπολογιστών. Στη συνέχεια θα παρουσιαστεί η έννοια αυτή στα πλαίσια του Web, καθώς θα περιγραφούν τα πλέον βασικά, συνάμα σημαντικά στοιχεία του Web caching και η υποστήριξη αυτών απο το πλέον διαδεδομένο πρωτόκολλο επικοινωνίας, το HTTP. Τέλος, θα αναλυθεί το πρόβλημα του Web cache consistency, όπως αυτό υφίσταται στην περίπτωση των cache servers, αφού πλέον θα έχει χτιστεί το κατάλληλο υπόβαθρο στις προηγούμενες ενότητες. Πιο συγκεκριμένα η διάρθωση του κεφαλαίου είναι η ακόλουθη:

- **Ενότητα 2.1**: Η ενότητα 2.1 κάνει μια σύντομη εισαγωγή στην έννοια της κρυφής μνήμης, περιγράφοντας την λειτουργικότητα της στα συστήματα μνήμης των προσωπικών υπολογιστών.
- **Ενότητα 2.2**: Η ενότητα 2.2 εισάγει τον αναγνώστη στον Web caching μηχανισμό, περιγράφοντας τα βασικότερα μέρη αυτού. Δίνει ουσιαστικά το περίγραμμα του πλαισίου λειτουργίας αυτού του μηχανισμού, περιγράφοντας την ανάγκη υπαρξής, την υλοποιησή, καθώς και την υποστηριξη του απο τον HTTP.
- **Ενότητα 2.3**: Η ενότητα 2.3 αναλύει το πρόβλημα της συνέπειας του Web caching μηχανισμού, παρουσιάζοντας τους μηχανισμούς που έχουν προταθεί για την επιλυσή του, μερικοί εκ των οποίων χρησιμοποιούνται ευρέως στο Web.
- **Ενότητα 2.4**: Η ενότητα 2.4 κάνει μια σύνοψη όλων όσων παρουσιάστηκαν στις προηγούμενες ενότητες.

2.1 Εισαγωγή στη Κρυφή Μνήμη

Απο την γέννηση των υπολογιστών υπήρχε η ανάγκη για απεριόριστες ποσότητες μνήμης με γρήγορη πρόσβαση στα δεδομένα τους. Για να ικανοποιηθεί η ανάγκη αυτή, η μνήμη ενός υπολογιστή είναι οργανωμένη σε μια **ιεραρχία μνήμης** (memory hierarchy), η οποία εκμεταλεύεται την αρχή της **τοπικότητας** (locality). Τι λέει, όμως, η αρχή της τοπικότητας και απο τι αποτελείται η ιεραρχία μνήμης;

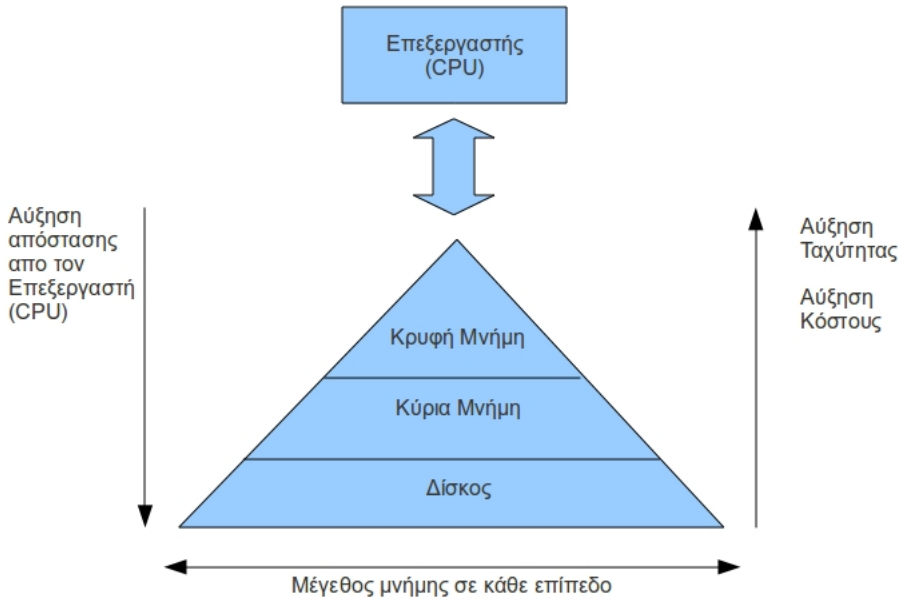
Η αρχή της τοπικότητας λέει οτι τα προγράμματα προσπελαύνουν ένα σχετικά μικρό τμήμα του χώρου διευθύνσεων τους σε κάθε χρονική στιγμή. Μάλιστα, υπάρχουν δύο διαφορετικοί τύποι τοπικότητας [5]:

- **Χρονική τοπικότητα** (temporal locality): Αν γίνει αναφορά σε ένα αντικείμενο, τότε συνήθως θα ξαναγίνει αναφορά στο ίδιο αντικείμενο σύντομα. Για παράδειγμα εάν το

στοιχείο βρίσκεται μέσα σε έναν **βρόγχο** (loop), είναι σχεδόν σίγουρο οτι θα προσπελαστεί ξανά στο άμεσο μέλλον (εφόσον η συνθήκη του βρόγχου συνεχίζει να είναι αληθής) .

- **Χωρική τοπικότητα** (spatial locality): Αν γίνει αναφορά σε ένα αντικείμενο, συνήθως θα γίνει σύντομα αναφορά σε αντικείμενα των οποίων οι διευθύνσεις είναι κοντά. Για παράδειγμα τα στοιχεία ενός πίνακα παρουσιάζουν υψηλό βαθμό χωρικής τοπικότητας, εφόσον υπο φυσιολογικές συνθήκες είναι αποθηκευμένα σε διαδοχικές θέσεις μνήμης.

Την παραπάνω αρχή εκμεταλεύονται οι υπολογιστές, χρησιμοποιώντας μια ιεραρχία μνήμης. Δηλαδή το σύστημα μνήμης ενός υπολογιστή δεν αποτελείται απο μία ενιαία γρήγορη και απεριόριστη μνήμη, αλλά απο μια ιεραρχία μνήμης η οποία αποτελείται απο πολλά επίπεδα μνήμης με διαφορετικά μεγέθη και ταχύτητες. Οι μνήμες οι οποίες βρίσκονται ψηλότερα στην ιεραρχία, είναι ταχύτερες, με μικρότερο μέγεθος και μεγαλύτερο κόστος ανα bit πληροφορίας (βλέπε Σχήμα 2.1).



Σχήμα 2.1: Ιεραρχία μνήμης.

Τρείς είναι οι βασικές τεχνολογίες που χρησιμποιούνται στις μέρες μας για την κατασκεύη μιας ιεραρχίας μνήμης. Η κύρια μνήμη υλοποιείται με **Δυναμική Μνήμη Τυχαίας Προσπέλασης** (Dynamic Random Access Memory, DRAM), ενώ τα επίπεδα που βρίσκονται πιο κοντά στον επεξεργαστή, όπως η κρυφή μνήμη, χρησιμοποιούν **Στατική Μνήμη Τυχαίας Προσπέλασης** (Static Random Access Memory, SRAM). Η DRAM έχει μικρότερο κόστος ανα bit απο την SRAM, αλλά είναι σημαντικά πιο αργή. Η τρίτη τεχνολογία, που χρησιμοποιείται για την υλοποίηση του μεγαλύτερου και πιο αργού επιπέδου της ιεραρχίας, είναι ο μαγνητικός δίσκος [5].

Λόγω, λοιπόν, αυτών των διαφορών στο κόστος κατασκευής και στην ταχύτητα προσπέλασης μεταξύ των διαφορετικών τεχνολογιών, υπάρχουν πλεονεκτήματα στη δόμηση της μνήμης

ως μια ιεραρχία. Μάλιστα όσο πιο ψηλά βρίσκεται ένα επίπεδο μνήμης στην ιεραρχία αυτή, τόσο πιο κοντά στον επεξεργαστή τοποθετείται. Επιπλέον, ένα επιπέδο το οποίο βρίσκεται πιο κοντά στον επεξεργαστή περιέχει ουσιαστικά ένα υποσύνολο δεδομένων οποιουδήποτε επιπέδου βρίσκεται μακρύτερα.

Απο τα παραπάνω είναι πλέον εμφανής ο ρόλος της κρυφής μνήμης σε μια ιεραρχία· η κρυφή μνήμη είναι η μνήμη εκείνη που βρίσκεται πιο κοντά στον επεξεργαστή, παρεχοντάς του γρήγορη πρόσβαση στα δεδομένα που χρειάζεται. Πως, όμως, το επιτυγχάνει αυτό;

Όταν ο επεξεργαστής χρειάζεται κάποια δεδομένα τα αναζητεί στην κρυφή μνήμη. Εάν τα συγκεκριμένα δεδομένα βρεθούν σε αυτή, τότε έχουμε **ευστοχία** (hit ή cache hit). Εάν δεν βρεθούν, έχουμε **αστοχία** (miss ή cache miss) και η αίτηση θα προωθηθεί σε ένα επίπεδο χαμηλότερο στην ιεραρχία και πιο απομακρυσμένο απο τον επεξεργαστή [1]. Σε περίπτωση αστοχίας τα δεδομένα που θα βρεθούν σε ένα επίπεδο χαμηλότερο απο την κρυφή μνήμη, θα αντιγραφούν σε αυτή, ούτως ώστε οι επόμενες αιτήσεις του επεξεργαστή προς αυτά να ικανοποιηθούν απο τη κρυφή μνήμη. Απο τη στιγμή που υπάρχουν πολλά (συχνά χιλιάδες) μπλοκ[2] στην κρυφή μνήμη, χρειάζεται κάποιος τρόπος για να εξακριβώνεται γρήγορα εαν βρίσκεται σε αυτή κάποιο συγκεκριμένο μπλοκ. Ο συνηθισμένος τρόπος είναι να κατακερματίζονται οι διευθύνσεις των συσκευών και του δίσκου και να γίνεται αναζήτηση σε **πίνακα κατακερματισμού** (hash table). Όλα τα μπλοκ που έχουν την ίδια τιμή κατακερματισμού τοποθετούνται μαζί σε μια συνδεδεμένη λίστα ώστε να δημιουργηθεί η **αλυσίδα σύγκρουσης** (collision chain) [5, 6].

Επιπλέον, όταν μια κρυφή μνήμη είναι γεμάτη και πρόκειται να φορτωθεί σε αυτή ένα καινούριο μπλοκ, πρέπει να δημιουργηθεί ο κατάλληλος χώρος αφαιρώντας απο αυτή κάποιο άλλο μπλοκ, το οποίο θα ξαναγραφεί στο δίσκο εαν έχει τροποποιηθεί απο την ώρα που μεταφέρθηκε στη μνήμη. Σε αυτή τη περίπτωση, η κρυφή μνήμη πρέπει να παρέχει και μια συγκεκριμένη πολιτική αντικατάστασης, όπως π.χ. First In First Out, FIFO ή Least Recently Used, LRU. Στη περίπτωση της κρυφής μνήμης, μάλιστα, ο αλγόριθμος LRU είναι αρκετά αποδοτικός και εύκολα υλοποιήσιμος, λόγω του μικρού μεγέθους της αλλά και λόγω της οργάνωσης των δεδομένων σε αυτή, μιας και τα μπλοκ διατρέχονται απο μια λίστα διπλής κατεύθυνσης [6]. Συνεπώς, μπορεί εύκολα να υλοποιηθεί μια LRU πολιτική αντικατάστασης και έτσι να αξιοποιηθεί η αρχή της τοπικότητας.

Βέβαια, η παραπάνω λειτουργικότητα της κρυφής μνήμης γεννά αναπόφευκτα καινούρια προβλήματα. Προβλήματα που σχετίζονται, πρωτίστως[3], με τον χειρισμό των εγγραφών και ουσιαστικά με αυτό που έχει ήδη αναφερθεί αρκετές φορές σε προηγούμενες ενότητες, τη συνέπεια των δεδομένων ή καλύτερα τη συνέπεια της μνήμης. Εάν για παράδειγμα γίνει κάποια ενημέρωση στα δεδομένα που βρίσκονται στην κρυφή μνήμη χωρίς αυτή η ενημέρωση να εφαρμοστεί στα δεδομένα της κύριας μνήμης, τότε η κρυφή μνήμη και η κύρια είναι **ασυνεπείς** (inconsistent)[4].

[1] Το αμέσως χαμηλότερο επίπεδο μπορεί να είναι μια κρυφή μνήμη επιπέδου 2 ή η κύρια μνήμη. Και σε αυτό το επίπεδο επαναλαμβάνεται η ίδια διαδικασία εως ότου βρεθούν τα ζητούμενα δεδομένα. Εφόσον βρεθούν ακολουθεί μια αντίστροφη διαδικασία, όπου τα δεδομένα αντιγράφονται σε κάθε επίπεδο.

[2] Το μπλοκ είναι μια ακολουθία απο bytes με καθορισμένο μέγεθος. Τα δεδομένα (ή καλύτερα τα ρεύματα δεδομένων) που μετακινούνται μεταξύ συσκεύων ενός υπολογιστή ή/και των προγραμμάτων είναι δομημένα σε μπλοκ.

[3] Άλλα σημαντικά προβλήματα έχουν να κάνουν με τον χειρισμό των αστοχιών κρυφής μνήμης ή καταρρεύσεις του **συστήματος αρχείων** (file system).

[4] Ισοδύναμα το σύστημα της μνήμης δεν είναι **συνεπές** (consistent)

Ο καλύτερος τρόπος για να διατηρηθεί η συνέπεια μεταξύ κρυφής και κύριας μνήμης, είναι η εγγραφή των δεδομένων και στις δύο μνήμες μετά απο κάθε ενημέρωση αυτών. Η μέθοδος αυτή ονομάζεται **ταυτόχρονη εγγραφή**[5] (write-through) και οι κρυφές μνήμες που την εφαρμόζουν ονομάζονται write-through caches. Αν και αυτή η σχεδίαση χειρίζεται τις εγγραφές πολύ καλά, δεν έχει πολύ καλή απόδοση. Η εναλλακτική λύση στη μέθοδο της ταυτόχρονης εγγραφής είναι μια μέθοδος που ονομάζεται **ετερόχρονη έγγραφη** (write back)[6]. Σε αυτή την μέθοδο, όταν συμβαίνει μια εγγραφή, η νέα τιμή γράφεται μόνο στο μπλοκ της κρυφής μνήμης. Το τροποποιημένο μπλοκ γράφεται στο χαμηλότερο επίπεδο της ιεραρχίας όταν αντικαθίσταται. Η μέθοδος αυτή μπορεί να βελτιώσει εντυπωσιακά την απόδοση, ειδικά όταν ο επεξεργαστής κάνει πολλές εγγραφές· η υλοποιησή της όμως είναι πολύπλοκη [5–7].

Πέραν της κρυφής μνήμης που βρίσκεται στον επεξεργαστή και η οποία ανήκει στο **υλικό** (hardware) του υπολογιστή, μπορούμε να βρούμε και σε άλλα μέρη του υπολογιστή κρυφές μνήμες, ως κομμάτι του **λογισικού** (software) του. Χαρακτηριστικό παράδειγμα αποτελούν η **κρυφή μνήμη μπλοκ** (block cache) και η **κρυφή μνήμη προσωρινής αποθήκευσης** (buffer cache), που χρησιμοποιούνται για να μειωθούν οι προσβάσεις στο δίσκο. Αυτό μας δείχνει οτι η έννοια της λέξης cache που στα ελληνικά προφέρεται «κας» και προέρχεται απο το γαλλικό ρήμα cacher, που σημαίνει «κρύβω», είναι ευρύτερη και δηλώνει τη συλλογή εκείνη απο μπλοκ τα οποία ανήκουν λογικά στο δίσκο, αλλα διατηρούνται στη μνήμη για λόγους απόδοσης [6]. Επιπλέον, η έννοια της κρυφής μνήμης, δεν πρέπει να συγχέεται με αυτή της **προσωρινής μνήμης** (buffer). Η προσωρινή μνήμη αποτελεί μια πρόσκαιρη δέσμευση μνήμης με σκοπό, είτε την επικοινωνία μεταξύ συσκευών ή/και προγραμμάτων, είτε την τροποποίηση κατα μη σειριακό τρόπο δεδομένων που στη συνέχεια θα εγγραφούν σειριακά στο δίσκο.

Εν κατακλείδι, η κρυφή μνήμη είναι μια μνήμη που βρίσκεται ψηλά στην ιεραρχία μνήμης και σκοπός της είναι η παροχή γρήγορης πρόσβασης στα δεδομένα, που χρειάζεται ένα μέρος του υπολογιστή, αξιοποιώντας την αρχή της τοπικότητας. Γι αυτό το λόγο, τοποθετείται κοντά στο μέρος του υπολογιστή που την χρειάζεται· η **κρυφή μνήμη του επεξεργαστή** (CPU cache) βρίσκεται στο chip του επεξεργαστή, η **κρυφή μνήμη του σκληρού δίσκου** (dist cache) είναι μέρος του δίσκου κ.ο.κ.

2.2 Η Κρυφή Μνήμη στο Παγκόσμιο Ιστό

Στη προηγούμενη ενότητα έγινε μια συνοπτική, συνάμα περιεκτική εισαγωγή στα βασικά ζητήματα της οργάνωσης της μνήμης σε έναν υπολογιστή. Παρουσιάστηκαν οι έννοιες της ιεραρχίας μνήμης και της αρχής της τοπικότητας, μέσω των οποίων έγινε δυνατή η περιγραφή και ανάλυση της λειτουργικότητας της κρυφής μνήμης. Σε αυτή την ενότητα, θα περιγραφεί η σημασία της λειτουργικότητας αυτής, στα πλαίσια του Web.

Έχει ήδη αναφερθεί αρκετές φορές, πως το Web είναι ένα ευρέως διαδεδομένο και ταχέως εξελισσόμενο ΠΣ. Πως ορίζεται, όμως, ένα ΠΣ; Ακολουθεί ένας απλός ορισμός, ο οποίος είναι αρκετός στα πλαίσια της παρούσας εργασίας:

[5] Στη βιβλιογραφία μπορεί να εμφανιστεί και με την μετάφραση **άμεση εγγραφή**.

[6] Η εντολή sync του UNIX, η οποία εξαναγκάζει όλα τα τροποποιημένα μπλοκ να γραφούν αμέσως το δίσκο, κάνει χρήση αυτής της μεθόδου.

> *Πληροφοριακό σύστημα είναι ένα σύνολο αλληλοσυσχετιζόμενων στοιχείων που αφορούν σε διαδικασίες, υποδομή, ανθρώπινο δυναμικό και δεδομένα, τα οποία συλλέγουν, ανακτούν, επεξεργάζονται, αποθηκεύουν και διανέμουν πληροφορίες που υποστηρίζουν τη λήψη αποφάσεων, το συντονισμό και τον έλεγχο σε ένα οργανισμό.* [8]

Απο τον παραπάνω ορισμό, αλλά και απο το γεγονός οτι το Web είναι ένα ΠΣ που «πατάει» πάνω στο Διαδίκτυο[7] για να υλοποιήσει μια απο τις βασικότερες λειτουργίες του - τη **διανομή** εγγράφων υπερκειμένου - προκύπτει ένα επιπλέον σημαντικό χαρακτηριστικό· το Web είναι ένα **κατανεμημένο σύστημα, ΚΣ** (distributed system, DS).

Πρέπει, λοιπόν, σε αυτό το σημείο να δοθεί και ένας ορισμός για το ΚΣ:

> *Κατανεμημένο σύστημα είναι μια συλλογή απο ανεξάρτητες οντότητες οι οποίες συνεργάζονται για να λύσουν ένα πρόβλημα το οποίο δε μπορεί να επιλυθεί μεμονωμένα. Κατανεμημένα συστήματα υπάρχουν απο την γέννηση του σύμπαντος... Ένα κατανεμημένο σύστημα μπορεί να χαρακτηριστεί ως μια συλλογή, ως επί το πλείστον, αυτόνομων επεξεργαστών που επικοινωνούν μέσω ενός δικτύου και έχουν τα ακόλουθα χαρακτηριστικά: μη **κοινό φυσικό ρολόι** (common physical clock), μη **διαμοιραζόμενη μνήμη** (shared memory), γεωγραφική διασπορά, αυτονομία και **ανομοιογένεια** (heterogeneity)...* [9]

Είναι προφανές, σε όλους εκείνους που έχουν εμπειρία χρήσης του Web, πως οι παραπάνω ορισμοί και ειδικότερα αυτός του ΚΣ ικανοποιούνται πλήρως. Βέβαια, το γεγονός και μόνο οτι το Web είναι ένα ΚΣ, γεννάει προκλήσεις (σίγουρα και προβλήματα) τα οποία σχετίζονται άμεσα με τον σχεδιασμό και το χτισιμό του ως ένα τέτοιο σύστημα. Πιο συγκεκριμένα, μερικές απο τις λειτουργίες που πρέπει να ληφθούν υπόψιν κατα τον σχεδιασμό είναι οι ακόλουθες: **επικοινωνία** (communication), **διεργασίες** (processes), **ονοματοδοσία** (naming), **συγχρονισμός** (synchronization), **αποθήκευση** και **προσπέλαση δεδομένων**, **συνέπεια** και **αναπαραγωγή** (replication), **ανοχή σε σφάλματα** (fault tolerance), **ασφάλεια** (security), **Διεπαφή Προγραμματισμού Εφαρμογών**[8]**, ΔΠΕ** (Application Programming Interface, API) και **διαφάνεια** (transparency), **επεκτασιμότητα** και **τμηματοποίηση** (modularity) [9].

Όλα τα παραπάνω ζητήματα, λοιπόν, είναι ζητήματα που κάθε ΚΣ πρέπει να επιλύσει, άρα και το Web. Το πλέον σημαντικό ζήτημα είναι αυτό της επεκτασιμότητας, μιας έννοιας που έχει ήδη αναφερθεί σε προηγούμενες ενότητες και η οποία ορίζεται πολύ απλά ως:

> *η ικανότητα ενός συστήματος, δικτύου ή διεργασίας να διαχειρίζεται τον αυξανόμενο φόρτο εργασίας με επιδέξιο τρόπο*[9] [10, 11].

[7] Σε αυτό το σημείο υπενθυμίζεται, οτι το Διαδίκτυο είναι ένα δίκτυο αποτελούμενο απο δίκτυα, εν πολλοίς, αυτόνομων κόμβων.

[8] Στην βιβλιογραφία μπορεί να συναντηθεί και με τον όρο **Διασύνδεση Προγραμματισμού Εφαρμογών**.

[9] Ένας ισοδύναμος ορισμός είναι ο ακόλουθος: η ικανότητα ενός συστήματος, δικτύου ή διεργασίας να μεγενθύνεται, ώστε να διαχειριστεί την ίδια την αναπτυξή του.

Για να επιτευχθεί η προσδοκώμενη επεκτασιμότητα, πρέπει οι αλγόριθμοι, τα δεδομένα και οι υπηρεσίες να κατανεμηθούν όσο το δυνατόν περισσότερο. Σε αυτό το σημείο έρχονται στο προσκήνιο τεχνικές οι οποίες μας βοηθάνε να το επιτύχουμε, όπως η αποθήκευση δεδομένων στην κρυφή μνήμη (caching) και η διαχείριση της (cache management), η αναπαραγωγή, κ.α. Εάν το bandwidth του δικτύου και οι χωρητικότητες των Web servers ήταν απεριόριστα, τότε δεν θα υπήρχε η ανάγκη των παραπάνω τεχνικών· δυστυχώς, όμως, δεν είναι. Κι ενώ οι εταιρείες που παρέχουν δικτυακές υπηρεσίες (π.χ. **Πάροχοι Υπηρεσιών Διαδικτύου** (Internet Service Providers, ISPs)) αυξάνουν διαρκώς το bandwidth και τις χωρητικότητες των Web Servers, αυξάνονται ταυτόχρονα και οι ανάγκες των χρηστών του Web με την εμφάνιση ολοένα και περισσότερων απαιτητικών υπηρεσιών, κάνοντας έτσι ακόμα πιο επιτακτική την ανάγκη χρήσης και βελτίωσης των παραπάνω τεχνικών [12].

Σε συμφωνία πάντα με το θέμα της παρούσας εργασίας, καθίσταται πλέον δόκιμη η περιγραφή των βασικών ζητημάτων του Web caching. Το πρώτο καίριο ερώτημα που τίθεται αφορά την αποσαφήνιση του όρου Web caches. Με τον όρο, λοιπόν, Web caches εννοούνται τα στοιχεία εκείνα που συνθέτουν τον Web caching μηχανισμό, και τα οποία αποθηκεύουν τα Web αντικείμενα είτε τοπικά στον υπολογιστή ενός χρήστη είτε σε έναν Web server ή καλύτερα cache server[10]. Τέτοια στοιχεία μπορεί να είναι ένα απο τα ακόλουθα [13, 14]:

- **Κρυφή μνήμη προγράμματος περιήγησης** (browser cache): Ένα **πρόγραμμα περιήγησης**[11] (browser) αποθηκεύει συνήθως τα Web αντικείμενα, που προσπέλασε πρόσφατα, στην κύρια μνήμη ή/και στον δίσκο του υπολογιστή του χρήστη.

- **Πληρεξούσιος κρυφής μνήμης** (proxy cache): Ένας proxy cache είναι εγκατεστημένος κοντά στους Web χρήστες, ας πούμε στα πλαίσια μιας **επιχείρησης** (enterprise). Οι χρήστες της επιχείρησης έχουν ρυθμίσει τους browsers να χρησιμοποιούν τον **πληρεξούσιο** (proxy) και επομένως όλες οι αιτήσεις περνάνε απο αυτόν. Εάν ο proxy βρεί το ζητούμενο αντικείμενο στην κρυφή του μνήμη τότε την ικανοποιεί, ειδάλλως προωθεί την αίτηση σε κάποιον άλλο proxy ή στην ίδια την **ιστοσελίδα** (website). Για το λόγο αυτό ονομάζεται και **πληρεξούσιος προώθησης** (forward proxy).

- **Διάφανος πληρεξούσιος κρυφής μνήμης** (transparent proxy cache): Ο transparent proxy cache ο οποίος μπορεί να συναντηθεί στην βιβλιογραφία και ως **πύλη εισόδου** (gateway) ή μερικές φορές ως **πληρεξούσιος διοχέτευσης** (tunneling proxy) και ο οποίος αφήνει τις αιτήσεις/απαντήσεις να περνάνε δίχως να τις τροποποιεί, είναι μια περίπτωση proxy που δεν μας αφορά.

- **Αντίστροφος πληρεξούσιος κρυφής μνήμης** (reverse proxy cache): Ένας reverse proxy cache ή surrogate είναι μια περίπτωση proxy που τοποθετείται μπροστά απο ένα website, με σκοπό να μειώσει τον φόρτο εργασίας στους servers του website, εκμεταλευόμενος κυρίως την τεχνική του replication. Κατα τ΄ άλλα η λειτουργία του είναι ακριβώς ίδια με αυτή του κανονικού proxy cache, δηλαδή του forward proxy.

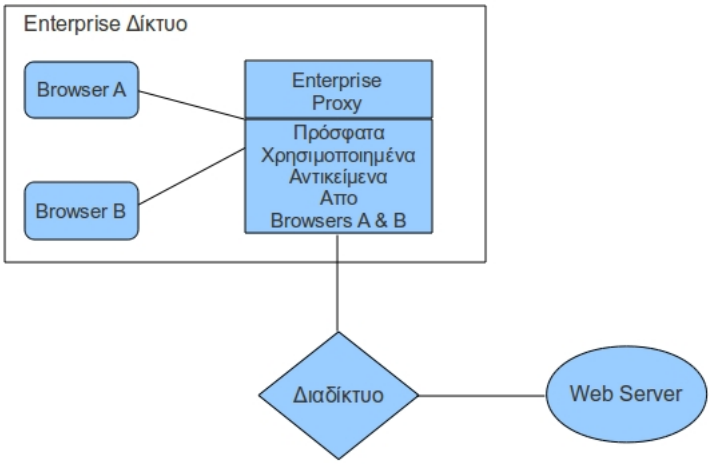
[10] Απο το σημείο αυτό και έπειτα εάν χρειαστεί να χρησιμοποιηθεί κάποιος απο τους δύο όρους, θα προτιμηθεί ο όρος cache server.

[11] Στην βιβλιογραφία μπορεί να συναντηθεί και με τον όρο φυλλομετρητής.

Στο σημείο αυτό καθίσταται απαραίτητο να αποσαφηνιστεί η σχετική με το Web caching ορολογία που έχει διατυπωθεί, καθότι δεν είναι αμελητέα και επομένως μπορεί να προκαλεί μια σχετική σύγχυση στον αναγνώστη. Ο όρος, λοιπόν, Web caches περιλαμβάνει και τις browser caches και τους cache servers. Ο όρος cache server είναι συνώνυμος του όρου Web caching proxy. Οι Web caching proxies σε αντίθεση με τις browser caches που αφορούν έναν χρήστη, χρησιμοποιούνται για να παρέχουν διαμοιραζόμενη μνήμη σε έναν αριθμό απο χρήστες. Όμως, οι πληρεξούσιοι του Web (Web proxies) δεν προσφέρουν μονάχα λειτουργικότητα κρυφής μνήμης. Για το λόγο αυτό υπάρχει στην βιβλιογραφία ο εξειδικευμένος όρος Web caching proxies, ώστε να διακρίνονται οι proxies, που παρέχουν λειτουργικότητα κρυφής μνήμης, απο τους υπολοίπους. Τέλος, οι Web caching proxies χωρίζονται σε δύο μεγάλες κατηγορίες: τους forward Web caching proxies και τους reverse Web caching proxies. Επειδή, όμως, η παρούσα εργασία ενδιαφέρεται μονάχα για τους forward Web caching proxies και την συμβολή τους στο Web caching, η χρήση του όρου proxy στο υπόλοιπο της εργασίας θα ισοδυναμεί με αυτή του forward Web caching proxy.

Συνεπώς, γίνεται αντιληπτό οτι το πρόβλημα της συνέπειας του Web caching, που μελετάται στα πλαίσια αυτής της εργασίας, μελετάται μέσα απο την λειτουργία ενός forward Web caching proxy. Αυτό δεν επηρεάζει ιδιαίτερα τα αποτελέσματα της εργασίας, μιας και η λύση που προτείνεται έχει γενική εφαρμογή.

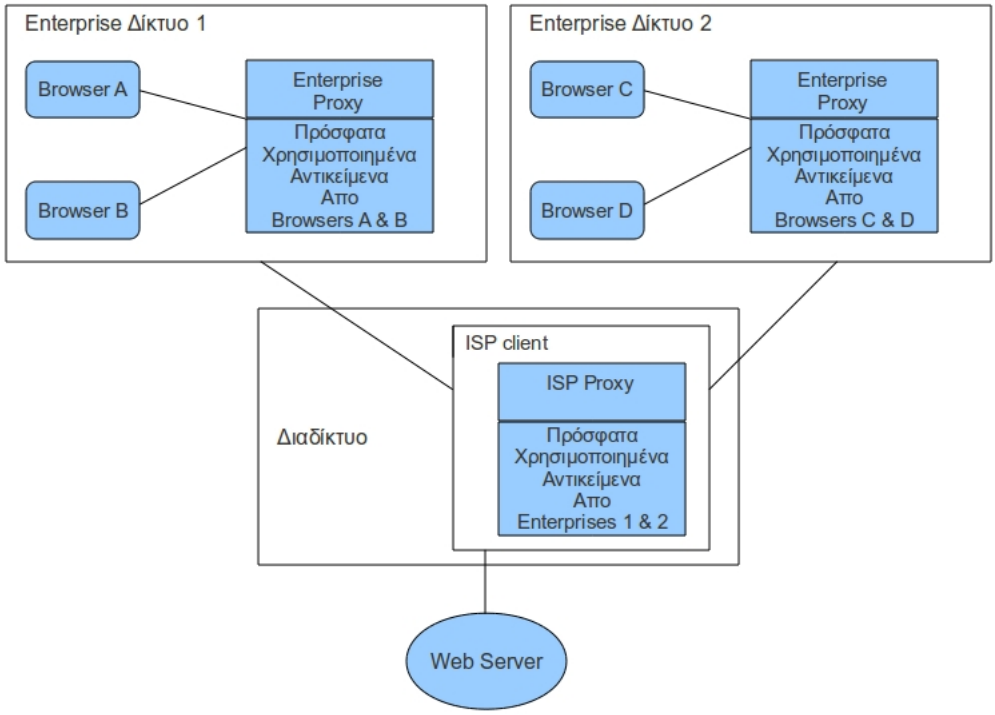
Πλέον μπορεί να περιγραφει η βασική λειτουργία του Web caching, κάνοντας χρήση αντιπροσωπευτικών παραδειγμάτων. Ας πάρουμε το παράδειγμα ενός επιχειρησιακού δικτύου. Αρχικά ένας browser προσπαθεί να ικανοποιήσει τις αίτησεις στην τοπική του cache, στέλνοντας τις αιτήσεις εκείνες που δεν μπορεί να ικανοποιήσει στον proxy της επιχείρησης. Ο proxy με τη σειρά του προσπαθεί να ικανοποιήσει τις αίτησεις βάσει της δικής του cache και προωθεί στον Web server όλες εκείνες που δεν μπορεί να διαχειριστεί [15] (βλέπε Σχήμα 2.2).



Σχήμα 2.2: Browsers χρησιμοποιούν έναν Web proxy.

Επειδή οι browsers λαμβάνουν απαντήσεις απο τον proxy, ο proxy λειτουργεί ως Web server γι αυτούς. Την ίδια στιγμή ο proxy λειτουργεί ως Web client προς τον Web server, αφού του στέλνει αιτήσεις για λογαριασμό των browsers που εξυπηρετεί. Στη διττή αυτή φύση οφείλεται και η χρήση του όρου proxy που στα ελληνικά μεταφράζεται ως πληρεξούσιος [15].

Με την ίδια λογική που ένας proxy μπορεί να ικανοποιήσει αιτήσεις απο Web clients, μια αίτηση ενός proxy μπορεί να ικανοποιηθεί απο έναν άλλο proxy κ.ο.κ. Για παράδειγμα είναι δυνατόν να σχηματιστεί μια αλυσίδα στην οποία browsers να χρησιμοποιούν τον proxy μιας επιχείρησης, ο οποίος με την σειρά του να χρησιμοποιεί τον proxy μιας ISP η οποία παρέχει σύνδεση της επιχείρησης στο Διαδίκτυο [15] (βλέπε Σχήμα 2.3).



Σχήμα 2.3: Αλυσίδα απο Web proxies, με σημείο εκκίνησης έναν browser.

Απο τα παραπάνω παραδείγματα, είναι εμφανές οτι η χρήση proxy caching υπόσχεται μεγάλα οφέλη σε όλους εκείνους που συναποτελούν το Web· του χρήστες, τις ISPs και τους **εξυπηρετητές προέλευσης** (origin servers)[12]. Οι χρήστες έχουν πιο γρήγορη πρόσβαση στα αντικείμενα που ζητούν, οι ISPs εξοικονομούν bandwidth στην μεταξύ τους επικοινωνία το οποίο μεταφράζεται σε μείωση κόστους αλλά και εξυπηρέτηση περισσότερων συνδέσεων, και τέλος οι origin servers ξαλαφρώνουν από μεγάλο φόρτο εργασίας. Συνολικά, ευνοείται η επεκτασιμότητα του ίδιου του Διαδικτύου, καθώς μεγάλα κομμάτια του δικτύου αποσυμφορίζονται απο τις caches των proxies που βρίσκονται κοντά στους χρήστες και αναλαμβάνουν να ικανοποιήσουν μεγάλο μέρος των αιτησεών τους [15].

Βέβαια, η παραπάνω εικόνα αποτελεί μια εξιδανίκευση του Web caching μηχανισμού, διότι έρχονται στο προσκήνιο καινούρια ζητήματα τεχνικής και μη φύσεως. Για παράδειγμα, τίθονται ζητήματα ιδιωτικότητας· μπορεί κάποιοι **πάροχοι περιεχομένου** (content providers) να προσφέρουν ιδιωτικό υλικό ή υλικό με **προστασία πνευματικών δικαιωμάτων** (copyright). Σε άλλες περιπώσεις κάποια δεδομένα είναι εκ φύσεως ιδιωτικά, όπως τα στοι-

[12]Origin server ονομάζεται ο Web server στον οποίο εν τέλει βρίσκεται το ζητούμενο Web αντικείμενο, αποτελεί δηλαδή την πηγή του αντικειμένου.

χεία σύνδεσης κάποιου σε έναν λογαριασμό. Επιπλέον, η μη δυνατότητα παρέμβασης στον μηχανισμό συνέπειας του Web caching μπορεί να οδηγήσει σε ανεπιθύμητα αποτελέσματα στην περίπτωση διακίνησης δεδομένων τα οποία πρέπει να είναι αυστηρώς ενημερωμένα. Για όλους τους παραπάνω λόγους, μεγάλο μέρος των διακινούμενων αντικειμένων στο Web δεν μπορεί να αποθηκευτεί και να διαχειριστεί απο το μηχανισμό κρυφής μνήμης αυτού· τα αντικείμενα αυτά χαρακτηρίζονται ως uncacheable ή non cacheable.

Ωστόσο, ο πιο σημαντικός λόγος μείωσης της απόδοσης του Web caching έγκειται στην ίδια την αρχιτεκτονική του Web και συγκεκριμένα στο πρωτόκολλο επικοινωνίας που χρησιμοποιεί, το HTTP. Το HTTP είναι ένα πρωτόκολλο **επιπέδου εφαρμογής** (application layer) σχεδιασμένο στα πλαίσια της **Σουίτας Πρωτοκόλλων του Διαδικτύου** (Internet Protocol Suite) το οποίο προϋποθέτει ως υπόστρωμα ένα αξιόπιστο πρωτόκολλο του **επιπέδου μεταφοράς** (transport layer) [3], συνήθως το **Πρωτόκολλο Ελέγχου Μεταφοράς** (Transmission Control Protocol, TCP)[13] [16]. Το TCP είναι απο μόνο του ένας ανασταλτικός παράγοντας στην ταχύτητα του HTTP και συνεπώς του Web, λόγω της ίδιας της λειτουργικοτητάς του· χαρακτηριστικό παράδειγμα αποτελεί η εγκαθίδρυση μιας TCP σύνδεσης η οποία απαιτεί **χειραψία τριών βημάτων** (three-way handshake) [18]. Εάν, λοιπόν, ληφθεί υπόψιν ότι αυτή η εγκαθίδρυση σύνδεσης πρέπει να γίνεται κάθε φορά, ακόμα και σε ένα hit[14], και επιπλέον οτι συνήθως η ζεύξη μεταξύ ενός Web client και ενός Web proxy είναι αργή, τότε ο proxy καθίσταται ένα **σημείο συμφόρησης του δικτύου** (bottleneck) με επακόλουθη αύξηση αντί για μείωση της καθυστέρησης απόκρισης [15, 19, 20].

Μια απλή και αποδοτική λύση στο παραπάνω πρόβλημα, είναι το **connection caching**. Υπεθυμίζεται οτι η παρούσα **έκδοση** (version) του HTTP - το HTTP 1.1 - υποστηρίζει **persistent TCP** συνδέσεις [3]. Ένας proxy, λοιπόν, μπορεί να διατηρήσει τις συνδέσεις αυτές στις δύο πλευρές του (την πλευρά προς τους clients και την πλευρά προς τους servers), έτσι ώστε να τις επαναχρησιμοποιήσει σε μελλοντικές αιτήσεις. Αυτή ουσιαστικά είναι και η λειτουργία του connection caching, κατα αντιστοιχία με την λειτουργία του caching αντικειμένων. Αυτή η τεχνική σε συνδυασμό με caching δεδομένων μπορεί να οδηγήσει σε μεγάλη μείωση της καθυστέρησης, αλλά και της κατανάλωσης bandwidth [15, 20–23].

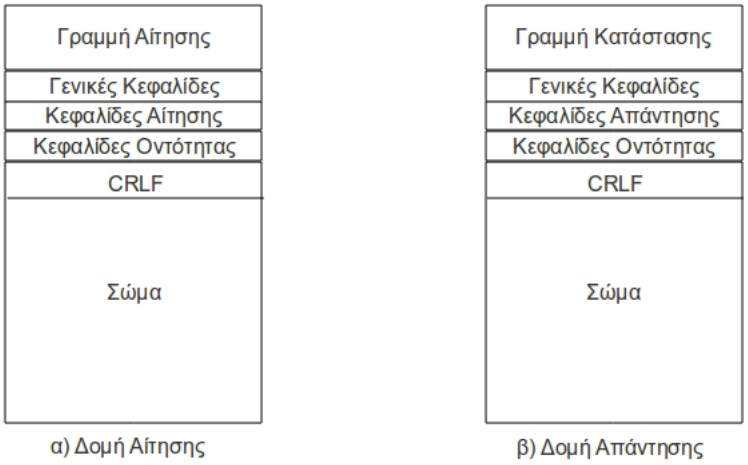
Αυτό που μένει να ξεκαθαριστεί είναι η υποστήριξη του Web caching απο το πρωτόκολλο HTTP, μιας και αυτό είναι το κυρίαρχο πρωτόκολλο επικοινωνίας στο Web. Για να γίνει αυτό, είναι απαραίτητο να περιγραφούν τα βασικά στοιχεία της αριτεκτονικής του.

Η πρώτη ευρέως διαδεδομένη έκδοση του HTTP ήταν η 0.9, η οποία αντικαταστάθηκε απο την HTTP 1.0 [24] και στην συνέχεια απο την HTTP 1.1 [3]. Το HTTP είναι ένα απλό πρωτόκολλο **αίτησης - απάντησης** (request - response) το οποίο χρησιμοποιεί Uniform Resource Locators, URLs [25] για να ονομάσει τα αντικείμενα που ανασύρει. Η επικοινωνία μεταξύ των κομβών γίνεται μέσω HTTP μηνύματων. Τα μηνύματα αυτά συναπαρτίζονται απο τις HTTP **κεφαλίδες** (headers) και το **σώμα** (entity body)· οι κεφαλίδες διαχωρίζονται απο το σώμα μέσω των carriage return, CR και line feed, LF χαρακτήρων. Υπάρχουν δύο τύποι HTTP μηνυμάτων, οι **HTTP αιτήσεις** (HTTP requests) οι οποίες εκπέμπονται απο τους clients και

[13] Βέβαια το HTTP μπορεί να χρησιμοποιήσει και μη αξιόπιστα πρωτόκολλα μεταφοράς, όπως το User Datagram Protocol, UDP [17].

[14] Ακόμα χειρότερα στην περίπτωση ενός miss πρέπει να εγκαθιδρυθεί η TCP σύνδεση εις διπλούν, μεταξύ client proxy και μεταξύ proxy server.

οι **HTTP απαντήσεις** (HTTP responses) οι οποίες παράγονται απο τους servers (βλέπε Σχήμα 2.4). Οι κεφαλίδες στα HTTP μηνύματα είναι κωδικοποιήμενες σε American Standard Code for Information Interchange, ASCII ώστε τα μηνύματα να είναι εύκολα αναγνώσιμα απο τους ανθρώπους [3, 15, 26].



Σχήμα 2.4: Δομή HTTP μηνυμάτων.

Όπως φαίνεται και στο Σχήμα 2.4, μια HTTP αίτηση αποτελείται απο την **γραμμή αίτησης** (request line), τις **γενικές κεφαλίδες** (general headers), τις **κεφαλίδες αίτησης** (request headers) και τις **κεφαλίδες οντότητας** (entity headers), και πιθανά ένα σώμα, το οποίο περιέχει δεδομένα που ο client θέλει να στείλει στον server. Οι γενικές κεφαλίδες μπορούν να χρησιμοποιηθούν και απο τους δύο τύπους μηνυμάτων. Οι κεφαλίδες αίτησεις αφορούν μονάχα τις HTTP αιτήσεις, ενώ οι κεφαλίδες οντότητας περιγράφουν τα δεδομένα που περιέχονται στην αίτηση (αντιστοίχως στην HTTP απάντηση). Η γραμμή αίτησης είναι εκείνη που περιγράφει την ενέργεια που ζητείται απο τον server και αποτελείται απο τα εξής πεδία: μέθοδος, **κομμάτι μονοπατιού** (path) του URL και έκδοση του πρωτοκόλλου. Οι πρότυπες μέθοδοι είναι οι GET, HEAD, OPTIONS, POST, PUT, DELETE και TRACE [3, 15, 26].

Απο την άλλη, μια HTTP απάντηση αποτελείται απο την **γραμμή κατάστασης** (status line), τις γενικές κεφαλίδες, τις **κεφαλίδες απάντησης** (response headers) και τις κεφαλίδες ονότητας, τους CR, LF χαρακτήρες και το σώμα το οποίο (συνήθως) περιέχει το αντικείμενο που ζητήθηκε (σ.σ. απο τον client). Η γραμμή κατάστασης περιέχει την κατάσταση μια απάντησης και περιλαμβάνει τα ακόλουθα πεδία: έκδοση πρωτοκόλλου, αριθμητικό **κωδικό κατάστασης** (status code) και μια εξήγηση του κωδικού. Το HTTP 1.1. παρέχει αυτή τη στιγμή περίπου 40 κωδικούς κατάστασης χωρισμένους σε πέντε κατηγορίες, με πιο κοινό κωδικό εξ΄ αυτών τον *200, OK*, που σημαίνει οτι η αίτηση ικανοποιήθηκε επιτυχώς απο τον Web server [3, 15, 26]..

Βάσει, λοιπόν, των παραπάνω αρχιτεκτονικών χαρακτηριστικών παρέχεται και η κατάλληλη στήριξη στο Web caching[15]. Το πλέον σημαντικό χαρακτηριστικό του HTTP είναι η υποστήριξη **υπο συνθήκη αιτήσεων** (conditional requests). Η λογική των υπο συνθήκη αιτήσεων είναι πολύ απλή και λειτουργεί ως εξής:

[15]Στην βιβλιογραφία μπορεί να συναντηθεί και ο όρος HTTP caching ο οποίος χρησιμοποιείται για να περιγράψει εκείνα ακριβώς τα χαρακτηριστικά του HTTP πρωτοκολλού που υποστηρίζουν το Web caching.

- **Βήμα 1ο**: Ο client στέλνει μια αίτηση στον server η οποία περιλαμβάνει ειδικες συνθήκες στις κεφαλίδες της.

- **Βήμα 2ο**: Ο server εξετάζει τις συνθήκες αυτές και παράγει την κατάλληλη απάντηση.

- **Βήμα 3ο**: Ο client παραλαμβάνει μια απάντηση απο τον server, η οποία περιέχει σώμα εαν οι συνθήκες (σ.σ. της αίτησης) αποτιμήθηκαν ως **αληθείς** (true). Ειδάλλως η απάντηση δεν περιέχει σώμα και ο κωδικός κατάστασης της είναι *304, Not Modified*.

Για να υλοποιηθούν οι ειδικές συνθήκες, πρέπει στις κεφαλίδες των μηνυμάτων να συμπεριληφθεί κατάλληλη **μετα-πληροφορία** (meta-information). Οι κεφαλίδες που χρησιμοποιούνται για να υλοποιήσουν αυτή την μετα-πληροφορία είναι οι *Last-Modified* και *Etag* απο πλευράς απαντήσεων και οι κεφαλίδες *If-Modified-Since* και *If-None-Match* απο πλευράς αιτήσεων. Οι κεφαλίδες *If-Modified-Since* και *If-None-Match* είναι εκείνες που ουσιαστικά υποστηρίζουν τις υπο συνθήκες αιτήσεις. Απο την άλλη οι κεφαλίδες *Last-Modified* και *Etag* χρησιμοποιούνται απο τον server για να ενημερώσουν τον πελάτη εάν το ζητούμενο αντικείμενο έχει αλλάξει ή όχι, καθώς η μεν τιμή της *Last-Modified* κεφαλίδας περιέχει την ημερομηνία και ώρα της τελευταίας τροποποίησης, η δε τιμή της *Etag* κεφαλίδας είναι μια μοναδική τιμή ενός στιγμιοτύπου του αντικειμένου, π.χ. η MD5[16] τιμή του.

Άρα, το παραπάνω σενάριο μπορεί να γίνει πιο συγκεκριμένο τώρα, κάνοντας χρήση των *Last-Modified* και *If-Modified-Since* κεφαλίδων:

- **Βήμα 1ο**: Ο client στέλνει μια αίτηση στον server η οποία περιλαμβάνει την κεφαλίδα *If-Modified-Since*. Η τιμή της κεφαλίδας αυτής προέρχεται απο την τιμή της *Last-Modified* κεφαλίδας του αντικειμένου που βρίσκεται ήδη στην cache.

- **Βήμα 2ο**: Ο server εξετάζει την τιμή της *If-Modified-Since* κεφαλίδας της αίτησης με την τιμή της *Last-Modified* κεφαλίδας του αντικειμένου του και παράγει μια απάντηση που περιέχει το αντικείμενο αυτό, μονάχα εαν παρατηρήσει οτι οι δύο τιμές διαφέρουν[17].

- **Βήμα 3ο**: Ο client παραλαμβάνει μια απάντηση απο τον server, η οποία περιέχει σώμα εαν όντως το αντικείμενο βρέθηκε τροποποιημένο στον server, διαφορετικά παραλαμβάνει μια απάντηση δίχως σώμα και με κωδικό κατάστασης *304, Not Modified*.

Πέραν, λοιπόν, των παραπάνω χαρακτηριστικών, το HTTP παρέχει ένα μεγάλο οπλοστάσιο κεφαλίδων, κωδικών, μεθόδων και κανόνων εν γένει που καθορίζουν το Web caching. Υπάρχουν για παράδειγμα οι **Cache-Control οδηγίες** (directives) που καθορίζουν το **cacheability** ή όχι ενός αντικειμένου, υπάρχουν οι κεφαλίδες **Age** και **Expires** που καθορίζουν έναν ρητό **χρόνο ζωής** (Time to Live, TTL) του αντικειμένου στην cache μετά την παρέλευση του οποίου το αντικείμενο θεωρείται **μη έγκυρο** (invalid) κ.α. [3, 15, 26, 28].

[16]Εννοείται η MD5 B(hash function) [27].

[17]Για την ακρίβεια θα πρέπει να παρατηρήσει *Last-Modified* τιμή στο αντικείμενο του μεγαλύτερη απο την *If-Modified-Since* τιμή της αίτησης.

2.3 Το πρόβλημα της Συνέπειας της Κρυφής Μνήμης στο Παγκόσμιο Ιστό

Έχει ήδη ειπωθεί οτι ο Web caching μηχανισμός δημιουργεί πολυάριθμα αντίγραφα ενός αντικειμένου τα οποία βρίσκονται διασκορπισμένα σε όλο το Διαδικτύο. Εάν ένα αντικείμενο τροποποιηθεί στον origin server, τότε όλα τα αντίγραφα του αντικειμένου αυτού που βρίσκονται σε διάφορες caches γίνονται stale. Το πρόβλημα αυτό είναι γνωστό ως πρόβλημα συνέπειας κρυφής μνήμης (cache consistency problem) και στη συγκεκριμένη περίπτωση ως πρόβλημα συνέπειας κρυφής μνήμης στο Παγκόσμιο Ιστό (Web caching consistency problem). Το πρόβλημα αυτό δεν είναι πρωτόφαντο· δηλαδή δεν συμβαίνει μονάχα στο Web αλλά σε κάθε σύστημα που χρησιμοποιεί caches, όπως για παράδειγμα στην ιεραρχία μνήμης που περιγράφηκε στην ενότητα 2.1. Στη συγκεκριμένη περίπτωση, όμως, απο τη μία η μεγάλη κλίμακα του Web με την αυτονόητη ανάγκη για επεκτασιμότητα και απο την άλλη η χαλαρή διασύνδεση μεταξύ των στοιχείων του[18][19] καθιστούν το πρόβλημα αυτό ξεχωριστό και ιδιαίτερα δύσκολο.

Οι προσεγγίσεις ως προς την επίλυση του προβλήματος αυτού διαφέρουν. Δύο θεμελιώδεις προσεγγίσεις είναι οι ακόλουθες: προσέγγιση **επικύρωσης** (validation) και προσέγγιση **ακύρωσης** (invalidation) [15, 29]. Στην πρώτη προσέγγιση οι proxies αναλαμβάνουν να επαληθεύσουν την **εγκυρότητα** (validity) του αντικειμένου, που έχουν αποθηκευμένο στην cache τους, επικοινωνόντας με το origin server. Αντιθέτως, στην προσέγγιση του invalidation, ο origin server είναι εκείνος που σε περίπτωση τροποποίησης κάποιου αντικειμένου του, αναλαμβάνει να ενημερώσει όλους εκείνους τους proxies που έχουν κάποιο αντίγραφο στις caches τους.

Μια ακόμα κατηγοριοποίηση των μηχανισμών διασφάλισης συνέπειας στις Web caches είναι αυτή, που τους διαχωρίζει σε **σύγχρονους** (synchronous) και **ασύχγρονους** (asynchronous) [15]. Ως σύγχρονος ορίζεται κάθε μηχανισμός ο οποίος ελέγχει την εγκυρότητα ενός αντικειμένου τη στιγμή που ζητείται απο κάποιον client· το αντίθετο, δηλαδή ο ελεγχος της εγκυρότητας ενός αντικειμένου οποιαδήποτε στιγμή ορίζει έναν ασύγχρονο μηχανισμό. Εξυπακούεται οτι ένας invalidation μηχανισμός είναι de facto ασύγχρονος, ενώ ένας validation μηχανισμός έχει την δυνατότητα να είναι είτε σύγχρονος είτε ασύγχρονος, ανάλογα με τη πολιτική που ακολουθείται απο τον proxy.

Τέλος, οι μηχανισμοί διαχωρίζονται σε αυτούς που παρέχουν **ασθενή συνέπεια** (weak consistency) και σε εκείνους που παρέχουν **ισχυρή συνέπεια** (strong consistency) [15, 29, 30]. Αυτή η κατηγοριοποίηση είναι και η πιο σημαντική. Συνήθως, οι validation μηχανισμοί είναι αυτοί που παρέχουν ασθενή συνέπεια, μιας και τα δεδομένα σε αυτούς τους μηχανισμούς ελέγχονται περιοδικά. Στον αντίποδα οι invalidation μηχανισμοί δεν έχουν κάποιο λόγο να αναβάλλουν την ενημέρωση των proxies ύστερα απο την τροποποίηση ενός αντικειμένου στον origin server και γι αυτό το λόγο ανήκουν τυπικά στην κατηγορία των μηχανισμών ισχυρής συνέπειας.

Στη συνέχεια της ενότητας αυτής θα μελετηθούν μερικοί απο τους σημαντικότερους μηχανισμούς που έχουν προταθεί και ανήκουν σε μια απο τις δύο παραπάνω κατηγορίες (σ.σ. ασθενούς ή ισχυρής συνέπειας). Αυτό που πρέπει να γίνει αντιληπτο απο τον αναγνώστη της παρούσας εργασίας είναι πως ο κάθε μηχανισμός έχει τα πλεονεκτηματά του, καθώς και τα αντίστοιχα

[18] Η χαλαρή διασύνδεση των στοιχείων του Web είναι αναγκαίο χαρακτηριστικό, μιας και αποτελεί ένα ΚΣ, και συνεπώς πρέπει να υπακούει στην αρχή της αυτονομίας των κόμβων.

[19] Εδώ ως στοιχεία εννοούνται οι origin servers απο τη μια μεριά και οι clients, proxies απο την άλλη.

μειονεκτήματα. Θα μπορούσε να πει κανείς οτι είναι ουσιαστικά συμπληρωματικοί μηχανισμοί· δηλαδή τα πλεονεκτήματα τους ενός είναι μειονεκτήματα του άλλου και αντιστρόφως. Η αλήθεια είναι οτι οι μηχανισμοί που ευνοούνται απο το Web και υποστηρίζονται πλήρως απο το HTTP είναι αυτοί που χρησιμοποιούν validation λογική και παρέχουν ασθενή συνέπεια. Οι invalidation μηχανισμοί ενδείκνυνται σε επιχειρησιακά και **ιδιόκτητα** (proprietary) δίκτυα [15].

2.3.1 Μηχανισμοί Ασθενούς Συνέπειας

Στα πλαίσια αυτής της υποενότητας θα μελετηθούν μερικοί απο τους πιο σημαντικούς μηχανισμούς ασθενούς συνέπειας. Η **ισορροπία** (tradeoff) που πρέπει να ικανοποιηθεί απο έναν μηχανισμό της κατηγορίας αυτής βρίσκεται μεταξύ του βαθμού συνέπειας και του κόστους καθυστέρησης απόκρισης και συμφόρησης του δικτύου. Ένας μηχανισμός που ελέγχει πιο συχνά την εγκυρότητα ενός αντικειμένου αυξάνει την συμφόρηση του δικτύου, καθώς και την καθυστέρηση απόκρισης, αλλά μειώνει σε μεγάλο βαθμό την πιθανότητα παράδοσης stale αντικειμένου σε κάποιον χρήστη.

Συνήθως οι ασθενείς μηχανισμοί συνέπειας λειτουργούν βάσει μιας TTL λογικής. Δηλαδή είτε δίνεται απο τον origin server ένας ρητός χρόνος ζωής στο αντίγραφο του αντικειμένου που θα τοποθετηθεί στην cache κάποιου proxy, μετά την παρέλευση του οποίου το αντικείμενο θα θεωρηθεί μη έγκυρο, είτε ο ίδιος ο proxy θα υπολογίσει αυτόν τον χρόνο μέσω ενός **ευρετικού** (heuristic) μηχανισμού.

Στην πρώτη περίπτωση ο origin server παράσχει στο αντίγραφο του αντικείμενου μια απο τις *Expires* ή *Max-Age* κεφαλίδες. Ο proxy που λαμβάνει το αντίγραφο αυτό, το αποθηκεύει στην cache του και στη συνέχεια ικανοποιεί όλες τις εισερχόμενες αιτήσεις, εως ότου λήξει ο χρόνος ζωής, όπως αυτός ορίζεται απο τις συγκεκριμένες κεφαλίδες. Στη συνέχεια ο proxy πρέπει να ελέγξει την εγκυρότητα του αντιγράφου, στέλνοντας μια αίτηση στον origin server. Εάν ο origin server είχε παράσχει στο αντίγραφο και την *Last-Modified*[20] κεφαλίδα, ο proxy μπορεί να στείλει μια υπο συνθήκη αίτηση τύπου GET[21], εφοδιάζοντας την με την *If-Modified-Since*[22] κεφαλίδα (IMS GET)[23], διαφορετικά είναι αναγκασμένος να στείλει μια κανονική αίτηση τύπου GET.

Στο σχήμα που μόλις περιγράφηκε μπορεί κάποιος να διακρίνει τα ακόλουθα συμβάντα[24] [31, 33]:

- **Αργή ευστοχία** (slow hit): Στην περίπτωση αυτή ο proxy λαμβάνει μια αίτηση για κάποιο αντικειμένο που βρίσκεται στην cache του και το οποίο θεωρεί μη έγκυρο. Στη συνέχεια προωθεί την αίτηση στον origin server, μονάχα για να διαπιστώσει οτι το αντίγραφο που έχει στην cache του παραμένει έγκυρο.

[20] Υπενθυμίζεται πως η κεφαλίδα αυτή έχει ως τιμή την **χρονοσφραγίδα** (timestamp) (ημερομηνία και ώρα) τελευταίας τροποποίησης του αντικειμένου στον origin server.

[21] Συνήθως, μια αίτηση χαρακτηρίζεται απο την μέθοδο που ορίζει στην γραμμή αίτησης. Επομένως μια αίτηση τύπου GET είναι μια αίτηση που ζητάει δεδομένα απο τον server και καλείται GET request.

[22] Υπενθυμίζεται πως η κεφαλίδα αυτή λαμβάνει ως τιμή, την τιμή της *Last-Modified* κεφαλίδας.

[23] Είναι ο πιο συχνός τύπος υπο συνθήκη αίτησης τύπου GET, όπου το IMS προέρχεται απο τα αρχικά της *If-Modified-Since* κεφαλίδας.

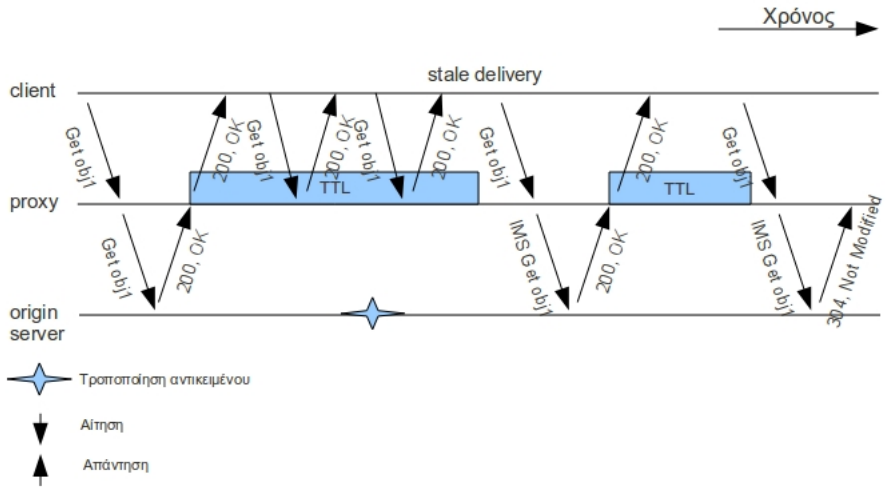
[24] Στη βιβλιογραφία τα συμβάντα αυτά μπορεί να αναφέρονται με διαφορετικούς όρους, όπως π.χ. freshness hit, content miss κ.λπ. [32].

- **Γρήγορη ευστοχία** (fast hit): Σε αντίθεση με το slow hit, ο proxy λαμβάνει μια αίτηση για κάποιο αντικείμενο που βρίσκεται στην cache του και την οποία ικανοποιεί αμέσως, θεωρώντας το έγκυρο. Σε αυτή την περίπτωση, μιας και ο proxy δεν έχει καμία επικοινωνία με τον origin server, ελλοχεύει ο κίνδυνος **παράδοσης έωλου** (stale delivery) αντικειμένου.

- **Αστοχία συνέπειας** (consistency miss): Σε αυτή την περίπτωση ο proxy λαμβάνει μια αίτηση για κάποιο αντικειμένο που βρίσκεται στην cache του και το οποίο θεωρεί μη έγκυρο. Στη συνέχεια προωθεί την αίτηση στον origin server, για να διαπιστώσει οτι όντως το αντικείμενο έχει τροποποιηθεί και συνεπώς πρέπει να αναμεταδοθεί.

- **Αναγκαστική αστοχία** (compulsory miss): Στο compulsory miss ο proxy δεν μπορεί να ικανοποιήσει την αίτηση, μιας και το αντικείμενο δε βρίσκεται στην cache του. Είναι αναγκασμένος, λοιπόν, να προωθήσει την αίτηση στον origin server, ώστε να «κατεβάσει» το ζητούμενο αντικείμενο.

Απο τα παραπάνω συμβάντα το πιο πολύτιμο είναι το fast hit, μιας και οδηγεί σε μεγάλη μείωση του latency, καθώς και της κατανάλωσης bandwidth, ενώ και τα slow hits παρότι δεν είναι τόσο αποδοτικά είναι περισσότερο επιθυμητά απο τα misses. Στο σχήμα 2.5 όπου αναπαρίσταται η λειτουργία ενός μηχανισμού βασισμένου σε TTL, μπορούμε να διακρίνουμε ένα stale delivery συμβάν.



Σχήμα 2.5: Μηχανισμός ασθενούς συνέπειας βασισμένος σε TTL.

Ο πλέον διαδεδομένος, συνάμα εδραιωμένος, μηχανισμός ασθενούς συνέπειας βασισμένος σε TTL είναι ο **Adaptive Time to Live, ATTL**. Ο μηχανισμός αυτός οφείλει την προελευσή του στο πρωτόκολλο Alex [34] και βασίζεται σε μια πολύ απλή λογική:

> *όσο περισσότερο έχει παραμείνει ένα αρχείο (αντίστοιχα ένα αντικείμενο) αμετάβλητο τόσο περισσότερο τείνει να παραμείνει αμετάβλητο στο μέλλον.*

Βάσει αυτού του μηχανισμού, λοιπόν, όταν ένας pxory λαμβάνει ένα αντικείμενο, υπολογίζει το σχετικό του TTL ως ένα κλάσμα του χρόνου που έχει κυλήσει μεταξύ της τελευταίας τροποποιησής του και του χρόνου που απεστάλει απο τον origin server (δηλαδή της τιμής της *Date* κεφαλίδας):

$$TTL = min\{k * (send\_time - last\_modified), threshold\}$$

όπου το k είναι μια σταθερά με τυπικές τιμές 0.1 ή 0.2 και το *threshold*, ένα κατώφλι το οποίο χρησιμοποιείται για να εξασφαλιστεί οτι ακόμα και τα αντικείμενα τα οποία είναι πολύ παλιά θα ελεγχθούν.

Η απόδοση του παραπάνω μηχανισμού είναι αρκετά καλή. Προσομοιώσεις με $k = 0.2$ είχαν ως αποτέλεσμα πολύ χαμηλό stale delivery, της τάξης του 0.8% επι του συνόλου των cache hits ή 0.22% επι του συνόλου των αιτήσεων, ενώ μεγάλο ποσοστό των **αιτήσεων επικύρωσης** (validation requests), της τάξης του 58.5%, συνέβαλαν σε απάντηση με status code *304, Not Modified* [20].

Μια ασύγχρονη προσέγγιση ενός μηχανισμού βασισμένου σε TTL αποτελεί ο μηχανισμός **Piggyback Cache Validation, PCV** [35]. Στόχος της προσέγγισης αυτής είναι η αποφυγή αποστολής στον origin server αιτήσεων επικύρωσης που θα οδηγήσουν σε slow hit, δηλαδή σε απάντηση με status code *304, Not Modified*. Για το λόγο αυτό, είναι αναγκαίο να αποδεσμευτεί ο έλεγχος της εγκυρότητας ενός αντικειμένου απο την λήψη μια αίτησης προς αυτό. Το πρόβλημα που ανακύπτει σε αυτή τη προσέγγιση σχετίζεται με τον προσδιορισμό μιας κατάλληλης χρονικής στιγμής πυροδότησης του ελέγχου. Ο *PCV* μηχανισμός επιλύει το πρόβλημα αυτό με έναν εκλεπτυσμένο τρόπο. Σύμφωνα με τον *PCV*, όποτε ο proxy επικοινωνεί με κάποιον origin server για να στείλει μια *IMS GET* αίτηση στα πλαίσια ελέγχου κάποιου αντικειμένου, εξετάζει εαν διαθέτει άλλα αντικείμενα που να προέρχονται απο τον ίδιο origin server και τα οποία να προσεγγίζουν της λήξη του χρόνου ζωής τους, και στέλνει *HEAD* αιτήσεις για όσα τέτοια αντικείμενα βρεί[25]. Κατ΄ αυτόν τον τρόπο αποφεύγεται η λήψη κάποιου εκ των υπολοίπων αντικείμενων ακόμα και αν έχει τροποποιηθεί. Με την μετα-πληροφορία όμως που επιστρέφεται, ο proxy έχει την δυνατότητα είτε να ακυρώσει το αντικείμενο, είτε να ανανεώσει το χρόνο ζωής του στην cache.

2.3.2 Μηχανισμοί Ισχυρής Συνέπειας

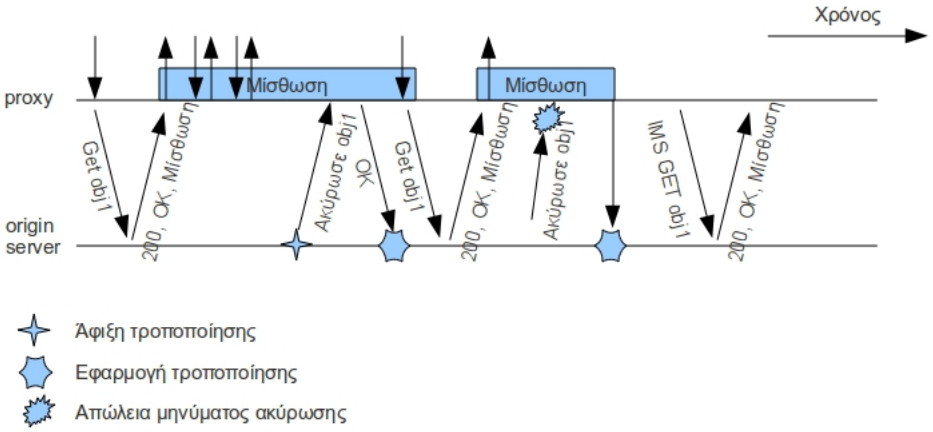
Ένας απλός μηχανισμός ισχυρής συνέπειας είναι η **Επικύρωση Πελάτη** (Client Validation), ο οποίος είναι γνωστός και ως *Polling Every Time* [29]. Σε αυτό τον μηχανισμό ο proxy αντιμετωπίζει τα αντιγράφα αντικειμένων που έχει αποθηκευμένα στην cache ως μη έγκυρα και κάθε φορά που λαμβάνει μια αίτηση προς κάποιο απο αυτά, στέλνει στον origin server μια *IMS GET* αίτηση. Είναι προφανές οτι ο μηχανισμός αυτός μπορεί να οδηγήσει σε μια κατάσταση όπου ο origin server να αναπαράγει πολλά *304, Not Modified* μηνύματα, προκαλώντας ανώφελη συμφόρηση στο δίκτυο μεταξύ του proxy και του origin server.

Πιο σημαντικοί μηχανισμοί της κατηγορίας αυτής είναι οι invalidation μηχανισμοί. Παρότι υπάρχει δυνατότητα υποστηριξής τους απο τις μεθόδους που παρέχει το HTTP, δεν είναι ευ-

[25]Οι *HEAD* αιτήσεις επιστρέφουν μονάχα μετα-πληροφορία σχετική με το αντικείμενο.

ρύτερα αποδεκτοί στο Web λόγω τεράστιων ζητημάτων που θέτουν επι τάπητος αλλά και επειδή, προς το παρόν, δεν υπάρχει κάποιο ολοκληρωμένο πρωτόκολλο επικοινωνίας που να επιτρέπει κάτι τέτοιο. Πέραν αυτού, οι μηχανισμοί αυτοί καλούνται να λύσουν δύο σημαντικά προβλήματα: το **πρόβλημα της λίστας πελατών** (client list problem) και το **δίλημμα των καθυστερημένων ενημέρωσεων** (delayed updates dilemma) [15].

Ένας invalidation μηχανισμός που παρέχει ισχυρή συνέπεια, ενώ ταυτόχρονα αντιμετωπίζει με αποδοτικό τρόπο όλα τα παραπάνω προβλήματα είναι αυτός που κάνει χρήση της έννοιας της **μίσθωσης** (lease) [36]. Σύμφωνα με τον μηχανισμό αυτό, ένας proxy που κατέχει αντίγραφο κάποιου αντικειμένου του origin server, ανήκει στην client list (σ.σ. που αφορά το αντικείμενο αυτό) για όσο διαρκεί η μίσθωση. Εάν «φτάσει» μια ενημέρωση προς το αντικείμενο αυτό, τότε ο origin server είναι υποχρεωμένος στα χρονικά περιθώρια της μίσθωσης να στείλει μήνυμα ακύρωσης σε όλους τους proxies που κατέχουν αντίγραφο. Εάν βεβαιωθεί οτι όλοι proxies έλαβαν το μήνυμα ακύρωσης, τότε μπορεί να τροποποιήσει το αντικείμενο του. Εάν δεν λάβει επιβεβαίωση λήψης του μηνύματος ακύρωσης απο όλους του proxies που ανήκουν στην client list, τότε είναι υποχρεωμένος να περιμένει μέχρι την λήξη της μίσθωσης και όταν αυτή επέλθει, να εφαρμόσει την τροποποίηση.



Σχήμα 2.6: Invalidation μηχανισμός ισχυρούς συνέπειας βασισμένος σε μίθωση.

Εφόσον λήξει μια μίσθωση η client list καθαρίζεται. Συνεπώς, στην πρώτη αίτηση που θα λάβει ένας proxy μετά την λήξη της μίσθωσης είναι αναγκασμένος να στείλει μήνυμα ελέγχου εγκυρότητας προς τον origin server. Επομένως, ο μηχανισμός αυτός είναι συνδυασμός invalidation κατα τη διάρκεια μιας μίσθωσης και validation μόλις η μίσθωση λήξει. Όσον αφορά την επιλογή της διάρκεια της μίσθωσης, μια αποδοτική λύση είναι η εφαρμογή μιας ευρετικής παρόμοιας με αυτή του $ATTL$ [37]. Στο σχήμα 2.6 παρουσιάζεται η λειτουργία του μηχανισμού που μόλις περιγράφηκε.

Τέλος, είναι σημαντικό να ειπωθεί στο σημείο αυτό πως αν και έχει επικρατήσει η άποψη οτι οι μηχανισμοί που προσφέρουν ισχυρή συνέπεια και συγκεκριμένα οι invalidation μηχανισμοί είναι απαγορευτικοί απο το Web [38], διότι μειώνουν την επεκτασιμοτητά του, υπάρχουν εργασίες που υποστηρίζουν το αντίθετο [39].

2.4 Σύνοψη

Συνοψίζοντας, η ανάγκη για γρήγορη και απεριόριστη μνήμη οδήγησε στην έννοια της ιεραρχίας μνήμης όπου γίνεται αξιοποίηση της αρχής της τοπικότητας. Καίριο ρόλο σε αυτή την ιεραρχία παίζει η κρυφή μνήμη η οποία ικανοποιεί το κομμάτι εκείνο της ανάγκης που σχετίζεται με την ταχύτητα.

Η ιδέα της κρυφής μνήμης, όμως, δεν συναντάται μονάχα στην ιεραρχία μνήμης των προσωπικών υπολογιστών. Η ιδέα αυτή διαδραματίζει ιδιαίτερο ρόλο στο Web και συγκεκριμένα στον Web caching μηχανισμό, ο οποίος είναι εκ των σημαντικότερων στη διασφάλιση της επεκτασιμότητας του Web. Το πρόβλημα που ανακύπτει απο την χρήση του Web caching είναι γνωστό ως Web caching consistency problem και συνίσταται στη διασφάλιση χρήσης αντιγράφων απο τις Web caches τα οποία είναι συνεπή με τα αντίστοιχα αντικείμενα των origin servers. Το πρόβλημα αυτό παρουσιάζει ιδιαίτερο ενδιαφέρον απο τη μεριά των cache servers, δηλαδή των proxies εκείνων που βρίσκονται ανάμεσα στους clients και τους origin servers και αναλαμβάνουν τη διάφανη διαχείριση των αντιγράφων.

Αυτή τη στιγμή υπάρχουν διάφοροι μηχανισμοί οι οποίοι προσπαθούν να αντιμετωπίσουν το πρόβλημα αυτό. Οι μηχανισμοί αυτοί διαχωρίζονται, κυρίως, σε μηχανισμούς ασθενούς και ισχυρής συνέπειας, με τους πρώτους να υποστηρίζονται πλήρως απο το HTTP, καθώς επίσης να προτιμούνται περισσότερο για λόγους επεκτασιμότητας του Web. Σημαντικοί μηχανισμοί της πρώτης κατηγορίας είναι ο *ATTL* και ο *PCV*, ενώ της δεύτερης είναι ο *Polling Every Time* και ο *Server Invalidation με χρήση leases*.

3. ΘΕΩΡΙΑ ΒΕΛΤΙΣΤΗΣ ΠΑΥΣΗΣ

Στα πλαίσια αυτού του κεφαλαίου θα γίνει η απαραίτητη εισαγωγή στην θεωρία της Βέλτιστης Παύσης. Θα δοθεί το ιστορικό πλαίσιο μέσα στο οποίο αναπτύχθηκε, καθώς επίσης θα παρουσιαστούν εκείνα τα χαρακτηριστικά προβλήματα που συνέβαλαν στην αναπτυξή της. Στόχος του κεφαλαίου αυτού είναι να παρουσιαστεί το απαραίτητο θεωρητικό υπόβαθρο για την μετέπειτα κατανόηση του μηχανισμού διασφάλισης συνέπειας στο Web caching, του οποίου η παρουσίαση θα γίνει στο αμέσως επόμενο κεφάλαιο. Πιο συγκεκριμένα η διάρθωση του κεφαλαίου είναι η ακόλουθη:

- **Ενότητα 3.1**: Η ενότητα 3.1 κάνει μια σύντομη αναφορά στο ιστορικό πλαίσιο μέσα στο οποίο η θεωρία της Βέλτιστης Πάυσης θεμελιώθηκε και αναπτύχθηκε. Στη συνέχεια παρουσιάζει την αξιωματική της θεμελίωση, και περιγράφει παραδείγματα μερικών εκ των σημαντικότερων προβλημάτων που οδήγησαν στη θεμελίωση και αναπτυξή της.
- **Ενότητα 3.2**: Η ενότητα 3.2 προσπαθεί να φέρει σε επαφή τον αναγνώστη με σημαντικές έννοιες της θεωρίας Βέλτιστης Πάυσης, και κυρίως να παρουσιάσει τις πιο σημαντικές μεθόδους που χρησιμοποιούνται για την επίλυση των προβλημάτων. Επιπλέον, δίνει μια λεπτομερή περιγραφή και λύση ενός προβλήματος της θεωρίας αυτής το οποίο θεωρείται απο τα πλέον σημαντικά με πλείστες εφαρμογές και επεκτάσεις.
- **Ενότητα 3.3**: Η ενότητα 3.3 ασχολείται με μια ειδική κατηγορία προβλημάτων και παρουσιάζει τον αλγόριθμο *Sum the Odds to One and Stop*, που αυτή τη στιγμή αποτελεί τον (state of the art) αλγόριθμο για την κατηγορία αυτή.
- **Ενότητα 3.4**: Η ενότητα 3.4 κάνει μια σύνοψη όλων όσων παρουσιάστηκαν στις προηγούμενες ενότητες.

3.1 Τα βασικά της θεωρίας Βέλτιστης Παύσης

Αντικείμενο της θεωρίας Βέλτιστης Παύσης είναι η επιλογή κατάλληλης χρονικής στιγμής για την εκτέλεση μιας δεδομένης ενέργειας. Η επιλογή αυτή βασίζεται στην ακολουθιακή παρατήρηση **τυχαίων μεταβλητών, τμ** (random variables, rv) και στόχος της είναι η μεγιστοποίηση της αναμενόμενης **ανταμοιβής** (payoff) ή η ελαχιστοποίηση του αναμενόμενου **κόστους**. Τέτοιου τύπου προβλήματα συναντώνται στην περιοχή της **Στατιστικής**, όπου η δεδομένη ενέργεια μπορεί να είναι ο έλεγχος μιας υπόθεσης ή η εκτίμηση μιας παραμέτρου, καθώς και στην περιοχή της **Επιχειρησιακής Έρευνας** (Operations Research), όπου η δεδομένη ενέργεια μπορεί να είναι η αντικατάσταση μιας μηχανής, η πρόσληψη μιας γραμματέως, η αναδιοργάνωση του αποθέματος, κ.λπ. Σε αυτό το κεφάλαιο, θα παρουσιαστεί η μαθηματική ερμηνεία του προβλήματος και θα δοθεί ένας αριθμός απο παραδείγματα εφαρμογής [4].

Σύμφωνα με τον Thomas S. Ferguson [4] το πρώτο βήμα για τον καθορισμό των παραπάνω προβλημάτων ως προβλήματα ακολουθιακής ανάλυσης στατιστικών παρατηρήσεων έγινε με την εργασία του Wald, *Sequential Tests of Statistical Hypotheses* [40], καθώς και τα βιβλία,

Sequential Analysis [41] και *Statistical Decision Functions* [42]. Στη συνέχεια οι Arrow, Blackwell και Girshick μεταχειρίστηκαν τα προβλήματα αυτά σε μια Bayesian προοπτική [43], ενώ η γενίκευση της ακολουθιακής ανάλυσης σε προβλήματα δίχως στατιστική δομή έγινε απο τον Snell το 1952 [44]. Ακολούθησε η δεκαετία του 1960 κατα την οποία οι εργασίες των Chow και Robbins έδωσαν μεγάλη ώθηση στο αντικείμενο της θεωρίας [45, 46]. Μια σύνοψη όλης αυτής της ανάπτυξης δίνεται στο βιβλίο, *Great Expectations: The Theory of Optimal Stopping*, των Chow, Robbins και Siegmund [47].

Τα προβλήματα με τα οποία ασχολείται η θεωρία Βέλτιστης Παύσης ονομάζονται προβλήματα εύρεσης **κανόνα παύσης** (stopping rule). Τα προβλήματα αυτά ορίζονται απο τα εξής δύο αντικείμενα [4]:

- μια ακολουθία τμ, $X_1, X_2, \ldots$, των οποίων η **κοινή κατανομή** (joint distribution) θεωρείται γνωστή, και
- απο μια ακολουθία πραγματικών **συναρτήσεων ανταμοιβής** (reward functions), $y_0, y_1(x_1), y_2(x_1, x_2),\ ..., y_\infty(x_1, x_2, \ldots)$.

Βάσει, λοιπόν, του παραπάνω ορισμού ο κανόνας παύσης μπορεί να διατυπωθεί απλούστερα ως εξής:

1. μπορείς να παρατηρήσεις τις $X_1, X_2, \ldots$, για όσο διάστημα θες,
2. σε κάθε βήμα $n = 1, 2, \ldots$, έχοντας παρατηρήσει $X_1 = x_1, X_2 = x_2, \ldots, x_n = x_n$, μπορείς είτε να σταματήσεις και να λάβεις ως ανταμοιβή την τιμή της συνάρτησης $y_n(x_1, x_2, \ldots, x_n)$, είτε να συνεχίσεις και να παρατηρήσεις την X_{n+1},
3. εάν επιλέξεις να σταματήσεις δίχως να κάνεις παρατηρήσεις, τότε θα λάβεις την σταθερή τιμή y_0,
4. εάν δεν σταματήσεις ποτέ, τότε θα λάβεις $y_\infty(x_1, x_2, \ldots)$.

Στόχος είναι η επιλογή εκείνου του χρόνου παύσης που θα μεγιστοποιήσει την αναμενόμενη ανταμοιβή.

Εάν στην παραπάνω διαδικασία οι αποφάσεις είναι **τυχαιοκρατικές** (randomized), τότε ο κανόνας παύσης ονομάζεται **τυχαιοκρατικός κανόνας παύσης** (randomized stopping rule) και συμβολίζεται ως:

$$\varphi = (\varphi_0, \varphi_1(x_1), \varphi_2(x_1, x_2), \ldots),$$

όπου για κάθε n και $x_1, x_2, \ldots, x_n$, ισχύει $0 \leq \varphi_n(x_1, x_2, \ldots, x_n) \leq 1$. Τυχαιοκρατική απόφαση σημαίνει οτι δεδομένης μια **φάσης** (stage) n, όπου έχουν παρατηρηθεί οι $X_1, X_2, \ldots, X_n$, η απόφαση για την επιλογή παύσης ή όχι γίνεται με πιθανότητα $\varphi_n(x_1, x_2, \ldots, x_n)$ η οποία μπορεί να εξαρτάται απο τις παρατηρήσεις αυτές.

Ως εκ τούτου, η τιμής της φ_0 εκφράζει την πιθανότητα να ενεργοποιηθεί ο κανόνας παύσης χωρίς την λήψη παρατηρήσεων, ενώ η τιμή της $\varphi_1(x_1)$ εκφράζει την πιθανότητα να ενεργοποιηθεί

ο κανόνας παύσης δεδομένου οτι ελήφθη η πρώτη παρατήρηση και αποτιμήθηκε με $X_1 = x_1$, κ.ο.κ. Με βάση τα παραπάνω, δηλαδή τον κανόνα παύσης φ και την ακολουθία παρατηρήσεων, $X_1, X_2, \ldots$, μπορεί πλέον να καθοριστεί ο τυχαίος χρόνος παύσης N, όπου $0 \leq N \leq \infty$, με $N = \infty$ να δηλώνει την ανυπαρξία ενός τέτοιου χρόνου.

Η **συνάρτηση μάζας πιθανότητας** (probability mass function, pmf) του N δεδομένου των $X_1 = x_1, X_2 = x_2, \ldots$, συμβολίζεται ως $\psi = (\psi_0, \psi_1, \psi_2, \ldots, \psi_\infty)$, όπου:

$$\begin{aligned} \psi_n(x_1, x_2, \ldots, x_n) &= P(N = n | X_1 = x_1, X_2 = x_2, \ldots, x_n = x_n), \textit{για } n = 0, 1, 2, \ldots \\ \psi_\infty(x_1, x_2, \ldots) &= P(N = \infty | X_1 = x_1, X_2 = x_2, \ldots) \end{aligned}$$

Επιπλέον, προκύπτει:

$$\begin{aligned} \psi_0 &= \varphi_0 \\ \psi_1(x_1) &= (1 - \varphi_0)\varphi_1(x_1) \\ &\vdots \\ \psi_n(x_1, \ldots, x_n) &= [\textstyle\prod_1^{n-1}(1 - \varphi_j(x_1, \ldots, x_j))]\varphi_n(x_1, \ldots, x_n) \\ &\vdots \\ \psi_\infty(x_1, x_2, \ldots) &= 1 - \textstyle\sum_0^\infty \psi_j(x_1, \ldots, x_j) \end{aligned}$$

Όπου το $\psi_\infty(x_1, x_2, \ldots)$ αναπαριστά την πιθανότητα να μην ενεργοποιηθεί ποτέ ο κανόνας παύσης δεδομένου του συνόλου των παρατηρήσεων.

Το πρόβλημα πλέον συνίσταται στην επιλογή του κανόνα παύσης φ που θα μεγιστοποιεί την αναμενόμενη απόδοση, $V(\varphi)$, ο οποίος ορίζεται ως:

$$\begin{aligned} V(\varphi) &= E_{y_N}(X_1, \ldots, X_N) \\ &= E \textstyle\sum_0^\infty \psi_j(X_1, \ldots, X_j) y_j(X_1, \ldots, X_j) \end{aligned}$$

Εν τέλει, κάνοντας χρήση του τυχαίου χρόνου παύσης N, ο κανόνας παύσης φ μπορεί να γραφεί ως εξής:

$$\varphi_n(X_1, \ldots, X_n) = P(N = n | N \geq n, X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n), \text{ για } n = 0, 1, 2, ...$$

Ο παραπάνω ορισμός διατυπώθηκε κάνοντας χρήση των εννοιών της απόδοσης και της ανταμοιβής. Ανάλογα με τη δομή του προβλήματος, μπορεί να είναι καταλληλότερη η χρήση εννοιών όπως κόστος ή ζημιά. Σε αυτή την περίπτωση η ανάλυση του προβλήματος παύσης δεν αλλάζει ιδιαίτερα· απλά ο κάνονας παύσης σχετίζεται, πλέον, με την επιλογή της χρονικής στιγμής που θα ελαχιστοποιήσει την τιμή $V(\varphi)$.

Επιπλέον, σε κάποιες εφαρμογές είναι πιο ρεαλιστική η περιγραφή της **ακολουθίας ανταμοιβής** (reward sequence) μέσω τμ $Y_0, Y_1, ..., Y_\infty$ των οποίων η κοινή κατανομή είναι γνωστή. Και σε αυτή την περίπτωση δεν υφίσταται σημαντική αλλαγή στην δομή του κανόνα παύσης, ούτε κάποια απώλεια στην γενικοτητά του. Η ακολουθία ανταμοιβής αποτελείται απο τις συναρτήσεις $y_n(x_1, x_2, \ldots, x_n)$ για κάθε $n = 0, 1, ..., \infty$, όπου:

$$y_n(x_1, x_2, \ldots, x_n) = E(Y_n | X_1 = x_1, \ldots, X_n = x_n).$$

Κάθε κανόνας παύσης φ με ακλουθία ανταμοιβής $Y_0, Y_1, ..., Y_\infty$ θα έδινε την ίδια αναμενόμενη απόδοση για την ακολουθία $y_0, y_1, ..., y_\infty$.

Απο τα πιο σημαντικά παραδείγματα χρήσης τέτοιων κανόνων αποτελεί το **πρόβλημα της πώλησης σπιτιού** (house selling problem), όπου προσφορές καταφθάνουν σε καθημερινή βάση και στόχος είναι η πώληση στη μεγαλύτερη προσφορά. Ο πωλητής του σπιτού παρατηρεί τις προσφορές δίχως να έχει κάποια γνώση για τις τιμές τους παρα μόνο όταν υποβληθούν. Η μόνη γνώση που έχει είναι η διαίσθηση οτι οι προσφορές είναι ανεξάρτητες και διέπονται απο μια κοινή κατανομή. Σε κάθε προσφορά έχει δύο επιλογές: 1) να επιλέξει την προσφορά και η αγοραπωλησία να τελειώσει, 2) να την απορρίψει και να περιμένει την επόμενη προσφορά πληρώνοντας ένα σταθερό κόστος c. Άρα το δίλημμα που καλείται να επιλύσει συνίσταται στην επιλογή εκείνης της προσφοράς που θα ναι μεν μεγαλύτερη απο τις προηγούμενες, θα αντισταθμίζει δε ικανοποιητικά τα αντίστοιχα κόστη παρατήρησης.

Ένα ακόμα σημαντικό παράδειγμα είναι η μεγιστοποίηση του **μέσου όρου** (average). Σε αυτή τη περίπτωση παρατηρείται η επαναλαμβανόμενη ρίψη ενός νομίσματος. Στόχος του παρατηρητή είναι η επιλογή του χρόνου παύσης που θα μεγιστοποιήσει τον μέσο όρο των παρατηρούμενων κεφαλών. Εάν, λοιπόν, στην πρώτη ρίψη έρθει κεφαλή, τότε πρέπει να σταματήσει μιας και η απόδοση θα είναι ένα, διαφορετικά να συνεχίσει την επαναλαμβανόμενη ρίψη του νομίσματος. Απο την άλλη, ο **ισχυρός κανόνας των μεγάλων αριθμών** (strong law of large numbers)[1] [48] έχει δείξει οτι ο μέσος όρος των κεφαλών συγκλίνει **σχεδόν σίγουρα** (almost surely, as) στο 1/2, και επομένως δεν θα πρέπει ποτέ κάποιος να σταματήσει την ρίψη του νομίσματος ενόσω ο μέσος όρος των κεφαλών είναι μικρότερος ή ίσος με 1/2. Κι σε αυτό το πρόβλημα το δίλημμα συνίσταται στην επιλογή του χρόνου παύσης που θα μεγιστοποιήσει την απόδοση.

Άλλα σημαντικά παραδείγματα χρήσης κανόνων παύσης αποτελούν: τα πρόβληματα εκτίμησης παραμέτρων βασισμένα σε ακολουθιακή και bayesian ανάλυση, τα one-armed bandit προβλήματα, κ.λπ.

3.2 Εφαρμογές

Σε πολλά προβλήματα ορίζεται ένα άνω φράγμα στον αριθμό των παρατηρήσεων που μπορούν να γίνουν. Συνεπώς ο αντίστοιχος κανόνας παύσης ορίζεται ως κανόνας παύσης με **πεπερασμένο ορίζοντα** T, που σημαίνει οτι πρέπει να ενεργοποιηθεί αναγκαστικά μετά την παρατήρηση των $X_1, X_2, \ldots, X_T$, δηλαδή όταν φτάσει στον ορίζοντα. Τα προβλήματα πεπερασμένου ορίζοντα λύνονται βάσει της μεθόδου της **προς τα πίσω επαγωγής** (backward induction). Η μέθοδος αυτή είναι αρκετά απλή. Ουσιαστικά βρίσκει όλους του βέλτιστους κανόνες παύσης ξεκινόντας απο τον ορίζονται T και προχωρόντας προς τα πίσω, δηλαδή τις φάσεις T-1, T-2, κ.ο.κ. Αρχικά ισχύει $V_T^{(T)}(x_1, x_2, \ldots, x_T) = y_T(x_1, x_2, \ldots, x_T)$ και επαγωγικά προς τα πίσω απο $j = T - 1$ εώς $j = 0$:

[1] Γνωστός και ως Kolmogorov's strong law.

$$V_j^{(T)}(x_1, x_2, \ldots, x_j) = max\{y_j(x_1, x_2, \ldots, x_j), E(V_{j+1}^{(T)}(x_1, \ldots, x_j, X_{j+1})|X_1 = x_1, \ldots, x_j = x_j)\}$$

Επαγωγικά, το $V_j^{(T)}(x_1, x_2, \ldots, x_j)$ αναπαριστά την μέγιστη απόδοση που κάποιος μπορεί να αναμένει ξεκινώντας απο την φάση j και έχοντας παρατηρήσει $X_1 = x_1, \ldots, x_j = x_j$. Στη φάση j συγκρίνεται η απόδοση της παύσης, δηλαδή η τιμή της $y_j(x_1, x_2, \ldots, x_j)$, με την αναμενόμη απόδοση της συνέχισης και χρήσης του βέλτιστου κανόνα για τις φάσεις $j+1$ εώς T, που στην φάση j είναι $E(V_{j+1}^{(T)}(x_1, \ldots, x_j, X_{j+1})|X_1 = x_1, \ldots, x_j = x_j)$. Συνεπώς, η βέλτιστη απόδοση είναι συνδεδεμένη με μια απο αυτές τις δύο τιμές, και είναι βέλτιστο να γίνει παύση στην φάση j εάν η μέγιστη τιμή του $V_j^{(T)}(x_1, x_2, \ldots, x_j)$ είναι ίση με $y_j(x_1, x_2, \ldots, x_j)$, διαφορετικά να συνεχίσει.

Δυστυχώς η παραπάνω μέθοδος δεν είναι εύκολα εφαρμόσιμη παρα μόνο εάν το πρόβλημα έχει **Markov δομή** (Markov structure), δηλαδή οι αποδόσεις αποτελούν συναρτήσεις μιας **Markov αλυσίδας** (Markov chain). Τα προβλήματα αυτά ονομάζονται Markov προβλήματα παύσης [49]. Τα υπόλοιπα προβλήματα είναι δυσεπίλυτα βάσει της μεθόδου αυτής, λόγω του μεγάλου όγκου των παρατηρήσεων που πρέπει να γίνουν. Μία πρώτη μέθοδος για την αντιμετώπιση αυτού του προβλήματος είναι η επιβολή πεπερασμένου ορίζοντα σε αυτό, γνωστή ως **μέθοδος περικοπής** (truncation method). Βέβαια, και πάλι το πρόβλημα δεν λύνεται, διότι εάν για παράδειγμα έχουμε T = 20 και κάθε x_i μπορεί να πάρει εώς 10 τιμές, τότε ο υπολογισμός της $V_j^{(T)}(x_1, x_2, \ldots, x_j)$ γίνεται πολύ μεγάλος ακόμα και για τους σημερινούς υπολογιστές.

Μια απλή και υπο συνθήκη βέλτιστη λύση είναι η χρήση *k-stage look ahead, k-sla* κανόνων. Μάλιστα, ο *1-stage look ahead, 1-sla* κανόνας είναι αρκετά καλός και υπο την συνθήκη μονοτονίας , είναι βέλτιστος για προβλήματα πεπερασμένου ορίζοντα[2]. Ο *k-sla* κανόνας ορίζεται ως εξής:

$$N_k = min\{n \geq 0 : Y_n \geq V_n^{(n+k)}\} = min\{n \geq 0 : Y_n \geq E(V_{n+1}^{(n+k)}|F_n)\}$$

Άρα, ο *k-sla* κανόνας είναι ο κανόνας εκείνος που σταματά στην φάση n, εάν ο βέλτιστος κανόνας για τις επόμενες k φάσεις σταμάτα, ειδάλλως συνεχίζει. Με απλά λόγια, ο *k-sla* κανόνας αποφασίζει την επόμενη κινησή του κοιτάζοντας μονάχα k φάσεις μπροστά.

Ο πιο απλός κανόνας είναι ο *1-sla* ο οποίος ορίζεται ακολούθως:

$$N_1 = min\{n \geq 0 : Y_n \geq V_n^{(n+1)}\} = min\{n \geq 0 : Y_n \geq E(Y_{n+1}|F_n)\}$$

Ο N_1, λοιπόν, σταματά στην πρώτη φάση n για την οποία η απόδοση είναι τουλάχιστον τόσο μεγάλη όσο η αναμενόμενη απόδοση συνέχισης μιας φάσης και ύστερα παύσης.

Ο N_1 κανόνας, ο επονομαζόμενος και μυωπικός κανόνας, είναι κάτω απο ορισμένες συνθήκες βέλτιστος. Βασική συνθήκη είναι η μονοτονία του προβλήματος η οποία ορίζεται ως εξής:

> *Έστω οτι το A_n συμβολίζει το **συμβάν** (event) $\{Y_n \geq E(Y_{n+1}|F_n)\}$. Λέμε οτι το πρόβλημα εύρεσης κανόνα διακοπής είναι μονότονο εάν ισχύει $A_0 \subset A_1 \subset A_2 ...$ as.*

[2]Για να είναι βέλτιστος αυτός ο κανόνας σε προβλήματα **μη πεπερασμένου** (infinite) ορίζοντα, χρειάζονται περισσότερες συνθήκες.

Απο τον ορισμό προκύπτει οτι το σύνολο A_n είναι το σύνολο στο οποίο ο *1-sla* καλεί για παύση στην φάση n, και πως η συνθήκη $A_n \subset A_{n+1}$ συνεπάγεται την ενεργοποίηση του κανόνα παύσης στην φάση $n+1$ άσχετα απο την τιμή του X_{n+1} as.

Στη συνέχεια ακολουθεί ένα απο τα σημαντικότερα προβλήματα εύρεσης κανόνων διακοπής με πλείστες εφαρμογές στην επιχειρησιακή έρευνα.

3.2.1 Το Πρόβλημα Επιλογής της Γραμματέως

Το **πρόβλημα της γραμματέως** (secretary problem) είναι ένα απο τα κλασσικότερα προβλήματα της θεωρίας αυτής, το οποίο μπορει να συναντηθεί στην βιβλιογραφία και ως **πρόβλημα επιλογής συζύγου** (marriage problem) ή επιλογής του μεγαλύτερου αριθμού απο ένα άγνωστο σύνολο αριθμών (Googol) [50]. Ανήκει στα προβλήματα πεπερασμένου ορίζοντα[3], καθώς επίσης στην κατηγοριά των **no-information** προβλημάτων. Ως no-information προβλήματα νοούνται τα προβλήματα εκείνα στα οποία η κατανομή των τμ $X's$ είναι πλήρως άγνωστη και μόνο σχετική **κατάταξη** (rank) μπορεί να αποδοθεί στα αντικείμενα που παρατηρούνται. Στον αντίποδα βρίσκονται τα **full-information** προβλήματα στα οποία τα $X's$ είναι **ομοιόμορφα ανεξάρτητα κατανεμημένα** (identically independently distributed, iid)[4][5] βάσει μιας γνωστής κατανομής.

Το πρόβλημα της γραμματέως έχει απασχολήσει πολλούς επιστήμονες της Στατιστικής και των Πιθανοτήτων. Αρκετές επεκτάσεις έχουν προταθεί και μελετηθεί γι αυτό το πρόβλημα, μερικές εκ των οποίων είναι η αβέβαιη πρόσληψη ή διαθεσιμότητα του υποψηφίου, η επανάκληση κάποιου που είχε απορριφθεί κ.λπ [51]. Στα πλαίσια της ενότητας αυτής θα περιγραφεί το **κλασσικό πρόβλημα της γραμματέως** (classical secretary problem, CSP), το οποίο ορίζεται ως εξής:

1. Υπάρχει μόνο μια διαθέσιμη θέση γραμματέως.
2. Υπάρχουν n **αιτούντες** (applicants) για την θέση, όπου το n είναι γνωστό.
3. Είναι δυνατή η γραμμική ταξινόμηση των αιτούντων από τον καλύτερο στον χειρότερο χωρίς να υπάρχουν ισοβαθμίες.
4. Οι αιτούντες περνούν από συνέντευξη διαδοχικά με τυχαία διάταξη και όλες οι $n!$ διατάξεις είναι ισοπίθανες.
5. Μετά από την συνέντευξη κάποιου αιτούντος, είτε γίνεται πρόσληψη του και το πρόβλημα απόφασης τελειώνει, είτε γίνεται απόρριψη και καλείται για συνέντευξη ο επόμενος (εάν υπάρχει).
6. Η απόφαση πρόσληψης ή απόρριψης ενός αιτούντος θα πρέπει να βασίζεται μονάχα στη σχετική διάταξη των υποψηφίων που έχουν εξεταστεί μέχρι εκείνη την στιγμή.

[3] Για την επίλυση του προβλήματος της γραμματέως σε μη πεπερασμένο ορίζοντα υπάρχουν οι εργασίες [52,53]

[4] Με πιο απλά λόγια τα $X's$ είναι πραγματικές τιμές των αντικειμένων που παρατηρούνται, οι οποίες είναι iid σε μια κατανομή, συνήθως (χωρίς απώλεια της γενικότητας) την ομοιόμορφη κατανομή στο [0,1].

[5] Άλλη κατηγορία προβλημάτων, ενδιάμεση των δύο που περιγράφηκαν πιο πάνω, είναι τα partial-information προβλήματα.

7. Δεν μπορεί να γίνει ανάκληση της απόρριψης ενός αιτούντος.

8. Στόχος είναι η επιλογή του καλύτερου αιτούντος· που σημαίνει οτι σε περίπτωση νίκης η απόδοση είναι 1, διαφορετικά 0.

Οι παρατηρήσεις στο παραπάνω πρόβλημα $X_1, X_2, \ldots, X_n$, αναπαριστούν τις σχετικές κατατάξεις των αιτούντων, με X_j να αντιπροσωπεύει την κατάταξη του j αιτούντος και την κατάταξη 1 (δηλαδή τιμή 1 για κάποιο απο τα $X's$) να αποτελεί την καλύτερη κατάταξη.

Ένας σχετικά καλός αιτών, δηλαδή ένας αιτών ο οποίος έχει σχετική κατάταξη 1, καλείται **υποψήφιος** (candidate). Για παράδειγμα, εάν ο j^{th} αιτών είναι υποψήφιος, τότε ισχύει $X_j = 1$. Εάν επιλεγεί ένας υποψήφιος στην φάση j, τότε η πιθανότητα νίκης δίνεται ως εξής:

$$\begin{aligned} P(\text{νίκης}) &= P(\text{καλύτερος συνολικά} \mid \text{υποψήφιος στους πρώτους j αιτούντες}) \\ &= \frac{P(\text{καλύτερος συνολικά, υποψήφιος στους πρώτους j αιτούντες})}{P(\text{υποψήφιος στους πρώτους j αιτούντες})} \\ &= \frac{P(\text{καλύτερος συνολικά})}{P(\text{υποψήφιος στους πρώτους j αιτούντες})} \\ &= \frac{1/n}{1/j} \\ &= \frac{j}{n} \end{aligned}$$

Επομένως, η συνάρτηση ανταμοιβής είναι η ακόλουθη:

$$y_j(x_1, \ldots, x_j) = \begin{cases} \frac{j}{n} & \text{εάν ο } j \text{ αιτών είναι υποψήφιος,} \\ 0 & \text{διαφορετικά} \end{cases}$$

Είναι προφανές απο τον ορισμό της παραπάνω συνάρτησης οτι το y_j εξαρτάται μονάχα απο το x_j, καθώς επίσης οτι το πρόβλημα είναι μονότονο, αφού δεν μπορεί να σταματήσει σε κάποιον αιτών που να μην είναι υποψήφιος. Άρα ο *1-sla* κανόνας θα είναι βέλτιστος για αυτό το πρόβλημα. Ακόλουθεί η λύση του προβλήματος που αναπτύχθηκε στα πλαίσια της εργασίας αυτής και βασίζεται σε *1-sla* κανόνα και την επαγωγική μέθοδο:

$$\begin{aligned} E(Y_{j+1}|F_j) &= p_{j+1}y_{j+1} + q_{j+1}p_{j+2}y_{j+2} + \cdots + q_{j+1}q_{j+2}\cdots p_n y_n \\ &= \tfrac{1}{j+1}\tfrac{j+1}{n} + (1 - \tfrac{1}{j+1})\tfrac{1}{j+2}\tfrac{j+2}{n} + \cdots + (1 - \tfrac{1}{j+1})(1 - \tfrac{1}{j+2})\cdots\tfrac{1}{n} \\ &= \tfrac{1}{n}(1 + \tfrac{j}{j+1} + \tfrac{j}{j+2} + \cdots + \tfrac{j}{n-1}) \\ &= \tfrac{j}{n}(\tfrac{1}{j} + \tfrac{1}{j+1} + \tfrac{1}{j+2} + \cdots + \tfrac{1}{n-1}) \\ &= \tfrac{j}{n}\textstyle\sum_{k=j+1}^{n}\tfrac{1}{k-1} \end{aligned}$$

όπου $p_j = P(X_j = 1)$ και $q_j = 1 - p_j = P(X_j = 0)$. Απο τον ορισμό του *1-sla* κανόνα προκύπτει:

$$\begin{aligned} Y_j \geq E(Y_{j+1}|F_j) &\implies \tfrac{j}{n} \geq \tfrac{j}{n}\textstyle\sum_{k=j+1}^{n}\tfrac{1}{k-1} \\ &\implies \textstyle\sum_{k=j+1}^{n}\tfrac{1}{k-1} \leq 1 \end{aligned}$$

Επομένως, ο βέλτιστος κανόνας παύσης για το CSP είναι ο ακόλουθος:

$$N_1 = min\{j \geq 1 : \sum_{k=j+1}^{n} \frac{1}{k-1} \leq 1\}$$

Οι κανόνες αυτής της μορφής συναντώνται στην βιβλιογραφία και ως **κανόνες κατωφλίου** (treshold rules), διότι ουσιαστικά αυτό που κάνουν είναι να προσπερνούν τους πρώτους j αιτούντες (όσο δηλαδή ορίζει το κατώφλι) και στη συνέχεια να επιλέγουν τον πρώτο υποψήφιο που θα συναντήσουν. Στη συνέχεια ακολουθεί πίνακας στον οποίο δίδονται ενδεικτικές τιμές οριζόντων και τα αντίστοιχα κατώφλια με τις πιθανότητες επιτυχίας:

Απόδοση βέλτιστου κανόνα για το *CSP*								
Ορίζοντας n	1	2	3	4	5	6	7	8
Κατώφλι j	1	1	2	2	3	3	3	4
Πιθανότητα Νίκης P_j	1.0	.500	.500	.458	.433	.428	.414	.410

Πίνακας 3.1: Απόδοση βέλτιστου κανόνα για το *CSP*.

Τέλος, για πολύ μεγάλο n, ισχύει:

$$\begin{array}{ll} \sum_{k=j+1}^{n} \frac{1}{k-1} \approx \int_{j+1}^{n} \frac{1}{k-1} & \Longrightarrow \\ \int_{j+1}^{n} \frac{1}{k-1} \approx \log(\frac{n}{k}) & \Longrightarrow \\ \log(\frac{n}{k}) \geq 1 & \Longrightarrow \\ k \geq \frac{n}{e} & \end{array}$$

Άρα, για πολύ μεγάλο n είναι προσεγγιστικά βέλτιστο να προσπεράσει ο κανόνας παύσης το $e^{-1} = 36.8\%$ των αιτούντων και στη συνέχεια να επιλέξει τον πρώτο υποψήφιο που θα συναντήσει. Η πιθανότητα νίκης είναι e^{-1}.

3.3 Προβλήματα Βέλτιστης Επιλογής

Το *CSP* που περιγράφηκε στην προηγούμενη ενότητα ανήκει σε μια ιδιαίτερη κατηγορία προβλημάτων της θεωρίας της Βέλτιστης Πάυσης· ανήκει στα **προβλήματα βέλτιστης επιλογής** (best-choice problems). Στα προβλήματα αυτά στόχος είναι η επιλογή της καλύτερης λύσης. Στο *CSP*, για παράδειγμα , στόχος είναι η επιλογή του καλύτερου αιτούντος. Με απλά λόγια ο υπεύθυνος πρόσληψης δεν θα ικανοποιηθεί παρα μόνο εάν επιλέξει τον καλύτερο απο όλους. Βέβαια, υπάρχουν προβλήματα στα οποία η επιλογή μπορεί να είναι πιο ελαστική, όπως για παράδειγμα η επιλογή ενός εκ των καλύτερων υποψηφίων κ.λπ. Συνήθως αυτά τα προβλήματα αποτελούν επεκτάσεις του *CSP*.

Την παρούσα εργασία θα απασχολήσουν τα προβλήματα βέλτιστης επιλογής, γι αυτό και στη συγκεκριμένη ενότητα θα παρουσιαστεί ο αλγόριθμος *Sum the Odds*[6] [54] του F. Thomas Bruss, ο οποίος αποτελεί μια εκτεταμένη και γενικευμένη λύση των προβλημάτων βέλτιστης επιλογής καθώς και του *CSP*.

[6]Η πλήρης ονομασία του αλγορίθμου είναι *Sum the Odds to One and Stop*. Για λόγους όμως συντομίας, απο αυτό το σημείο του κειμένου και έπειτα, ο αλγόριθμος θα αναφέρεται ως *Odds* αλγόριθμος.

Ο αλγόριθμος αυτός σχεδιάστηκε για να ασχολείται με προβλήματα όπου οι παρατηρήσεις συνθέτουν μια φραγμένη (έστω n) ακολουθία απο **δείκτριες συνάρτησεις** (indicator functions) των οποίων οι δυνατές τιμές είναι 0 για **αποτυχία** (failure) ή 1 για **επιτυχία** (success). Στόχος είναι η παύση στην τελευταία επιτυχία, δηλαδή στην τελευταία παρατήρηση που έχει δείκτρια συνάρτηση με τιμή 1. Αυτό σημαίνει οτι εάν ο κανόνας φτάσει στην τελευταία παρατήρηση (φτάσει δηλαδή στον ορίζοντα) δίχως να έχει σταματήσει σε μια επιτυχία ή σταματήσει σε μια επιτυχία που δεν είναι η τελευταία, τότε έχει χάσει.

Ένα απλό παράδειγμα αποτελεί η ρίψη ενός **δίκαιου**[7] (fair) ζαριού n φορές. Είναι εύκολο να ανακοινωθεί η πρώτη φορά που θα εμφανιστεί ένα 6, αλλά δεν μπορεί να ειπωθεί το ίδιο και για την ανακοίνωση της τελευταίας εμφάνισης του 6. Στην περίπτωση ενός δίκαιου ζαριού η απάντηση δεν είναι δύσκολη. Η πιθανότητα να εμφανιστεί μία φορά το 6 στις τελευταίες l ρίψεις δίνεται απο την **διωνυμική κατανομή** (binomial distribution) και είναι ίση με $l6^{-1}(\frac{5}{6})^{l-1}$. Μάλιστα η πιθανότητα αυτή μεγιστοποιείται για $l = 5$ ή $l = 6$. Διαισθητικά, θα ήταν βέλτιστο να ανακοινωθεί το πρώτο 6 που εμφανίζεται στις τελευταίες 6 ρίψεις, ως το τελευταίο 6.

Τι γίνεται όμως στην περίπτωση εκείνη όπου είναι επιτρεπτή η ρίψη διαφορετικού αριθμού ζαριών σε κάθε ρίψη ή χρήση ενός **μεροληπτικού** (biased) ζαριού; Η παραπάνω λύση δεν θα ήταν εφαρμόσιμη. Πολλά προβλήματα είναι της ίδιας φιλοσοφίας· θέλουν δηλαδή να εφαρμοστεί η παύση στην τελευταία επιτυχία, απο μια ακολουθία παρατήρησεων. Προβλήματα αυτού του είδους μπορούν εύκολα να επιλυθούν, εάν διαθέτουν Markov δομή. Η λύση, όμως, που εφαρμόζεται σε αυτά τα προβλήματα δεν μπορεί να επεκταθεί σε προβλήματα με διαφορετική δομή.

Σε αυτό το σημείο έρχεται το **Odds θεώρημα** το οποίο λέει [54]:

> Έστω οτι η ακολουθία $I_1, I_2, \ldots, I_n$ είναι μια ακολουθία ανεξάρτητων δείκτριων συναρτήσεων με $p_j = E(I_j)$. Έστω οτι $q_j = 1 - p_j$ και $r_j = \frac{p_j}{q_j}$. Τότε υπάρχει ένας βέλτιστος κανόνας για παύση στην τελευταία επιτυχία και είναι η παύση στην πρώτη επιτυχία k ($I_k = 1$), για την οποία ισχύει $k \geq s$, όπου:
>
> $$s = sup\{1, sup\{1 \leq k \leq n : \sum_{j=k}^{n} r_j \geq 1\}\},$$
>
> με $sup\varnothing = -\infty$.

Η πιθανότητα νίκης δίνεται απο την σχέση: $V(n) = \prod_{j=s}^{n} q_j \sum_{j=s}^{n} r_j$.

Με βάση το παραπάνω θεώρημα προκύπτει και ο **Odds αλγόριθμος** ο οποίος ορίζεται ως εξής [54]:

- **A.1:** Κατέγραψε σε μια λίστα, σε αντίστροφη σειρά, τις ποσότητες $q_j = 1-p_j$ και $r_j = \frac{p_j}{q_j}$ και υπολόγισε αναδρομικά τις ποσότητες $Q_k := q_n q_{n-1} \cdots q_k$ και $R_k := r_n + r_{n-1} + \cdots + r_k$. Σταμάτα, όταν το άθροισμα R_k φτάσει ή ξεπεράσει την τιμή 1 ή όταν ο κανόνας φτάσει το $k = 1$.
- **A.2:** $V(n) := Q_s R_s$

[7] Στην βιβλιογραφία μπορεί να συναντηθεί και με τον όρο αμερόληπτο (π.χ. αμερόληπτο ζάρι).

Ο δείκτης παύσης s στο A1 αναπαριστά τον χρόνο εφεξής είναι βέλτιστο να σταματήσει ο κανόνας στην πρώτη επιτυχία που θα συναντήσει. Το A2 δίνει τη βέλτιστη πιθανότητα νίκης. Ουσιαστικά αυτό που προσπαθεί να κάνει ο *Odds* αλγόριθμος είναι να βρεί τον χρόνο t που μεγιστοποιεί την τιμή του $P(I_t = 1, I_{t+1} = I_{t+2} = \cdots I_n = 0)$.

Ο *Odds* αλγόριθμος είναι εύκολα εφαρμόσιμος και στην περίπτωση τυχαίου ορίζοντα, δηλαδή τυχαίου αριθμού δεικτών[8], αρκεί να διατηρείται η ανεξαρτησία τους. Εάν, λοιπόν, η $N(t)_{t\geq 0}$ είναι μια **στοχαστική διεργασία καταμέτρησης** (stochastic counting process) ορισμένη στο $\Re^+$, αποτελούμενη απο ανεξάρτητα **βήματα** (increments) και χρόνους αφίξεων συμβάντων $T_1, T_2, \ldots$, τότε το I_k αντιπροσωπεύει την επιτυχία ή αποτυχία ενός συμβάντος που αφίχθει την χρονική στιγμή T_k[9].

Μια τέτοια διεργασία είναι η Poisson διεργασία. Για μια **μη ομοιογενή** (inhomoeneous) Poisson διεργασία που έχει **παράμετρο ρυθμού** (rate parameter)[10] $\lambda(t)$ και συνάρτηση επιτυχίας $h(t)$, η λύση κάνοντας χρήση Poisson ανέλιξης και του Odds αλγορίθμου είναι η ακόλουθη [54]:

$$\tau = sup\{0, sup\{0 \leq t \leq T : \int_t^T \lambda(u)h(u)du \geq 1\}\}$$

Δηλαδή είναι βέλτιστο να ενεργοποιηθεί η παύση στην πρώτη επιτυχία που θα συμβεί μετά την χρονική στιγμή τ. Τα συγκεκριμένα συμπεράσματα είναι αρκετά σημαντικά για την ανάλυση που θα αναπτυχθεί στο επόμενο κεφάλαιο, καθώς ο κανόνας διακοπής που υλοποιήθηκε στα πλαίσια της εργασίας αυτής, βασίζεται στη θεώρηση τυχαίων αφίξεων **ομοιογενούς** (homogeneous) Poisson διεργασίας. Ο παραπάνω κανόνας έχει ως κάτω φράγμα για την πιθανότητα νίκης, την τιμή $1/e$ [54, 55]. Αυτό σημαίνει οτι για μεγάλο n και p_k ομοιόμορφα μικρά, ο παραπάνω κανόνας μπορεί να αποτελέσει μια καλή και ταχέως συγκλίνουσα προσέγγιση της διακριτής περίπτωσης. Επιπλέον, τα αποτελέσματα του συμπίπτουν και με έναν άλλο σημαντικό κανόνα, τον **1/e-κανόνα** (1/e-rule), στον οποίο το πρόβλημα επιλύεται θεωρώντας υπο συνθήκη ανεξαρτησία των αφίξεων [56].

Τέλος, σημαντικές επεκτάσεις στο *Οδδς* θεώρημα παρέχει ο Thomas S. Ferguson στην εργασία [57], όπου το *Οδδς* θεώρημα επεκτείνεται κατα πολλούς τρόπος, π.χ. δίνεται η δυνατότητα άπειρου ορίζοντα κάνοντας χρήση του λήμματος Borel-Cantelli, οι τυχαίες μεταβλητές μπορεί να είναι εξαρτημένες, κ.α. Σημαντική είναι, επίσης, η εργασία των F. Thomas Bruss και Guy Louchard, οπου παρουσιάζεται μια νέα εκδοχή του *Odds* αλγορίθμου βασισμένη σε ακολουθιακή ενημέρωση [58].

3.3.1 Σύνοψη

Συνοψίζοντας, στα πλαίσια του κεφαλαίου αυτού έγινε μια προσπάθεια να εξοικειωθεί ο αναγνώστης με την θεωρία Βέλτιστης Παύσης. Μια θεωρία σχετικά νεαρή, αφού ξεκίνησε να αναπτύσσεται στην δεκατία του 1940· ταχέως όμως αναπτυσσόμενη.

[8] Εννοείται τυχαίου αριθμού δείκτριων συναρτήσεων.
[9] Η τιμή του I_k μπορεί να εξαρτάται απο την χρονική στιγμή T_k
[10] Στην βιβλιογραφία μπορεί να συναντηθεί και με τον όρο **ένταση** (intensiy).

Σημαντικά προβλήματα, όπως το πρόβλημα της πώλησης ενός σπιτιού ή της επιλογής της καλύτερης γραμματέως, συνέβαλαν τα μέγιστα στην θεμελίωση των κανόνων παύσης, του εργαλείου εκείνου δηλαδή που χρησιμοποιείται απο τη συγκεκριμένη θεωρία. Στόχος των κανόνων αυτών είναι η επιλογή ενός κατάλληλου χρόνου παύσης τέτοιου ώστε να μεγιστοποιείται μια συγκεκριμένη απόδοση ή να ελαχιστοποιείται ένα συγκεκριμένο κόστος, μέσω της παρατήρησης μιας ακολουθίας τμ και της χρήσης μια ακολουθίας ανταμοιβής.

Για την εύρεση τέτοιων κανόνων υπάρχουν διάφοροι μέθοδοι. Μια απο τις σημαντικότερες μεθόδους είναι αυτή της προς τα πίσω επαγωγής. Αυτή η μέθοδος, όμως, δεν είναι εύκολα εφαρμόσιμη για πολυδιάστατα προβλήματα. Την λύση σε αυτό το πρόβλημα την δίνει η μέθοδος της περικοπής, κυρίως όμως οι *k-sla* κανόνες οι οποίοι περιορίζουν το πρόβλημα εύρεσης βέλτιστης παύσης στα πλαίσια ενός μικρού αριθμού φάσεων.

Μια απο τις πιο σημαντικότερες κατηγορίες προβλημάτων είναι αυτή των προβλημάτων της βέλτιστης επιλογής. Σε αυτή την κατηγορία ανήκει και ένα απο τα σημαντικότερα προβλήματα· το κλασσικό πρόβλημα της γραμματέως, το οποίο είναι ένα no-information, πεπερασμένου ορίζοντα πρόβλημα με παρα πολλές εφαρμογές και επεκτάσεις. Ο αλγόριθμος εκείνος που αποτελεί στις μέρες μας την πλέον γενική και εκτεταμένη λύση στα προβλήματα αυτής της κατηγορίας είναι ο *Odds* αλγόριθμος.

4. Ο ΜΗΧΑΝΙΣΜΟΣ Vproxy

Στο παρών κεφάλαιο παρουσιάζεται ο μηχανισμός συνέπειας που προτείνεται στα πλαίσια αυτής της εργασίας. Υπευθυμίζεται οτι η παρούσα διπλωματική εργασία ασχολείται με το πρόβλημα της διασφάλισης συνέπειας στην κρυφή μνήμη του Παγκόσμιου Ιστού, γνωστότερο ως Web caching consistency problem. Πιο συγκεκριμένα την απασχολεί το πρόβλημα αυτό απο την πλευρά των εξυπηρετητών κρυφής μνήμης (cache servers), δηλαδή των proxies εκείνων που υλοποιούν το Web caching, και προτείνει έναν μηχανισμό συνέπειας που εφαρμόζει την θεωρία της Βέλτιστης Παύσης. Στη συνέχεια υλοποιείται ένα προσομοιωτής ο οποίος συμβάλει στην μελέτη του μηχανισμού που προτείνεται και στην διεξαγωγή κατάλληλων συμπερασμάτων, τα οποία θα παρουσιαστούν στα επόμενα κεφάλαια. Γι αυτό, λοιπόν, είναι πιο δόκιμο να γίνεται λόγος για μελέτη ενός μηχανισμού, παρα για υλοποίηση του.

Ο προσομοιωτής που υλοποιείται στα πλαίσια αυτής της εργασίας ονομάζεται *Vproxy*, που σημαίνει **vitual proxy**. Η λέξη virtual, που στα ελληνικά μεταφράζεται ως εικονικός, χρησιμοποιείται για να δηλώσει το γεγονός της προσομοίωσης του μηχανισμού, ενώ η λέξη proxy χρησιμοποιείται για να εκφράσει αυτό που ήδη ειπώθηκε, οτι ο μηχανισμός αφόρα cache servers κυρίως. Απο το σημείο αυτό του κειμένου και έπειτα, ο όρος *Vproxy* θα χρησιμοποιείται για να προσδιορίσει και τον μηχανισμό συνέπειας αλλά και τον προσομοιώτη του μηχανισμού αυτού.

Στη συνέχεια, λοιπόν, του κεφαλαίου αυτού θα παρουσιαστεί το θεωρητικό υπόβαθρο στο οποίο στηρίζεται ο μηχανισμός που μελετάται, αλλά και τα σημαντικότερα μέρη της υλοποίησης του προσομοιώτη που χρησιμοποιείται για την μελέτη του. Πιο συγκεκριμένα η διάρθρωση του κεφαλαίου είναι η ακόλουθη:

- **Ενότητα 4.1**: Η ενότητα 4.1 περιγράφει την βασική λειτουργία του *Vproxy*. Τα βασικά βήματα που ακολουθεί για να εκτιμήσει την συνέπεια ενός αντικειμένου[1] που βρίσκεται στην cache, καθώς και τον τρόπο με τον οποίο το επιτυγχάνει. Περιγράφει τα δύο συστατικά στοιχεία της λειτουργίας του, την μετρική και τον κανόνα παύσης.

- **Ενότητα 4.2**: Η ενότητα 4.2 ασχολείται με την υλοποίηση του προσομοιωτή. Βέβαια δεν μπαίνει σε ιδιαίτερες λεπτομέρειες, μιας και το τεχνικό κομμάτι τεκμηριώνεται σε άλλα έγγραφα που παρέχονται στο CD της εργασίας. Περιγράφει μονάχα τα σημαντικότερα μέρη του προσομοιώτη, καθώς και την **διεπαφή χρήστη** (user interface) που προσφέρει, μέσω ενός πίνακα που παρουσιάζει τις διάφορες επιλογές.

- **Ενότητα 4.3**: Η ενότητα 4.3 κάνει μια σύνοψη όλων όσων παρουσιάστηκαν στις προηγούμενες ενότητες.

[1] Υπενθυμίζεται οτι σε μια cache βρίσκεται το αντίγραφο ενός αντικειμένου. Όποτε, λοιπόν, γίνεται αναφορά σε ένα αντικείμενο μιας cache θα θεωρείται προφανές οτι εννοείται το αντίγραφο του αντικειμένου του origin server.

4.1 Εισαγωγή στη λειτουργία του Vproxy

Στο δεύτερο κεφάλαιο έγινε μια αναλυτική παρουσιάστη του Web caching consinstency προβλήματος, καθώς και των διαφόρων μηχανισμών που προσπαθούν να το αντιμετωπίσουν. Παρουσιάστηκαν οι διάφορες κατηγοριοποιήσεις των μηχανισμών αυτών σε ασθενείς και ισχυρούς, σύγχρονους και ασύγχρονους, validation και invalidation, καθώς και τα διάφορα πλεονεκτήματα και μειονεκτήματα τους. Έγινε σαφής η εύνοια του Web προς του μηχανισμούς εκείνους που προσφέρουν ασθενή συνέπεια και είναι validation, μιας και ικανοποιούν καλύτερα την ιδιότητα της επεκτασιμότητας και υποστηρίζονται πλήρως απο το πιο διαδεδομένο πρωτόκολλο επικοινωνίας, το HTTP.

Ο μηχανισμός συνέπειας που είναι state of art στις μέρες μας, είναι ο *ATTL*. Ο μηχανισμός αυτός είναι ασφαλώς ασθενής μιας και δουλεύει κάνοντας χρήση TTLs και validation διότι η Web cache είναι εκείνη που αναλαμβάνει την επικοινωνία με τον origin server για τον έλεγχο συνέπειας. Επιπρόσθετα, είναι ένας σύγχρονος μηχανισμός, μιας και ο έλεγχος της συνέπειας ενός αντικειμένου αποθηκευμένου στην cache[2] γίνεται κατα τη διάρκεια μιας εισερχόμενης αίτησης.

Επομένως και ο μηχανισμός που προτείνεται και μελετάται στην παρούσα διπλωματική εργασία, ο *Vproxy*, θα ικανοποιεί τις παραπάνω ιδιότητες· με άλλα λόγια θα είναι ασθενής, σύγχρονος και validation. Στόχος του μάλιστα είναι ο περιορισμός των μειονεκτημάτων που προκύπτουν απο αυτές τις ιδιότητες. Πιο συγκεκριμένα ο *Vproxy* μπορεί να «πατήσει» πάνω στον *ATTL*, ώστε να προσφέρει ισχυρότερη συνέπεια δίχως να αυξηθεί το κόστος που οφείλεται σε άστοχη επικοινωνία με τον origin server για έλεγχο συνέπειας αντικειμένων της cache που είναι συνεπή.

Για να μπορέσει να λειτουργήσει αποδοτικά ο *Vproxy*, κάνει χρήση μιας συγκεκριμένης έννοιας· της έννοιας του **cacheability**. Το cacheability ενός αντικειμένου που βρίσκεται σε μια cache ορίζει την δυνατότητα χρήσης αυτού του αντικειμένου προς ικανοποίηση μιας εισερχόμενης αίτησης. Ο *Vproxy*, λοιπόν, κάνοντας χρήση του cacheability προσπαθεί να ικανοποιήσει ένα συγκεκριμένο tradeoff. Το tradeoff αυτό απαρτίζεται απο την αποφυγή δύο συμβάντων: την αποφυγή ενός slow hit, δηλαδή την αποστολή μιας αίτησης επικύρωσης στον origin server η οποία θα επιστρέψει απάντηση *304, Not Modified*, και την αποφυγή ενός stale delivery, δηλαδή την ικανοποίηση μιας αίτησης απο το αντικείμενο μιας cache όταν αυτό έχει τροποποιηθεί στον origin server.

Με απλά λόγια, όταν ένας μηχανισμός συνέπειας «βιάζεται» να ελέγξει την εγκυρότητα ενός αντικειμένου, διότι είναι επιφυλακτικός, αυξάνει τις πιθανότητες να συμβεί ένα slow hit. Εάν απο την άλλη «αργοπορήσει», τότε αυξάνει τις πιθανότητες να συμβεί ένα stale delivery. Ο *Vproxy*, λοιπόν, προσπαθεί να «αργοπορήσει» τόσο όσο χρειάζεται για να αποφύγει και τα δύο αυτά συμβάντα, κυρίως όμως προσπαθεί να αποφύγει ένα stale delivery δίχως να το πληρώσει με μεγάλο αριθμό απο slow hits. Για να το επιτύχει αυτό κάνει κατάλληλη χρήσης της έννοιας του cacheability την οποία αξιοποιεί μέσω μιας μετρικής και ενός κανόνα παύσης.

Πώς το επιτυγχάνει όμως αυτό; Αυτό προσπαθεί να το επιτύχει κάνοντας μια συγκεκριμένη υπόθεση. Η υπόθεση που κάνει είναι η εξής: εάν ικανοποιήσει τις εισερχόμενες αιτήσεις σε έναν proxy, κάνοντας χρήση του αντικειμένου που βρίσκεται στην cache του εώς ότου παρατηρηθεί η

[2]Μιας και απο αυτό το σημείο του κειμένου και μετά εξυπακούεται οτι θα αναφερόμαστε σε Web caches θα αποφεύγεται ο προσδιορισμός Web προς χάριν συντομίας.

μέγιστη τιμή cacheability για το αντικείμενο αυτό και στη συνέχεια προωθήσει την αίτηση εκείνη που θα συμβάλλει στη μέγιστη τιμή cacheability προς τον origin server, τότε θα καταφέρει να πληρώσει το μικρότερο δυνατό κόστος σε slow hits και stale deliveries. Με άλλα λόγια, ο *Vproxy* αποφασίζει να ικανοποιήσει τις εισερχόμενες αιτήσεις σε έναν proxy, κάνοντας χρήση του αντικειμένου της cache και στη συνέχεια επιλέγει να ελέγξει την εγκυροτητά αυτού, όταν καταφτάσει μια αίτηση η οποία καθορίσει την πλέον cacheable κατάσταση του αντικειμένου, μετά απο την οποία το cacheability του θα φθίνει σταθερά και άρα ο κίνδυνος για stale delivery θα αυξάνεται.

Ως εκ τούτου ο *Vproxy* παρατηρεί τις εισερχόμενες αιτήσεις προς ένα αντικείμενο της cache, αποτιμά το cacheability του αντικειμένου μέσω μιας μετρικής και προσπαθεί μέσω ενός κανόνα παύσης να προβλέψει την στιγμή που θα παρατηρηθεί η μέγιστη τιμή του cacheability, ώστε να ελέγξει την συνέπεια του αντικειμένου. Είναι προφανές σε αυτό το σχήμα πως εάν η μετρική δεν είναι ορισμένη καλά, δηλαδή αποτιμά εσφαλμένα το cacheability, ή ο κανόνας παύσης δεν προβλέπει ορθά την εμφάνιση της μέγιστης τιμής, οτι η απόδοση δεν θα είναι βέλτιστη.

Πιο αναλυτικά τα βασικά βήματα που ακολουθεί ο *Vproxy* κατα την άφιξη μιας αίτησης απο κάποιον client είναι τα ακόλουθα:

1. Αποδοχή της εισερχόμενης αίτησης. Εάν το ζητούμενο αντικείμενο δεν βρίσκεται στην cache, η αίτηση προωθείται στον origin server.

2. Εάν υπάρχει το ζητούμενο αντικείμενο στην cache γίνεται αποτίμηση του cacheability του βάσει μιας μετρικής. Η μετρική είναι μια ποσότητα που λαμβάνει τιμές στο διάστημα $[0, 1]$ και διαμορφώνεται μέσω της εκτίμησης διαφόρων χαρακτηριστικών του αντικειμένου.

3. Έλεγχος του κανόνα παύσης. Εάν ο κανόνας παύσης ενεργοποιηθεί και η τρέχουσα τιμή της μετρικής είναι η μέγιστη τιμή, που σημαίνει οτι το αντικείμενο έχει φτάσει στην πλέον cacheable κατάσταση, αποστέλεται αίτηση επικύρωσης προς τον origin server, διαφορετικά η αίτηση ικανοποιείται κατευθείαν απο την cache δίχως να υπάρξει επικοινωνία με τον origin server.

Απο τα παραπάνω γίνεται εύκολα αντιληπτό οτι το πρόβλημα της συνέπειας μοντελοποιείται απο τον *Vproxy* ως ένα πρόβλημα βέλτιστης επιλογής ή ισοδύναμα ως ένα πρόβλημα παύσης στην τελευταία επιτυχία, όπου ως επιτυχία ορίζεται η άφιξη μιας αίτησης που αυξάνει το cacheability του αντικειμένου.

4.1.1 Μετρικές

Ουσιαστική λειτουργία της μετρικής είναι η ποσοτική εκτίμηση, βάσει κάποιων συγκεκριμένων χαρακτηριστικών, της διαίσθησης που σχετίζεται με το cacheability ενός αντικειμένου μιας cache. Πιο συγκεκριμένα μια μετρική - είτε αποτελεί μια απλή μέτρηση ενός μεγέθους, είτε μια σύνθεση απλών μετρικών - είναι μια ποσότητα που στόχο έχει να δώσει στο μηχανισμό μια διαίσθηση για την δυνατότητα του αντικειμένου να ικανοποιήσει την τρέχουσα αίτηση. Μάλιστα, μια ακριβής μετρική θα πρέπει να δίνει μια όσο το δυνατόν πιο αντιπροσωπευτική εικόνα της κατάστασης του αντικειμένου που αποτιμά και μάλιστα να προσαρμόζεται κατάλληλα στα χαρακτηριστικά και τις ιδιορυθμίες αυτού.

Ο *Vproxy* παρέχει δύο μετρικές: την u_3 και την u_4. Η διαφορά μεταξύ αυτών των δύο μετρικών είναι μικρή και θα γίνει αντιληπτή κατα την περιγραφή τους.

Η u_3 αποτελεί την σύνθεση τριών επι μέρους μετρικών: της **δημοτικότητας του αντικειμένου** (object popularity, op), του **ποσοστού ευστοχίας ενός ιστοτόπου** (site hit ratio, shr) και του **χρονικού ποσοστού της αίτησης** (request time ratio, rtr). Δηλαδή, η u_3 συντίθεται απο την *op* που είναι αντικειμενοστραφής (object oriented), την *shr* που είναι site oriented και την *rtr* που είναι request oriented[3].

Η *op* μετρική ορίζεται απο τον ακόλουθο τύπο:

$$op = \frac{object\ requests}{total\ requests}$$

Δηλαδή ορίζεται ως ο λόγος των αιτήσεων που έγιναν σε αυτό αντικείμενο, προς τον συνολικό αριθμό των αιτήσεων που ικανοποιήθηκαν απο την cache. Επειδή, όμως, ο συνολικός αριθμός των αιτήσεων αυξάνεται διαρκώς, απο ένα σημείο και μετά η τιμή του θα είναι αρκετά μεγάλη, με αποτέλεσμα η *op* να δίνει αρκετά μικρές τιμές (πολύ κοντά στο 0) και έτσι να μην είναι ιδιαίτερα χρήσιμη. Επιπλέον, μιας και ο αριθμός των αιτήσεων σε ένα αντικείμενο αυξάνεται κατα ένα, η *op* θα παρουσιάζει ασήμαντες μεταβολές.

Μια βελτίωση στα παραπάνω πρόβληματα συνιστά ο ορισμός της *op* βάσει του συνολικού αριθμού των αιτήσεων που έγιναν στο site του αντικειμένου[4]. Ο νέος αυτός ορισμός αποτελεί βελτιστοποίηση για έναν ακόμα λόγο· διότι ουσιαστικά συσχετίζει τη δημοτικότητα του αντικειμένου με την δημοτικότητα του site στο οποίο ανήκει, αφού ισχύει:

$$op = \frac{object\ requests}{site\ total\ requests} = \frac{object\ popularity}{site\ popularity}$$

Βέβαια η παραπάνω βελτίωση δεν είναι και τόσο αποδοτική, αφού για ένα δημοφιλές site προκύπτουν τα ίδια προβλήματα αναφορικά με τα αντικειμενά του. Δηλαδή, η *op* για τα αντικείμενα αυτού του site έχει ασήμαντη συνεισφορά στην u_3[5]. Άλλο σημαντικό πρόβλημα είναι η μη συσχέτιση της *op* με την κατάσταση της cache, καθώς οι παραπάνω ορισμοί δεν λαμβάνουν υπόψιν τα αποθηκευμένα αντικείμενα του site στην cache. Συγκεκριμένα, ο συνολικός αριθμός των αιτήσεων ενός site αποτελεί τον αριθμό των αιτήσεων που έχουν παρατηρηθεί απο τη πρώτη στιγμή που έγινε αίτηση σε κάποιο αντικείμενο του site αυτού. Τι συμβαίνει, όμως, όταν ο αριθμός των αντικειμένων ενός site που βρίσκονται στην cache αλλάζει δραματικά ή εν πάσει περιπτώσει παρουσιάζει σημαντικές διακυμάνσεις.

Για όλους τους παραπάνω λόγους, ο ορισμός της *op* που χρησιμοποιείται απο τον *Vproxy* είναι ο ακόλουθος:

$$op = \frac{object\ requests}{current\ site\ total\ requests},$$

[3]Ουσιαστικά και αυτή η μετρική είναι object oriented. Ο όρος request oriented χρησιμοποιείται για να καταστήσει σαφέστερη την επίδραση μιας συγκεκριμένης αίτησης προς το αντικείμενο.

[4]Ο συνολικός αριθμός των αιτήσεων ενός site ορίζεται ως τον συνολικό αριθμό των αιτήσεων που έχουν γίνει σε αντικείμενα του site αυτού.

[5]Αλλά και σε οποιαδήποτε μετρική χρησιμοποιεί την *op* με αυτό τον ορισμό.

όπου ο παρονομαστής οριζεί τον συνολικό αριθμό των αιτήσεων ενός site, όπως αυτός προκύπτει απο τα αντικείμενα του site που βρίσκονται εκείνη την στιγμή αποθηκευμένα στην cache.

Η επόμενη μετρική είναι η *shr* και είναι αρκετά πιο απλή. Ο ορισμός της είναι προφανής, καθώς μετράει το ποσοστό των αιτήσεων προς αντικείμενα ενός site που ικανοποιήθηκαν απο την cache, προς το συνολικό αριθμό των αιτήσεων προς το site αυτό. Ο τύπος που δίνει την τιμή της *shr* είναι ο ακόλουθος:

$$shr = \frac{site\ hits}{site\ total\ requests}$$

Η *rtr* μετρική είναι η πλέον σημαντικότερη, καθώς ενθυλακώνει στον ορισμό της την έννοια της ηλικίας ενός αντικειμένου. Η *rtr* ορίζεται ως εξής:

$$rtr = \min\{\frac{t_c - t_l}{t_e - t_l}, 1\},$$

όπου t_c είναι η τρέχουσα χρονική στιγμή[6], t_l είναι η χρονική στιγμή φόρτωσης του αντικειμένου στην cache και t_e είναι χρονική στιγμή λήξης του TTL του αντικειμένου. Σκοπός του min είναι ο περιορισμός της τιμής αυτής της μετρικής στο διάστημα $[0, 1]$. Δηλαδή $\forall t_c, t_e \leq t_c$ η *rtr* παίρνει τιμή 1. Σύμφωνα με το παραπάνω ορισμό η *rtr* αποτελεί τον λόγο της τρέχουσας ηλικίας ενός αντικειμένου της cache, προς το TTL αυτού.

Μετά την παρουσίαση των παραπάνω μετρικών είναι δυνατός ο ορισμός της u_3. Ο τύπος που ορίζει την u_3 είναι ο ακόλουθος:

$$u_3 = w_1 \cdot shr + w_2 \cdot op + w_3 \cdot rtr,$$

όπου τα w_1, w_2, w_3 αποτελούν τα βάρη των διάφορων μετρικών και συνήθως ισχύει $w_1 = w_2 = w_3 = 1/3$.

Απο τον τύπο προκύπτει οτι η u_3 είναι ανάλογη των τριών μετρικών. Δηλαδή, όσο αυξάνονται οι τιμές των μετρικών τόσο αυξάνεται η τιμή της u_3 και το αντίστροφο. Δηλαδή τόσο αυξάνεται το cacheability του αντικειμένου. Το πως αξιοποιεί αυτή την λειτουργικότητα ένας κανόνας παύσης, θα αναλυθεί στην επόμενη ενότητα.

Η δεύτερη μετρική που υλοποιείται απο τον *Vproxy* είναι η u_4. Η μετρική αυτή επεκτείνει την u_3, χρησιμοποιώντας μια επιπλέον μετρική, την **μεταβλητότητα του αντικειμένου** (object mutability, om).

Η *om* εκφράζει το βαθμό μεταβολής ενός αντικειμένου και ορίζεται ως ο λόγος του αριθμού των τροποποιήσεων που έχουν παρατηρηθεί σε ένα αντικείμενο, προς το συνολικό αριθμό αιτήσεων στο αντικείμενο αυτό. Ο τύπος που δίνει την τιμή της *om* είναι ο ακόλουθος:

$$om = \frac{object\ modifications}{object\ requests}$$

[6]Ως τρέχουσα χρονική στιγμή εννοείται η χρονική στιγμή αποδοχής της τρέχουσα αίτησης απο την cache.

Ο τύπος που ορίζει την u_4 είναι ο ακόλουθος:

$$u_4 = w_1 \cdot shr + w_2 \cdot op + w_3 \cdot rtr + w_4 \cdot om,$$

όπου τα w_1, w_2, w_3, w_4 αποτελούν τα βάρη των διάφορων μετρικών και συνήθως ισχύει $w_1 = w_2 = w_3 = w_4 = 1/4$.

Στόχος είναι η εξέταση της απόδοσης του *Vproxy* βάσει των δύο μετρικών, ώστε να βγούν τα κατάλληλα συμπεράσματα όσον αφορά την ακρίβεια του ορισμού τους. Στη συνέχεια ακολουθεί η περιγραφή των κανόνων παύσης, καθώς και ο τρόπος με τον οποίο αυτοί αξιοποιούν τις μετρικές.

4.1.2 Κανόνες Παύσης

Ο *Vproxy* παρέχει δύο κανόνες παύσης, τον *1/e* και τον *Sum the Odds* για περίπτωση συνεχούς χρονού και τυχαίο αριθμό αφίξεων.

Ο *1/e* κανόνας είναι πολύ απλός. Παρουσιάστηκε απο τον F.Thomas Bruss το 1984 στην εργασία του *A Unified Approach to a Class of Best Choice Problems with an Unknown Number of Options* σε μια προσπάθεια να ενοποιήσει τα προβλήματα βέλτιστης επιλογής, κάτω απο την πλήρη άγνοια του αριθμού των παρατηρήσεων αλλά και της κατανομής αυτού [56]. Τα περισσότερα προβλήματα αυτής της κατηγορίας, όπως για παράδειγμα στο CSP, είτε θέτουν ως προαπαιτούμενο την γνώση του αριθμού των παρατηρήσεων (π.χ. αριθμός αιτούντων για την θέση της γραμματέως), είτε την κατανομή που ακολουθεί αυτός ο αριθμός στην περίπτωση που είναι τμ.

Ο Lidley, το 1961, βρήκε οτι για την περίπτωση του CSP όπου είναι γνωστός ο αριθμός n, ο βέλτιστος κανόνας είναι η παράλειψη των πρώτων $k^*(n)$ αιτούντων και στη συνέχεια η αποδοχή του πρώτου υποψηφίου [59]. Μάλιστα για πολύ μεγάλο n ισχύει $\lim_{n\to\infty} \frac{k*(n)}{n} = e^{-1}$.

Οι Presman και Sonin έλυσαν το CSP το 1972, για την περίπτωση όπου ο αριθμός των αιτούντων είναι τυχαίος, καθορίζεται δηλαδή απο την τμ N. Η λύση όμως που έδωσαν έχει χειρότερη απόδοση, καθώς για μεγάλο n η τιμή της βέλτιστης πιθανότητας νίκης τείνει στην τιμή $2e^{-2}$ [60]. Επιπλέον, σε αυτή την εργασία θεωρήθηκε γνωστή η κατανομή του N.

Στην εργασία του F. Thomas Bruss προτείνεται ο κανόνας *1/e* ο οποίος αντιμετωπίζει όλα τα παραπάνω προβλήματα, αφού δεν θεωρεί γνωστό το n ούτε και την κατανομή αυτού, και επιπλέον επιτυγχάνει βέλτιστη πιθανότητα νίκης με κάτω φράγμα το $1/e$.

Η λογική, λοιπόν, του *1/e* κανόνα είναι πολύ απλή:

> Έστω οτι η χρονική στιγμή άφιξης κάθε αιτούντα είναι iid με συνάρτηση πυκνότητας πιθανότητας (probability density function, pdf) f στο διάστημα $[0, T]$ και συνάρτηση κατανομής (distribution function) F τέτοια ώστε να ισχύει:
>
> $$F(t) = \int_0^t f(s)ds, 0 \leq t \leq T.$$

> Και έστω χρόνος τ τέτοιος ώστε να ισχύει $F(\tau) = 1/e$. Τότε είναι βέλτιστο να περιμένεις μέχρι την χρονική στιγμή τ και στη συνέχεια να επιλέξεις, εάν είναι δυνατόν, τον πρώτο υποψήφιο που θα συναντήσεις.

Έαν υποτεθεί οτι οι χρόνοι άφιξης των αιτούντων είναι ομοιόμορφα iid στο $[0, T]$ τότε αυτό που ουσιαστικά θα κάνει ο *1/e* κανόνας θα είναι η επιλογή του πρώτου υποψηφίου με χρόνο άφιξης t, όπου $e^{-1}T \leq t$.

Απο τον παραπάνω ορισμό του *1/e* κανόνα γίνεται ολοφάνερη η ανύπαρκτη συσχετισή του με τις μετρικές. Ο *1/e* δεν λαμβάνει υπόψιν του κάποια μετρική για τον καθορισμό της βέλτιστης στρατηγικής.

Την συσχέτιση αυτή την δίνει ο δεύτερος κανόνας, ο *Sum the Odds* ή *Odds* για λόγους συντομίας. Ο *Odds* κανόνας προέρχεται απο την εφαρμογή του Odds θεωρήματος, όπως αυτό παρουσιάστηκε απο τον F. Thomas Bruss στην εργασία του *Sum the Odds to One and Stop* [54]. Σύμφωνα με αυτό το θεώρημα ο βέλτιστος κανόνας για την εύρεση της τελευταίας επιτυχίας δίνεται απο την ακόλουθη σχέση:

$$s = sup\{1, sup\{1 \leq k \leq n : \sum_{j=k}^{n} r_j \geq 1\}\}.$$

Η παραπάνω σχέση, όμως, λύνει το πρόβλημα της βέλτιστης επιλογής για την περίπτωση όπου ο αριθμός των παρατηρήσεων (π.χ. των αιτούντων στο *CSP*) είναι γνωστός εκ των προτέρων. Και συνεπώς σε αυτή την περίπτωση, επειδή ο χρόνος παύσης συμπίπτει με τον **δείκτη** (index) μιας παρατήρησης στην ακολουθία των παρατηρήσεων, λέμε οτι το πρόβλημα ανήκει στην περίπτωση του διακριτού χρόνου. Στον πραγματικό, όμως, κόσμο αυτή η μοντελοποίηση δεν είναι και τόσο εφάρμοσιμη. Ο χρόνος είναι συνεχής, οι παρατηρήσεις γίνονται σε τυχαίες χρονικές στιγμές ή όπως συνηθίζεται να διατυπώνεται οι αφίξεις των παρατηρήσεων γίνονται σε τυχαίες χρονικές στιγμές και τέλος τον ορίζοντα σε ένα τέτοιο πρόβλημα τον ορίζει ένα πραγματικό χρονικό διάστημα.

Για όλους του παραπάνω λόγους χρησιμοποιείται μια ειδική περίπτωση του *Odds* κανόνα. Συγκεκριμένα χρησιμοποιείται ο *Odds* κανόνας για συνεχές χρόνο και τυχαίες αφίξεις ομοιογενούς Poisson διεργασίας, ο οποίος ορίζεται ως εξής:

$$\tau = sup\{0, sup\{0 \leq t \leq T : \lambda \int_t^T h(u)du \geq 1\}\}$$

Ο παραπάνω κανόνας προέρχεται απο την εργασία [54], με τη διαφορά οτι έχει τροποποιηθεί ελαφρώς για να ικανοποιεί τον ορισμό μια ομοιογενούς Poisson διεργασίας. Επειδή έχει παρατηρηθεί πως οι χρονικές στιγμές των αφίξεων αιτήσεων προς ένα αντικείμενο της cache ακολουθούν την Poisson διεργασία [22], εξού και η χρησιμοποιήση της ομοιογενούς Poisson για την μοντελοποίηση του προβλήματος στο συνεχή χρόνο.

Υπάρχουν αρκετές εργασίες που έχουν προτείνει λύση στο συγκεκριμένο πρόβλημα, δηλαδή στο πρόβλημα της βέλτιστης επιλογής με τυχαίες αφίξεις. Οι Cowan και Zabczyk πρότειναν το 1976 μια λύση σε αυτό το πρόβλημα θεωρώντας ένα no-information μοντέλο με Poisson αφίξεις και γνωστό ρυθμό άφιξης [61]. Ο F. Thomas Bruss πρότεινε μια λύση το 1987 κάνοντας χρήση

ενός πιο ρεαλιστικού μοντέλου, όπου ο ρυθμός αφίξεων δεν είναι γνωστός αλλά έχει prior εκθετική κατανομή με παράμετρο θ [62].

Στα πλαίσια της εργασία αυτής θα θεωρηθεί οτι το πρόβλημα είναι full-information, μιας και οι παρατηρήσεις είναι αληθινές τιμές, είναι δηλαδή οι τιμές της μετρικής κατα τη διάρκεια αποτίμησης μια αίτησης, και επιπλέον δίχως απώλεια της γενικότητας οι παρατηρήσεις αυτές μπορούν να θεωρηθούν ως ομοιόμορφα iid στο $[0, 1]$ Επομένως, ο τύπος του *Odds* κανόνα που θα χρησιμοποιηθεί για Poisson αφίξεις και full-information μοντέλο είναι ο ακόλουθος:

$$\lambda(1 - x)t \leq 0.80435\ldots,$$

όπου t είναι ο υπολειπόμενος χρόνος, δηλαδή ο χρόνος που απομένει μέχρι τον ορίζοντα T (εν προκειμέων το TTL του αντικειμένου), λ είναι ο ρυθμός έλευσης (η ένταση) των Poisson αφίξεων (εν προκειμένω ο ρυθμός άφιξης των αιτήσεων) και x η **μέγιστη τιμή** (record) (εν προκειμένω της μετρικής), που έχει παρατηρηθεί στις προηγούμενες αιτήσεις. Στη συνέχεια ακολουθεί η απόδειξη του παραπάνω τύπου που αναπτύχθηκε στα πλαίσια της εργασίας αυτής και στην οποία γίνεται χρήση του Odds θεωρήματος.

Απόδειξη:

Υπενθυμίζεται οτι ο κανόνας του Odds θεωρήματος είναι ο εξής:

$$s = sup\{1, sup\{1 \leq k \leq n : \sum_{j=k}^{n} r_j \geq 1\}\} \quad (1)$$

Ο παραπάνω κανόνας μπορεί ισοδύναμα να γραφεί ως εξής:

$$s = \min\{k \geq 0 : \sum_{j=k+1}^{n} r_j \leq 1\} \quad (2)$$

Η (2) αποτελεί, ουσιαστικά, τον *1-sla* κανόνα για το Odds θεωρήμα και υπο την αυστηρή συνθήκη της μονοτονίας του προβλήματος.

Με μια πιο προσεκτική παρατήρηση της σχέσης (2) θα γίνει αντιληπτό οτι το Odds θεωρήμα λέει κάτι πολύ απλό:

> *Εάν η πιθανότητα να συμβεί ακριβώς μια επιτυχία μετά την φάση k είναι μικρότερη απο την πιθανότητα να μη συμβεί κάποια επιτυχία μετά την φάση k, τότε σταμάτα στην πρώτη επιτυχία (εάν είναι δυνατόν) που θα συναντήσεις αμέσως μετά την k.*

Έστω, λοιπόν, οτι έχουμε τις εξής δύο πιθανότητες[7]:

$$\begin{aligned} W_k &= P(\textit{ακριβώς μια επιτυχία μετά την φάση } k) \\ &= \sum_{j=k+1}^{n} P(X_{k+1} = \cdots = X_{j-1} = 0, X_j = 1, X_{j+1} = \cdots X_n = 0), \quad (3) \end{aligned}$$

$$\begin{aligned} V_k &= P(\textit{καμία επιτυχία μετά την φάση } k) \\ &= P(X_{k+1} = X_{k+2} = \cdots = X_n = 0), \quad (4) \end{aligned}$$

[7]Η σημειογραφία σε αυτό το σημείο προέρχεται απο την εργασία [57].

Απο την (2) και τις (3), (4) ο κανόνας γράφεται ισοδύναμα ως εξής:

$$s = \min\{k \geq 0 : \frac{W_k}{V_k} \leq 1\} \quad (5)$$

Επομένως, για να βρεθεί ο *Odds* κανόνας στην περίπτωση μιας Poisson διεργασίας, αρκεί να βρεθούν οι (3), (4), κ (5) γι αυτή.

Σύμφωνα με τον M. Sakaguchi [63], η πιθανότητα να μην έρθεί record στον υπολειπόμενο χρόνο t, δεδομένου οτι η τρέχουσα μέγιστη τιμή είναι x, είναι:

$$\sum_{j=0}^{\infty} x^j e^{-\lambda t} \frac{(\lambda t)^j}{j!} = e^{-\lambda t(1-x)} \quad (6)$$

Άρα, μιας και γίνεται λόγος για υπολειπόμενο χρόνο t και ο ορίζοντας είναι το T, η παραπάνω πιθανότητα ισοδυναμεί με την πιθανότητα κανενός record μετά την χρονική στιγμή $T-t$. Ποιά είναι, όμως, η πιθανότητα να έρθει ακριβώς ένα record στο υπολειπόμενο διάστημα t και μάλιστα μετά απο χρονικό διάστημα s; Έστω οτι αυτή η πιθανότητα συμβολίζεται με p_s, προκύπτει:

$$\begin{aligned} p_s &= P(\textit{κανένα record στο } [T-t, (T-t)+s)) \cdot P(\textit{μοναδικό record στο } (T-t)+s) \\ &= e^{-\lambda s(1-x)} \cdot \lambda dt \cdot P(\textit{κανένα record στο υπολειπόμενο } t-s | X_s > x) \\ &= e^{-\lambda s(1-x)} \cdot \lambda dt \cdot \int_x^1 e^{-\lambda s(1-y)(t-s)} dy \end{aligned}$$

Συνεπώς, η πιθανότητα να έρθεί ακριβώς ένα record στο υπολειπόμενο διάστημα t δεδομένης της τρέχουσας μέγιστης τιμής x είναι ίση με:

$$\int_0^t p_s ds = \int_0^t e^{-\lambda s(1-x)} \lambda \int_x^1 e^{-\lambda s(1-y)(t-s)} dy ds \quad (7)$$

Άρα, απο τις σχέσεις (5), (6) και (7) προκύπτει:

$$\begin{aligned} &\frac{\int_0^t e^{-\lambda s(1-x)} \lambda \int_x^1 e^{-\lambda s(1-y)(t-s)} dy ds}{e^{-\lambda t(1-x)}} \leq 1 \implies \\ &\int_0^t \frac{1}{t-s} (e^{\lambda(1-x)(t-s)} - 1) ds \leq 1 \implies \\ &\int_0^u \frac{1}{u} (e^u - 1) du \leq 1, \quad (8) \end{aligned}$$

όπου $u = \lambda(1-x)(t-s)$. Το πρόβλημα είναι μονότονο, εφόσον το ολοκλήρωμα στην (8) είναι μονότονο και για να ισχύει η ανισότητα πρέπει να ισχύει $u \leq 0.8435\ldots$.

Τελικά, ο βέλτιστος κανόνας παύσης για ομοιογενή Poisson διεργασία και full-information πρόβλημα, με εφαρμογή του Odds θεωρήματος είναι ο ακόλουθος:

$$\lambda(1-x)t \leq 0.8435\ldots$$

Η παραπάνω ανισότητα συμφωνεί απόλυτα με την λύση που έχουν δώσει οι Sakaguchi και Bodjecki στις αντίστοιχες εργασίες τους [63,64]. Μάλιστα συμφωνεί απόλυτα με την λύση που έχουν δώσει οι Ferguson, Hardwick και Tamaki στο πρόβλημα μεγιστοποίησης της διάρκειας

κατοχής ενός σχετικά καλύτερου αντικειμένου σε full-information μοντέλο, με τυχαίες αφίξεις και δυνατότητα ανάκλησης [65].

Σε αυτό το σημείο είναι σημαντικό να αναλυθεί η πρακτική σημασία του *Odds* κανόνα, ο τρόπος δηλαδή που λειτουργεί στην πράξη. Είναι προφανές απο τον παραπάνω τύπο πως ο *Odds* κανόνας θα ενεργοποιηθεί νωρίς, εάν ένα αντικείμενο έχει μικρό TTL ή πολύ μικρό λ. Τι γίνεται, όμως, με το x το οποίο συμβολίζει την μέγιστη τιμή ή record που έχει παρατηρηθεί μέχρι στιγμής; Εάν παρατηρηθεί ένα αρκετά μεγάλο x μέσα στις πρώτες αφίξεις τότε ο *Odds* κανόνας θα ενεργοποιηθεί νωρίς. Ακριβέστερα, όσο πιο μεγάλο είναι το x, τόσο πιο σύντομα τείνει να ενεργοποιηθεί ο *Odds* κανόνας! Αυτό είναι αρκετά λογικό, διότι εάν για παράδειγμα το x έχει τιμή $0.91\ldots$ τότε η πιθανότητα να παρατηρηθεί μια τιμή μεγαλύτερη απο αυτή στον υπολειπόμενο χρόνο είναι αρκετά μικρή[8]. Με απλά λόγια, εάν ο *Odds* κανόνας παρατηρήσει μια αρκετά μεγάλη τιμή για το x γίνεται αρκετά επιφυλακτικός απέναντι στο ενδεχόμενο συνέχισης λήψης περισσότερων παρατηρήσεων, καθώς πιστεύει οτι είναι απίθανο να συναντήσει μεγαλύτερη τιμή. Θα συνεχίσει να λαμβάνει παρατηρήσεις, παρότι το x έχει μεγάλη τιμή, μόνο εάν «δεί» οτι ο υπολειπόμενος χρόνος είναι αρκετός ή το λ μεγάλο.

Μιας και στον *Vproxy* και πιο συγκεκριμένα στον *Odds* κανόνα το x αντιπροσωπεύει την μέγιστη τιμή της παρατηρούμενης μετρικής, προκύπτει απο την παραπάνω ανάλυση ένα πολύ σημαντικό συμπέρασμα όσον αφορά την λειτουργία της μετρικής . Το συμπέρασμα είναι απλό και είναι το εξής: όσο πιο επίφοβη είναι η ικανοποίηση μιας εισερχόμενης αίτησης απο το αντικείμενο μιας cache, τόσο περισσότερο η μετρική θα τείνει να αποτιμά το αντικείμενο αυτό με μεγάλες τιμές, έτσι ώστε να επισπευθεί ο έλεγχος της συνεπειάς του. Εάν για παράδειγμα ένα αντικείμενο είναι αρκετά δημοφιλές, οι τιμές της μετρικής γι αυτό είτε θα είναι μεγάλες είτε θα αυξάνονται με ρυθμό ταχύτερο απο την περίπτωση ενός μη δημοφιλούς αντικειμένου.

Βέβαια, η μέγιστη αξιοποίηση μιας cache επιτυγχάνεται μέσω της ικανοποιήσης αιτήσεων προς δημοφιλή αντικείμενα, αφού έτσι αυξάνεται ο αριθμός των cache hits. Επιπλέον, η cache θα αποτελείται κυρίως απο δημοφιλή αντικείμενα αφού ακόμα κι αν συμβεί κάποια **έξωση** (eviction) σε ένα δημοφιλές αντικείμενο της cache[9], το αντικείμενο αυτό θα είναι σύντομα πίσω στην cache λόγω επακόλουθων αιτήσεων. Ακόμα περισσότερο ένας μηχανισμός συνέπειας ενδέχεται να πληρώσει πολύ μεγαλύτερο κόστος σε stale deliveries απο μια τροποποίηση σε ένα δημοφιλές αντικείμενο, ακόμα και αν σπάνια γίνονται τροποποιήσεις σε αυτό, σε σχέση με ένα μη δημοφιλές. Για όλους τους παραπάνω λόγους η μετρική για τα αντικείμενα που βρίσκονται στην cache δεν μπορεί, αλλά και δεν πρέπει να έχει μεγάλες διακυμάνσεις στην τιμή της[10] .

Τι θα συμβεί όμως εάν γίνει μια τροποποίηση σε ένα δημοφιλές αντικείμενο; Εάν ο υπολειπόμενος χρόνος για την λήξη του TLL είναι μεγάλος και δεδομένου οτι ένα δημοφιλές αντικείμενο θα έχει μεγάλο λ, ο *Odds* κανόνας θα αργήσει να ενεργοποιηθεί, με αποτέλεσμα να παρατηρηθεί το φαινόμενο της έκρηξης απο stale deliveries (stale deliveris burst). Πρέπει, λοιπόν, με κάποιο τρόπο να αποφευχθεί το φαινόμενο αυτό.

Η λύση που εφαρμόζει ο *Vproxy* είναι η προσαρμογή του κανόνα παύσης. Δηλαδή, εάν ο

[8] Αυτό βέβαια αποτελεί και άμεση συνεπαγωγή της ομοιόμορφης κατανομής που έχει υποτεθεί στην παρούσα μοντελοποίηση. Βάσει της ομοιόμορφης κατανομής για $x = 0.91\ldots$, ισχύει $P(X > 0.91\ldots) = 1 - P(X \leq 0.91\ldots) \approx 0.09$.

[9] Λόγω έλειψης χώρου, μέσω κάποιας πολιτικής αντικατάστασης (replacement policy)

[10] Αυτή η παρατήρηση αφορά κυρίως τα δημοφιλή αντικείμενα.

Vproxy θεωρήσει οτι ένα αντικείμενο της cache έχει δεχθεί μεγάλο αριθμό αιτήσεων[11], σταματά να χρησιμοποιεί τον *Odds* κανόνα και χρησιμοποιεί τον *1/e*. Πότε, όμως, θεωρείται μεγάλος ο αριθμός των αιτήσεων που έχει δεχθεί ένα αντικείμενο; Στα πλαίσια της εργασίας αυτής έχει υλοποιηθεί μια πολύ απλή λογική: για ένα αντικείμενο στο οποίο δεν έχει παρατηρηθεί καμία τροποποίηση ο αριθμός αυτός καθορίζεται στις 10000 αιτήσεις, για ένα αντικείμενο στο οποίο έχει παρατηρηθεί μια τροποποίηση ο αριθμός διπλασιάζεται στις 20000 αιτήσεις, κ.ο.κ. Παρότι φαίνεται παράδοξο να αυξάνεται το κατώφλι αλλαγής του κανόνα παύσης μετά απο κάθε τροποποίηση, δεν είναι. Υπευθυμίζεται οτι αυτή η λύση αφορά δημοφιλή αντικείμενα τα οποία συνήθως δεν παρουσιάζουν μεγάλη μεταβλητότητα και οτι επιπλέον έχει ενεργοποιηθεί ο *1/e* κανόνας ο οποίος είναι ιδιαίτερα επιφυλακτικός και άρα ακόμα και αν το αντικείμενο αλλάξει ξαφνικά συμπεριφορά και αρχίζει να παρουσιάζει συχνές τροποποιήσεις, αυτές θα ανιχνευτούν σύντομα.

4.2 Υλοποίηση

Στην ενότητα αυτή θα περιγραφούν τα βασικότερα ζητήματα που σχετίζονται με την υλοποίηση του προσομοιωτή. Έχει ήδη λεχθεί οτι ο μηχανισμός συνέπειας που προτείνεται και μελετάται στα πλαίσια αυτής της εργασίας αφορά έναν proxy. Συνεπώς και ο προσομοιωτής που υλοποιείται για τις ανάγκες της εργασίας, προσομοιώνει την λειτουργία ενός proxy που χρησιμοποιεί τον μηχανισμό συνέπειας όπως αυτός περιγράφηκε στην προηγουμένη ενότητα.

Πιο συγκεκριμένα ο προσομοιωτής έχει υλοποιηθεί σε γλώσσα Java με την χρήση διαφόρων **σχεδιαστικών προτύπων** (design patterns), χρησιμοποιεί ως εισοδό του Web traces και υποστηρίζει πλήρως το http caching, δηλαδή το Web caching όπως αυτό καθορίζεται απο το HTTP πρωτόκολλο [3, 28].

Τα βασικά κομμάτια του προσομοιωτή που είναι υπεύθυνα για κύριο μέρος της λειτουργίας του είναι τα εξής:

- **Parser:** Ο Parser είναι υπεύθυνος για την ορθή υλοποίηση της διεπαφής μεταξύ του χρήστη και του προσομοιωτή. Ουσιαστικά, είναι υπεύθυνος για την αναγνώριση της εισόδου του χρήστη και την κατάλληλη αρχικοποίηση των παραμέτρων της εφαρμογής (σ.σ. του προσομοιωτή), ώστε σε συνεργασία με το *vproxy.params* αρχείο που περιέχει τις **προεπιλεγμένες** (default) τιμές, να γίνει η κατάλληλη αρχικοποίηση του περιβάλλοντος στο οποίο θα γίνει η προσομοίωση.

- **PreProcessor:** Ο PreProcessor προσφέρει την διεπαφή προεπεξεργαστή στον προσομοιωτή. Εφόσον ο προσομοιωτής λαμβάνει ως είσοδο Web traces, πρέπει να έχει την δυνανότητα επεξεργασίας αυτών πριν ξεκινήσει την προσομοίωση. Διότι δεν έχουν όλα τα Web traces την ίδια δομή και επιπλέον μπορεί να ελλοχεύνουν διάφορα **σφάλματα** (bugs) ή ατέλειες για τα οποία απαιτείται η κατάλληλη διαχειρισή τους μέσω προεπεξεργασίας, ώστε η προσομοίωση να μην οδηγήσει σε παραπλανητικά αποτελέσματα.

[11] Στο σημείο αυτό είναι σημαντικό να διευκρινιστεί οτι ο αριθμός των αιτήσεων προς ένα αντικείμενο μετράται όσο αύτο βρίσκεται στην cache. Εάν κάποια στιγμή γίνει έξωση του αντικειμένου απο την cache και εν συνεχεία αυτό αποθηκευτεί ξανά στην cache λόγω μελλοντικών αιτήσεων, η μέτρηση των αιτήσεων θα ξεκινήσει πάλι απο την αρχή.

- **Generator:** Ο Generator γενν άει τα μηνύματα αιτήσεων/απαντήσεων που θα προσομοιώσουν την λειτουργία ενός proxy. Παράγει ουσιαστικά την ροή του διαδικτύου οδηγώντας έτσι στην προσομοίωση της επικοινωνίας στο Web μεταξύ των clients και του proxy, και μεταξύ του proxy και των origin servers.

- **ProxyServer:** Ο ProxyServer παρέχει την προσομοίωση της λειτουργίας ενός proxy που υλοποιεί Web caching.

- **Environment:** Είναι το πλέον σημαντικό κομμάτι του προσομοιωτή, καθώς ουσιαστικά περιέχει όλη εκείνη την κατάλληλη πληροφορία που είναι αναγκαία για την ενορχήστρωση της λειτουργίας του προσομοιωτή. Περιέχει την σχετική πληροφορία με τον κανόνα παύσης που έχει επιλεγεί, την μετρική, την πολιτική απόδοσης TTL στα αντικείμενα, το είδος της cache κ.λπ.

- **Cache:** Στόχος της cache είναι η προσομοίωση της λειτουργίας μιας Web cache. Ο προσομοιωτής παρέχει τη δυνατότητα επιλογής μεταξύ μιας LRU και μιας Infinite cache. Η cache προσφέρει λειτουργίες καταλόγου (catalog) και μνήμης (memory), ενώ επιπλέον προσφέρει και έναν validation μηχανισμό για τα αντικειμένα της.

- **Configuration:** Υλοποιεί τον μηχανισμό συνέπειας του *Vproxy* ως μια **μηχανή καταστάσεων** state machine, αφού σε κάθε εισερχόμενη αίτηση αποτιμάται το αντικείμενο της cache μέσω της μετρικής, του κανόνα παύσης και της αντίστοιχης μετα-πληροφορίας και μεταβαίνει σε μια νέα κατάσταση, βάσει του μηχανισμού αυτού.

- **ModificationMap:** Ο ModificationMap είναι ένας hashmap ο οποίος ενθυλακώνει όλη εκείνη την πληροφορία που είναι απαραίτητη για την προσομοίωση των τροποποιήσεων που συμβαίνουν στα αντικείμενα που βρίσκονται στους origin servers, έτσι ώστε να γίνει δυνατή η διεξαγωγή συμπερασμάτων σχετικά με slow hits, stale deliveries... Η δημιουργία του ModificationMap βασίζεται στην προεπεξεργασία των Web traces.

Απο τα παραπάνω προκύπτει οτι ο προσομοιωτής είναι σε μεγάλο βαθμό παραμετροποιήσιμος. Προσφέρει στον χρήση μια γκάμα απο επιλογές απαραίτητες για την εκτέλεση διαφόρων λειτουργιών και προσομοιώσεων. Σημειώνεται δε οτι παρέχει επιπλέον λειτουργίες που δεν ανήκουν στην εμβέλεια της προσομοίωσης, όπως για παράδειγμα την λήψη αρχείων Log είτε για να γίνει διακρίβωση της ορθότητας της λειτουργίας του προσομοιωτή, είτε για να μπορέσει ο χρήστης να παρατηρήσει με μεγαλύτερη λεπτομέρεια την συμπεριφορά του μηχανισμού συνέπειας. Στη συνέχεια ακολουθεί πίνακας στον οποίο δίνεται μια συνοπτική παρουσίαση των παραμέτρων/επιλογών που δίνει ο προσομοιωτής στον χρήστη:

Διεπαφή χρήστη προσομοιωτή *Vproxy*		
Παράμετρος	**Περιγραφή**	**Παρατηρήσεις**
-adaptive K, THR	Ενεργοποίηση adaptice TTL πολιτικής με fudge factor K και κατώφλι THR	Η τιμή του K ανήκει στο [0,1], το THR είναι θετικός ακέραιος
-cache TYPE, SIZE	Επιλογή της cache τύπου TYPE και μεγέθους SIZE	Το TYPE είναι inf ή lru, το SIZE είναι μεταξύ 1048576 & 8589934592L bytes
-help	Προσφέρει στον χρήστη μια συνοπτική παρουσίαση χρήσης των παραμέτρων	Δεν υποστηρίζεται προς το παρόν
Συνεχίζεται στην επόμενη σελίδα		

Πίνακας 4.1 – συνέχεια από την προηγούμενη σελίδα

Διεπαφή χρήστη προσομοιωτή *Vproxy*		
Παράμετρος	**Περιγραφή**	**Παρατηρήσεις**
-log	Ενεργοποίηση της logging λειτουργίας	
-metric METRIC	Ενεργοποίηση της μετρικής METRIC	Η METRIC μπορεί να πάρει τιμές u3 και u4
-mode MODE	Επιλογή τρόπου προσομοίωσης MODE	Υποστηρίζεται μόνο η επιλογή normal που σημαίνει οτι ακολουθείται HTTP caching [28]
-params FILE	Χρήση αρχείου FILE για την παραμετροποίηση του προσομοιωτή	
-pre-force	Ενεργοποίηση του προεπεξεργαστή	
-r | -R | -recursive DIR	Ενεργοποίηση αναδρομικής ανάγνωσης αρχείων κάτω απο τον DIR	Δεν υποστηρίζεται πλήρως
-rule RULE	Ενεργοποίηση του κανόνα παύσης RULE	Επιλογή 1e για 1/e κανόνα, επιλογή sto για Odds κανόνα, επιλογή none για κανένα κανόνα
-standard LB[, UB]	Ενεργοποίηση standard TTL πολιτικής με επιλογές LB, UB για το TTL	Με επιλογή μόνο LB απόδοση fixed TTL με τιμή LB. Με επιλογή LB και UB, απόδοση TTL με uniform κατανομή στο [LB, UB].

Πίνακας 4.1: Διεπαφή χρήστη προσομοιωτή *Vproxy*.

4.3 Σύνοψη

Συνοψίζοντας, σε αυτό το κεφάλαιο έγινε μια προσπάθεια αναλυτικής παρουσίασης του μηχανισμού διασφάλισης συνέπειας σε εξυπηρετητές κρυφής μνήμης Παγκόσμιου Ιστού, που προτείνεται και μελετάται στα πλαίσια της παρούσας διπλωματικής εργασίας, δηλαδή του *Vproxy*.

Ο *Vproxy* χρησιμοποιεί την έννοια του cacheability για να μπορέσει να υλοποιήσει έναν αποδοτικό μηχανισμό συνέπειας που να ικανοποιεί τις εξής ιδιότητες: να είναι ασθενής, σύγχρονος και validation. Μάλιστα στόχος του είναι να αξιοποιήσει τα πλεονεκτήματα ενός άλλου μηχανισμού, του *ATTL*, καταφέρνοντας ταυτόχρονα να αυξήσει ακόμα περισσότερο την συνέπεια αυτού δίχως να πληρώσει μεγάλο κόστος σε slow hits. Για να το επιτύχει αυτό πρέπει να ικανοποιήσει το tradeoff που ορίζεται απο την αποφυγή ενός slow hit και ενός stale delivery. Σε αυτό το σημείο έρχεται η έννοια του cacheability να παίξει τον καιριό της ρόλο, καθώς η έννοια αυτή ορίζει την δυνατότητα ενός αντικειμένου της cache να ικανοποιήσει μια εισερχόμενη αίτηση και ο *Vproxy* αξιοποιεί την έννοια αυτή ικανοποιώντας όλες τις εισερχόμενες αιτήσεις μέχρι το cacheability να φτάσει στην μέγιστη τιμή.

Για να επιτελέσει την παραπάνω λειτουργία ο *Vproxy* χρησιμοποιεί μια μετρική η οποία αποτιμά το cacheabillity ενός αντικειμένου βάσει συγκεκριμένων χαρακτηριστικών και έναν κανόνα παύσης ο οποίος προσπαθεί να προβλέψει την μέγιστη τιμή του cacheabillity. Ο *Vproxy* παρέχει δύο μετρικές, την u_3 και u_4, καθώς και δύο κανόνες παύσεις, τον $1/e$ και τον *Odds*.

Τέλος, υλοποιείται και ο κατάλληλος προσομοιωτής ο οποίος χρησιμοποιεί όλα τα παραπάνω για να προσομοιώσει τον *Vproxy* με κατάλληλη είσοδο δεδομένων που προέρχονται απο Web traces, καθώς και κατάλληλη παραμετροποίηση μέσω της διεπαφής χρήστη που προσφέρει.

5. ΠΡΟΣΟΜΟΙΩΣΗ

Στο κεφάλαιο αυτό θα παρουσιαστούν τα αποτελέσματα απο τις προσομοιώσεις του μηχανισμού *Vproxy* που έλαβαν χώρα στα πλαίσια της παρούσας διπλωματικής εργασίας. Θα περιγραφεί η μεθοδολογία που ακολουθήθηκε για την κατάλληλη προσομοίωση του *Vproxy* και την διεξαγωγή αξιόπιστων αποτελεσμάτων, καθώς επίσης και θα παρουσιαστούν τα απαραίτητα γραφήματα τα οποία οπτικοποιούν τα αποτελέσματα. Πιο συγκεκριμένα η διάρθρωση του κεφαλαίου είναι η ακόλουθη:

- **Ενότητα 5.1**: Η ενότητα 5.1 παρέχει το θεωρητικό υπόβαθρο που χρειάζεται να έχει κάποιος για να διεξάγει προσομοίωση ενός συστήματος ή μηχανισμού. Περιγράφει την τρέχουσα μεθοδολογία που υπάρχει, καθώς και τις λεπτομέρειες της μεθοδολογίας που ακολουθήθηκε σε αυτή την εργασία.
- **Ενότητα 5.2**: Η ενότητα 5.2 παρουσιάζει τα αποτελέσματα απο τις διάφορες προσομοιώσεις που έγιναν στα πλαίσια αυτής της εργασίας. Γίνεται κύριως σύγκριση μεταξύ του *Vproxy* και του *ATTL*, καθώς επίσης και μεταξύ των διαφόρων επιλογών του *Vproxy* σε κανόνες παύσης και μετρικές, με στόχο να βγούν τα κατάλληλα συμπεράσματα όσον αφορά την συνολική απόδοση του *Vproxy*, αλλά και την σχετική απόδοση, δηλαδή εκείνη την απόδοση που εξαρτάται απο την επιλογή του κανόνα παύσης και της μετρικής.
- **Ενότητα 5.3**: Η ενότητα 5.3 κάνει μια σύνοψη όλων όσων παρουσιάστηκαν στις προηγούμενες ενότητες.

5.1 Μεθοδολογία

Προτού παρουσιαστούν τα αποτελέσματα απο τις διάφορες προσομοιώσεις που έγιναν στα πλαίσια ελέγχου του *Vproxy*, είναι σημαντικό να παρουσιαστούν οι μέθοδοι που υπάρχουν για την αποτίμηση ενός μηχανισμού. Οι μέθοδοι αυτοί είναι οι εξής: **live measurements**, **trace μέθοδοι** και **benchmarking**.

Η πρώτη μέθοδος δεν αποτελεί ουσιαστικά μέθοδος προσομοίωσης ενός μηχανισμού, αλλά ελέγχου του σε πραγματικές συνθήκες, δηλαδή σε επίπεδο κανονικής λειτουργίας. Η χρήση live measurements, όμως, παρουσιάζει αρκετά προβλήματα, με αποτέλεσμα να μην είναι εύκολη αλλά και αποδοτική η χρήση τους. Εάν για παράδειγμα προσπαθήσει κάποιος να χρησιμοποιήσει μια τέτοια μέθοδο για την αποτίμηση της λειτουργίας ενός μηχανισμού που αφορά proxies, όπως είναι ο παρών μηχανισμός, θα υποχρεωθεί να ενορχηστρώσει κατάλληλα ή να απαιτήσει απο όλους του χρήστες του proxy, που θα συμμετάσχουν σε αυτή την αποτίμηση, να ενορχηστρώσουν κατάλληλα τους browsers τους. Επιπλέον, σε κατάσταση πραγματικής λειτουργίας ενός μηχανισμού σπανίως αναπαράγωνται έντονες συνθήκες, με αποτέλεσμα να μην μπορεί να αποτιμηθεί η απόδοση του μηχανισμού κάτω απο τέτοιες συνθήκες. Τέλος, είναι δύσκολο να επαναληφθούν τα αποτελέσματα μιας live measurement, μιας και είναι σχεδόν αδύνατο να αναπαραχθούν πλήρως οι συνθήκες κάτω απο τις οποίες είχαν ληφθεί οι προηγούμενες μετρήσεις [15].

Μια άλλη μέθοδος αποτιμήσης ενός μηχανισμού περιλαμβάνει την χρήση Web traces. Τα Web traces καταγράφουν πληροφορία όπως την χρονική στιγμή μιας αίτησης, το URL του αντικειμένου που ζητήθηκε, το μέγεθος της απάντησης, τον κωδικό κατάστασης της απάντησης κ.α. και συλλέγονται κυρίως σε **αρχεία πρόσβασης** (access log) στους proxies και τους servers[1]. Ο ορισμός για το trace που δίνεται στα πλαίσια αυτής της εργασίας, ώστε να γίνει αντιληπτή η σημασία του απο τον αναγνώστη είναι ο εξής:

> *Το trace αποτελεί την καταγραφή των ιστορικών αποτυπωμάτων της συμπεριφοράς του συστήματος σε μια δεδομένη χρονική περίοδο η οποία προφανώς ανήκει στον παρελθόν. Λόγος υπάρξης ενός trace είναι η κατα το δυνατότερο ακριβότερη μίμηση της συμπεριφοράς του συστήματος της δεδομένης χρονικής περιόδου, στο μέλλον.*

Απο τον παραπάνω ορισμό προκύπτουν δύο συμπεράσματα: **1)** οτι ένα trace δεν μπορεί να είναι γενικά αντιπροσωπευτικό και **2)** ενδεχόμενη αλλοίωση ενός trace μπορεί να οδηγήσει σε εσφαλμένα αποτελέσματα.

Μια προσομοίωση, λοιπόν, που χρησιμοποιεί traces για να αποτιμήσει την συμπεριφορά ενός Web μηχανισμού, ονομάζεται trace-driven προσομοίωση. Άλλη σημαντική χρήση των traces περιλαμβάνει την στατιστική ανάλυση των δεδομένων ενός συστήματος, γι αυτό και μια τέτοια διαδικασία ονομάζεται trace-driven ανάλυση.

Ενώ, λοιπόν, η προσομοίωση με την χρήση traces είναι μια μέθοδος αρκετά αποδοτική, το βασικό πρόβλημα που παρουσιάζει είναι αυτό που επισημάνθηκε λίγο πιο πάνω και συνίσταται στην μη γενική αντιπροσώπευση μια συμπεριφοράς. Για παράδειγμα το ίδιο το σύστημα μπορεί να παρουσιάζει διαφορετική συμπεριφορά μια χρονική περιόδο και διαφορετική μια άλλη. Επιπλέον, Web traces τα οποία προέρχονται απο ένα πανεπιστημιακό δίκτυο μπορεί να παρουσιάζουν ολότελα διαφορετική συμπεριφορά απο τα Web traces ενός επιχειρησιακού δικτύου.

Η τρ ίτη μέθοδος που χρησιμοποιείται για την αποτίμηση της απόδοσης ενός συστήματος είναι το benchmarking, η δημιουργία δηλαδή ενός συνθετικού και τεχνητού **φόρτου εργασίας** (workload), που απο τη μία προσπαθεί να μιμηθεί έναν πραγματικό φόρτο εργασίας, ενώ απο την άλλη να δοκιμάσει το σύστημα κάτω απο όσο το δυνατόν πιο έντονες συνθήκες. Η μέθοδος αυτή προσπαθεί να αντιμετωπίσει το βασικό πρόβλημα της προηγούμενης μεθόδου, δηλαδή το πρόβλημα της αντιπροσωπευτικής συμπεριφοράς σε γενικό επίπεδο. Για να γίνει, λοιπόν, δυνατή η δημιουργία ενός σωστού benchmark απαιτείται η μελέτη των Web traces, ώστε να διεξαχθούν κατάλληλα στοιχεία όσον αφορά τη συμπεριφορά ενός συστήματος μέσω της καταγραφής του **μέσου** (mean) και **διαμέσου** (median) μεγέθους των αντικειμένων που διακινούνται σε αυτό, της κατανομής της δημοφιλίας των αντικειμένων κ.λπ. Αυτή η διαδικασία είναι επίπονη και πολλές φορές μπορεί να οδηγήσει σε συμπεράσματα που να μην είναι όντως αντιπροσωπευτικά. Για το λόγο αυτό και υπάρχει μεγάλη βιβλιογραφία γύρω απο το ζήτημα προσομοίωσης του Web και ακόμα περισσότερο γύρω απο το benchmarking [66–72].

Στην παρούσα εργασία, χρησιμοποιείται η δεύτερη μέθοδος για την προσομοίωση του *Vproxy*. Συγκεκριμένα χρησιμοποιούνται τα Web traces της Digital Equipment Corporation, DEC, τα οποία ελήφθησαν σε μια περίοδο τρεισήμισι εβδομάδων το έτος 1996 και περιλαμβάνουν

[1] Υπάρχει η δυνατότητα συλλογής τέτοιας πληροφορίας και σε επίπεδο browser, με την κατάλληλη ρυθμισή του.

24.477.674 **αναφορές** (references) [73]. Σε αυτά τα traces έγινε μια σχετική επεξεργασία, καθώς αφαιρέθηκαν πεδία των αναφορών που δεν ήταν σημαντικά για την προσομοίωση, ενώ προστέθηκαν κάποια νέα πεδία, όπως το πεδίο header το οποίο ουσιαστικά προσφέρει τις απαραίτητες Cache-Control οδηγίες για τον χαρακτηρισμό του cacheability ενός αντικειμένου. Επειδή οι προσομοιώσεις ακολούθησαν πλήρως τις οδηγίες του HTTP caching [3] είναι απαραίτητο σε αυτό το σημείο να επισημανθεί οτι μεγάλο μέρος των αντικειμένων που διακινούνται στο Web είναι **uncacheable** [3, 74, 75] και συνακόλουθα μεγάλος μέρος των αναφορών στα Web traces αφορούν τέτοια αντικείμενα και αυτό απαιτούσε την κατάλληλη διαχείριση. Η επεξεργασία βέβαια περιελάμβανε και άλλες διορθώσεις, όπως την αναγνώριση **lost** και **interrupted** συνδέσεων, τη διόρθωση τιμών στις *Last-Modified* κεφαλίδες, καθώς και πολλές άλλες που περιγράφονται στα πλαίσια της τεχνικής τεκμηρίωσης.

5.2 Αποτελέσματα

Πρίν γίνει η παρουσίαση και ανάλυση των γραφημάτων, που απεικονίζουν τα αποτελέσματα των διαφρόρων προσομοιώσεων, είναι απαραίτητο να διευκρινιστούν καθώς και να επισημανθούν κάποια πράγματα. Υπενθυμίζεται οτι ο προσομοιωτής είναι γραμμένος στη γλώσσα προγραμματισμού Java και οτι χρησιμοποιεί ως είσοδο Web traces για να μιμηθεί την συμπεριφορά ενός πραγματικού συστήματος. Τα Web traces προέρχονται απο την DEC, ελήφθησαν σε μια χρονική περίοδο τρεισήμισι εβδομάδων το 1996 και περιέχουν περίπου 25 εκατομύρια εγγραφές. Οι διάφορες προσομοιώσεις έγιναν σε εκείνο το mode του προσομοιωτή που ακολουθεί πλήρως τους κανόνες του HTTP caching κι έτσι μεγάλος αριθμός αιτήσεων αξιολογήθηκε ως uncacheable, με αποτέλεσμα το **ποσοστό ευστοχίας** (hit ratio) επι του συνόλου των αιτήσεων να κυμαίνεται στο 30 με 40%. Τέλος χρησιμοποιήθηκε μια LRU cache[2] με μέγεθος 4 gigabytes, GB.

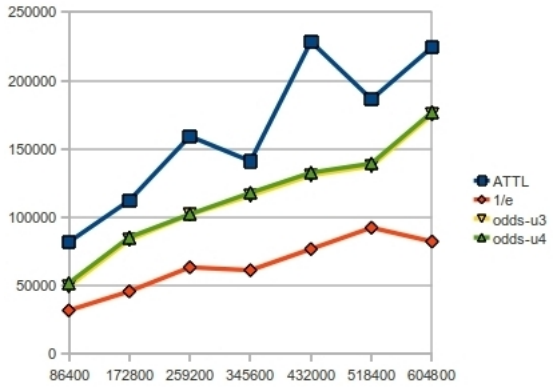
Στην πρώτη σειρά προσομοιώσεων ελέγχθηκε η απόδοση του *Vproxy* χρησιμοποιώντας τους κανόνες παύσης και τις μετρικές που διαθέτει και σε σύγκριση με τον $ATTL$. Πιο συγκεκριμένα μετρήθηκε το κόστος του $ATTL$ σε stale deliveries, slow hits και consistency misses, και συγκρίθηκε με αυτό του *Vproxy* για τις περιπτώσεις όπου χρησιμοποιεί $1/e$ κανόνα παύσης, *Odds* κανόνα παύσης με μετρική u_3 και *Odds* κανόνα παύσης με μετρική u_4. Σε όλες τις παραπάνω περιπτώσεις ο *Vproxy* «πάτησε» πάνω στον $ATTL$, δηλαδή χρησιμοποίησε τον $ATTL$ ως υπόστρωμμα[3] .

Στα Σχήματα 5.1(α′) - (γ′) παρουσιάζεται το κόστος, σε συνάρτηση με το κατώφλι της απόδοσης TTL και με σταθερό $k = 0.2$. Υπενθυμίζεται οτι κατα την απόδοση TTL στην adaptive πολιτική, δίνεται ένα κατώφλι το οποίο καθορίζει το μέγιστο TTL που μπορεί να λάβει ένα αντικείμενο το οποίο τοποθετείται σε μια cache, έτσι ώστε να διασφαλιστεί πως για όλα τα αντικείμενα θα ελεγχθεί κάποια στιγμή η συνεπειά τους μέσω της κατάλληλης επικοινωνίας με τον origin server.

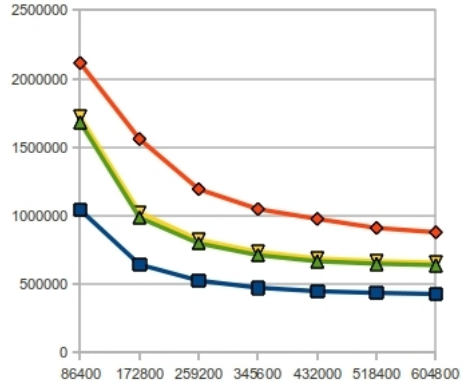
Η πρώτη σημαντική παρατήρηση που προκύπτει σχετίζεται με τα stale deliveries. Παρατηρείται, λοιπόν, μια σημαντική βελτίωση στον αριθμό των stale deliveries. Την καλύτερη απόδοση παρουσιάζει ο *Vproxy* μηχανισμός με κανόνα $1/e$, καθώς το ποσοστό των stale deliveries επι

[2]LRU cache είναι μια cache η οποία χρησιμοποιεί ως πολιτική αντικατάστασης τον LRU αλγόριθμο.

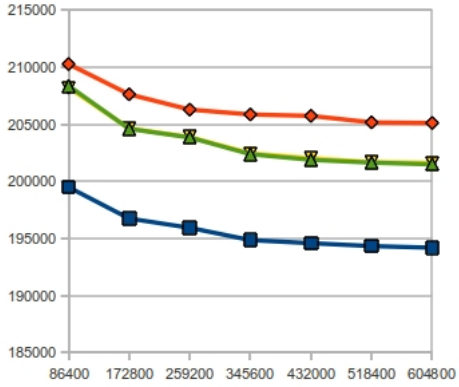
[3]Πιο συγκεκριμένα χρησιμοποιήσε τον $ATTL$ ως πολιτική απόδοσης TTL τιμών στα αντικείμενα της cache.



(α′) Stale deliveries



(β′) Slow hits



(γ′) Consistency misses

Σχήμα 5.1: Μέτρηση κόστους για διάφορες τιμές threshold.

του συνόλου των αιτήσεων κυμαίνεται μεταξυ 0.13% και 0.38%. Στη συνέχεια ακολουθεί ο *Vproxy* με *Odds* κανόνα και u_3 μετρική, όπου το ποσοστό κυμαίνεται απο 0.2% εώς 0.72%[4] και τελευταίος έρχεται ο *ATTL* μηχανισμός με απόδοση που βρίσκεται μεταξύ του 0.43% και του 0.95%.

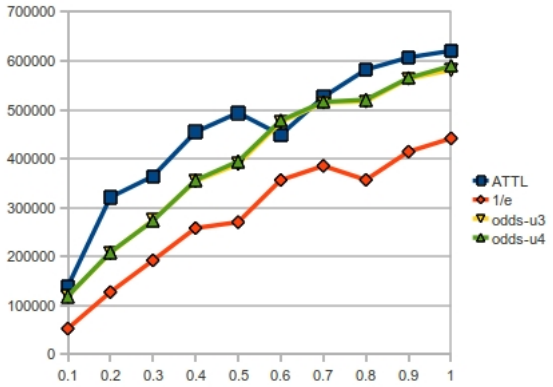
Τα παραπάνω αποτελέσματα είναι λογικά. Πρώτον ο $1/e$ κανόνας είναι ο πιο επιφυλακτικός κανόνας, με αποτέλεσμα να ελέγχει την εγκυρότητα ενός αντικειμένου αρκετά πιο νωρίς απο τους υπόλοιπους μηχανισμούς. Απο την άλλη και ο *Odds* κανόνας είναι επιφυλακτικός, μιας και θεωρεί ως ορίζοντα των παρατηρησεών του (δηλαδή της αποτίμησης της μετρικής κάθε φορά που μια αίτηση καταφτάνει) το TTL που έχει ήδη καθοριστεί απο τον *ATTL*. Συνεπώς ο *Vproxy* μηχανισμός και για τους δύο κανόνες παύσης (Odds κανόνα και ο $1/e$) τείνει να ελέγχει την εγκυρότητα ενός αντικειμένου πριν λήξει το TTL, με αποτέλεσμα να ανιχνεύει περισσότερες τροποποιήσεις απο τον *ATTL* μηχανισμό. Τα συμπεράσματα αυτά επιβεβαιώνονται και απο τα άλλα δύο σχήμα (Σχήμα 5.1(β′) και Σχήμα 5.1(γ′)), όπου φαίνεται ξεκάθαρα οτι ο *Vproxy* πληρώνει το κέρδος της αποφυγής των stale deliveries, με το κόστος των περισσότερων slow hits και consistency misses.

Βέβαια το κόστος που σχετίζεται με τα consistency misses είναι ευπρόσδεκτο διότι αυξάνει τον βαθμό συνέπειας. Όσον αφορά τα slow hits, την καλύτερη απόδοση μετά τον *ATTL* την

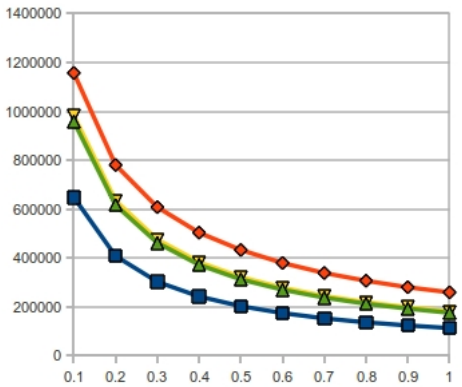
[4]Ο Odds κανόνας με μετρική u_4 παρουσιάζει μια αμελητέα αύξηση στον αριθμό των stale deliveries

παρουσιάζει ο *Odds* κανόνας με μετρική u_4. Παρόλο που ο *Odds* κανόνας συμβάλει σε περίπου διακόσες χιλιάδες περισσότερα slow hits, αύξηση της τάξης του 53% επι των slow hits του *ATTL*, η αποδοσή του κινείται μέσα σε αποδεκτά όρια, μιας και το κόστος των slow hits είναι δυσανάλογα μικρότερο απο το κόστος ενός miss και επιπλέον απο το κόστος που προκαλεί η ασυνέπεια. Άλλωστε η μείωση των stale deliveries, με την αντίστοιχη αύξηση των consistency misses δεν μπορεί να επιτευχθεί δίχως την αντίστοιχη αύξηση των slow hits. Το ζητούμενο είναι αυτή η μείωση να μην οδηγήσει σε πολύ μεγάλη αύξηση του αριθμού των slow hits, όπως συμβαίνει στον $1/e$ κανόνα όπου παρατηρείται αύξηση του αριθμού των slow hits εως και 142%[5].

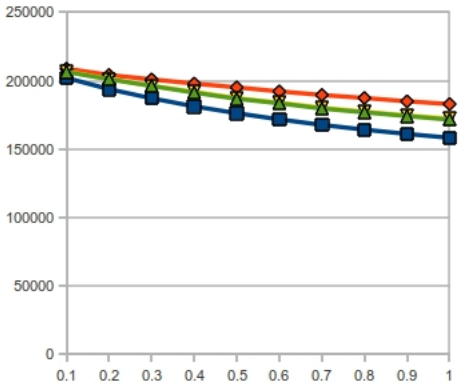
Στη συνέχεια παρουσιάζονται τα αποτελέσματα της επόμενης σειράς προσομοιώσεων, κατα την οποία ελέγχθηκε η απόδοση του *Vproxy* και των παραλλαγών του σε σύγκριση με τον *ATTL*, βάσει του k το οποίο ορίζει το ποσοστό της ηλικίας ενός αντικειμένου που θα θεωρηθεί ως TTL. Το k θα πάρεις τις τιμές $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.91.0\}$. Για να μην επηρεαστεί η απόδοση TTL απο την τιμή του threshold, θα δοθεί μια τεράστια τιμή σε αυτό. Στα Σχήματα 5.2(α′) - (γ′) παρουσιάζονται τα αποτελέσματα.



(α′) Stale deliveries



(β′) Slow hits



(γ′) Consistency misses

Σχήμα 5.2: Μέτρηση κόστους για διάφορες τιμές k.

Και σε αυτή τη σειρά προσομοιώσεων τα αποτελέσματα είναι σχεδόν αναμενόμενα. Και πάλι ο καλύτερος μηχανισμός συνέπειας όσον αφορά τα stale deliveries είναι ο *Vproxy* με κανόνα $1/e$. Όσον αφορά τον *Vproxy* με *Odds* κανόνα, αυτός παρουσιάζει καλύτερη απόδοση χρησι-

[5]Ισχύει για την περίπτωση όπου $k = 0.2$ και $threshold = 172800$.

μοποιώντας την u_3 μετρική παρα την u_4, αλλά και πάλι η διαφορά είναι αμελητέα, καθώς είναι της τάξης των μερικών χιλιάδων αιτήσεων. Σε σύγκριση με τον $ATTL$ μηχανισμό, ο *Vproxy* με κανόνα παύσης τον *Odds* είναι σαφώς καλύτερος για τιμές του $k = \{0.2, 0.3, 0.4, 0.5, 0.8, 0.9\}$, ενώ για τις άλλες τιμές είτε παρουσιάζει χειρότερη απόδοση, όπως για παράδειγμα στην τιμή $k = 0.6$, είτε ελαφρώς βελτιωμένη. Πάντως για τιμές του k μικρότερες ή ίσες του 0.5, που είναι οι τιμές που χρησιμοποιούνται συνήθως, η αποδοσή του είναι συνολικά καλύτερη απο αυτή του $ATTL$.

Όσον αφορά τα consistency misses τα αποτελέσματα είναι αναμενόμενα. Υπάρχει μια ελαφριά αύξηση του αριθμού τους στον $1/e$, καθώς και στον Odds κανόνα, η οποία γίνεται πιο αισθητή καθώς το k μεγαλώνει. Πάντως σε όλες τις περιπτώσεις το ποσοστό των consistency misses επι του συνόλου των αιτήσεων κινείται κάτω απο το 1%.

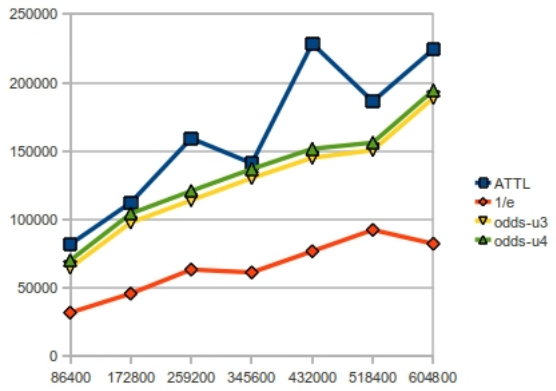
Στα slow hits και πάλι την καλύτερη απόδοση παρουσιάζει ο $ATTL$, ενώ την χειρότερη ο *Vproxy* με $1/e$ κανόνα. Ο *Vproxy* για *Odds* κανόνα και για τις δύο μετρικές, παρουσιάζει μια απόδοση που θα μπορούσε να χαρακτηριστεί ως η μέση απόδοση των δύο άλλων μηχανισμών. Καθώς βέβαια το k προσεγγίζει το 1, παρατηρείται μια ταχεία σύγκλιση προς τον αριθμό των slow hits του $ATTL$.

Οι επόμενες δύο σειρές προσομοιώσεων αφορούν κυρίως τον *Odds* κανόνα. Στο προηγούμενο κεφάλαιο περιγράφηκε ο *Odds* κανόνας που χρησιμοποιείται απο τον *Vproxy* ο οποίος αποτελεί μια ειδική περίπτωση του *Odds* αλγορίθμου για συνεχή χρόνο και τυχαίες αφίξεις Poisson διεργασίας. Σε αυτό τον κανόνα καίριο λόγο παίζει ο μέσος ρυθμός αφίξης των αιτήσεων της Poisson διεργασίας λ. Ένα απο τα συνηθέστερα προβλήματα της στατιστικής ανάλυσης είναι η εκτίμηση του λ. Υπάρχουν πολλές μέθοδοι για την εκτιμησή του. Επειδή ο συγκεκριμένος μηχανισμός κάνει ακολουθιακή ανάλυση των δεδομένων, πρέπει και οι μέθοδοι εκτίμησης του λ να είναι ακολουθιακές. Ένα σημαντικό βιβλίο που ασχολείται με αυτά τα ζητήματα είναι το *Sequential Estimation* των Ghosh, Mukhopadhyay και Sen [76]. Βέβαια, σε όλες αυτές τις μεθόδους, στόχος είναι η εύρεση ενός κανόνα παύσης για την βέλτιστη εκτίμηση του λ, με ταυτόχρονη ελαχιστοποιήση του αντίστοιχου κόστους που σχετίζεται με την λήψη ακολουθιακών παρατηρήσεων.

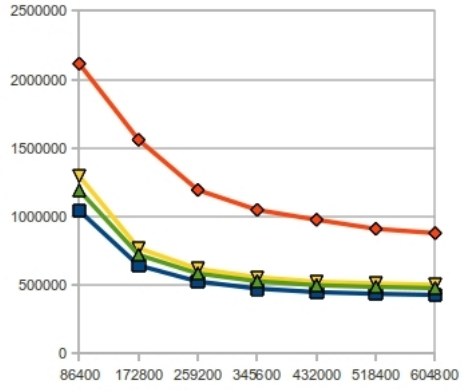
Στον *Vproxy*, όμως, το κόστος αυτό δεν υφίσταται. Δηλαδή δεν υπάρχει κάποιο επιπλέον κόστος απο την διαρκή παρατήρηση και αναπροσαρμογή του λ. Αυτό, όμως, που μπορεί να συμβεί και παρατηρήθηκε στα πλαίσια των προσομοιώσεων είναι η απότομη αλλαγή του ρυθμού αφίξεων των αιτήσεων προς ένα αντικείμενο με επακόλουθο τον επηερασμό του κανόνα παύσης. Για να γίνει αυτό αντιληπτό διεξήχθη το εξής πείραμα: στον *Odds* κανόνα παύσης προστέθηκε στον ορίζοντα μια επιπλέον τιμή, η $\frac{1}{\lambda}$. Η τιμή $\frac{1}{\lambda}$ αντιπροσωπεύει τον μέσο **χρόνο μεταξύ δύο αφίξεων** (interarrival time). Επομένως, αυξάνοντας τον ορίζοντα του *Odds* κανόνα κατα $\frac{1}{\lambda}$, ισοδυναμεί με αυξησή του κατα τον αναμενόμενο χρόνο άφιξης της πρώτης αίτησης μετά την λήξη του TTL. Τι ελέγχεται, όμως, μέσω αυτού του πειράματος;

Αυτό που ελέγχεται είναι ο βαθμός στον οποίο επηρεάζει το λ τον *Odds* κανόνα. Για παράδειγμα παρατηρήθηκε το εξής φαινόμενο: αρχικά γινόντουσαν αιτήσεις προς ένα αντικείμενο με χαμηλό ρυθμό άφιξης και στη συνέχεια, λίγο πριν λήξει το TTL του άρχισαν να γίνονται αιτήσεις πιο συχνά, ο ρυθμός άφιξης όμως δεν προλάβαινε να προσαρμοστεί με αποτέλεσμα ο *Odds* να ενεργοποιείται, να στέλνει αίτηση επικύρωσης προς τον origin server και να συμβάλει έτσι σε ένα slow hit συμβάν. Επεκτείνοντας, λοιπόν, τον ορίζοντα κατα τον $\frac{1}{\lambda}$ παράγοντα,

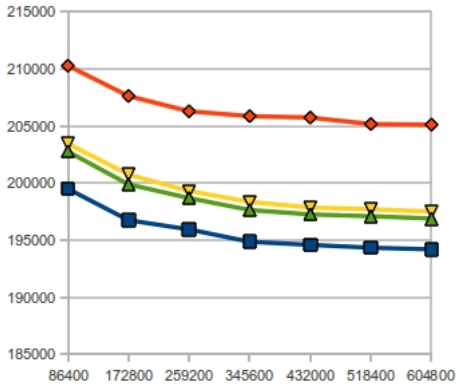
επιτυγχάνεται η εξάλειψη του παραπάνω φαινομένου. Στα Σχήματα 5.3(α′) - (γ′) και 5.4(α′) - (γ′) παρουσιάζονται τα αποτελέσματα, με την χρήση του παράγοντα $\frac{1}{\lambda}$.



(α′) Stale deliveries



(β′) Slow hits



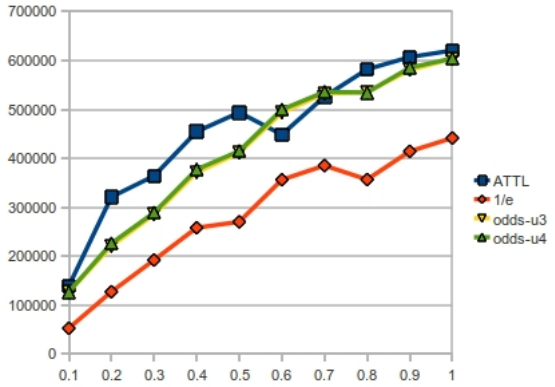
(γ′) Consistency misses

Σχήμα 5.3: Μέτρηση κόστους για διάφορες τιμές threshold και παράγοντα $\frac{1}{\lambda}$.

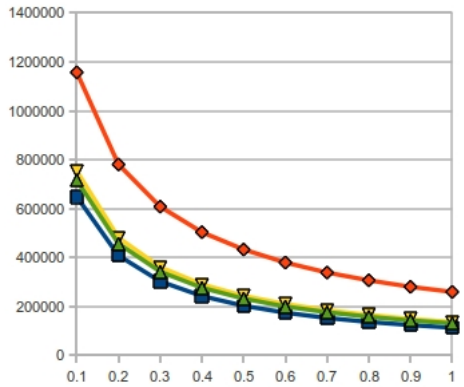
Στα Σχήματα 5.3(α′) - (γ′) παρουσιάζονται τα αποτελέσματα για διάφορες τιμές threshold στον *ATTL*. Αυτό που παρατηρείται στα νέα αποτελέσματα είναι οτι ο *Vproxy* με *Odds* κανόνα συνεχίζει να έχει καλύτερη απόδοση όσον αφορά τα stale deliveries σε σχέση με τον *ATTL*, αν και το χάσμα έχει μειωθεί σε σχέση με πριν. Το εντυπωσιακότερο, όμως, αποτέλεσμα σχετίζεται με τον αριθμό των slow hits, καθώς η ψαλίδα μεταξύ του *Vproxy* με *Odds* κανόνα και του *ATTL* έχει κλείσει σε πολύ μεγάλο βαθμό· ειδικά όταν γίνεται χρήση της u_4 μετρικής ο αριθμός των slow hits είναι αυξημένος μόνο κατα 10% σε σχέση με τον ATTL, την ίδια στιγμή που αριθμός των stale deliveries έχει μειωθεί κατα 13.37%[6]. Παρατηρείται, λοιπόν, με τη χρήση του παράγοντα $\frac{1}{\lambda}$ μια πιο ισοσκελισμένη συμπεριφορά.

Στα 5.4(α′) - (γ′) παρουσιάζονται τα αποτελέσματα για τις διάφορες τιμές του k. Κι εδώ τα αποτελέσματα είναι τα αναμενόμενα. Δεν παρατηρείται καμιά αλλαγή στην συμπεριφορά εξόν της εξομάλυνσης που περιγράφηκε και προηγουμένως του αριθμού των slow hits στον *Odds*

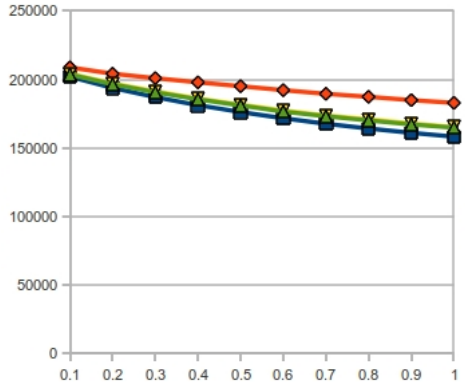
[6]Αυτά τα αποτελέσματα αφορούν την περίπτωση που το *threshold* έχει τιμή 604800. Ενώ για την περίπτωση όπου το *threshold* έχει τιμή 432000, η μείωση των stale deliveries αγγίζει το 34% με αντίστοιχη αύξηση των slow hits κατα 11.22%. Επισημαίνεται οτι τα αποτελέσματα αυτά αφορούν τον *Vproxy* με *Odds* κανόνα και u_4 μετρική και υπόστρωμα τον *ATTL* με $k = 0.2$.



(α′) Stale deliveries



(β′) Slow hits



(γ′) Consistency misses

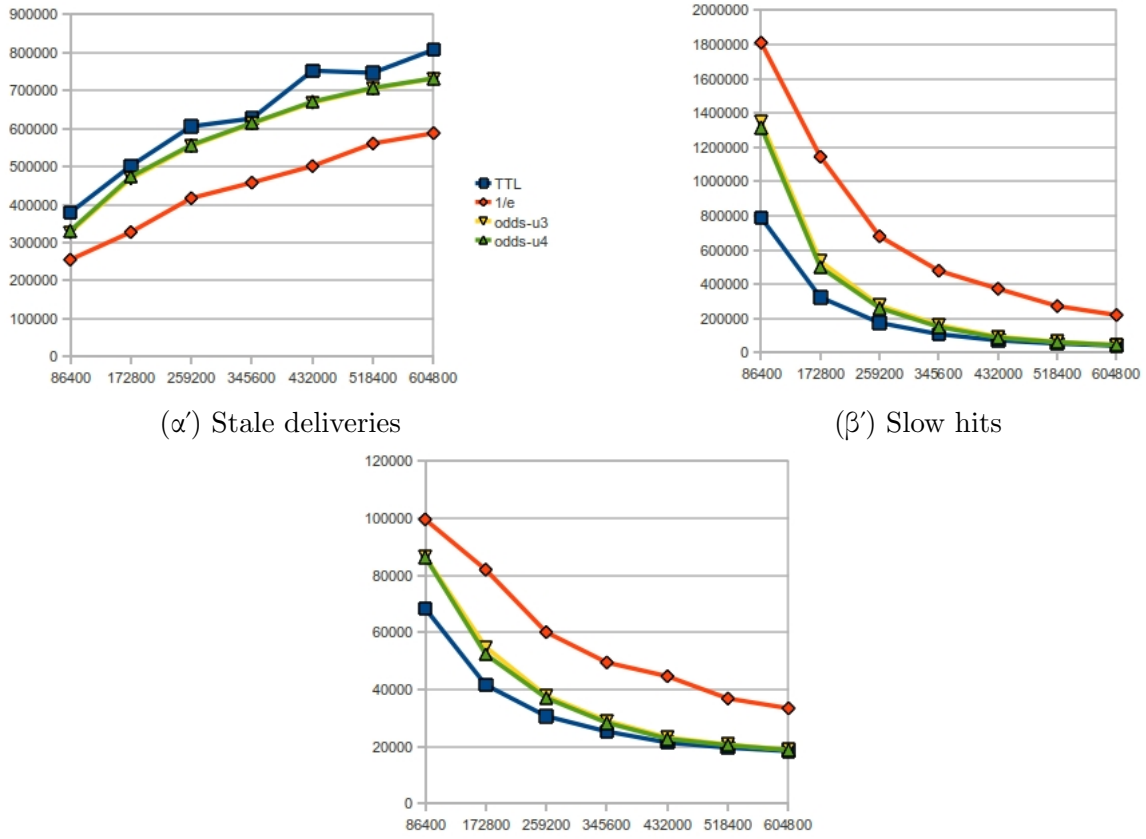
Σχήμα 5.4: Μέτρηση κόστους για διάφορες τιμές k και παράγοντα $\frac{1}{\lambda}$.

κανόνα, καθώς και της ελλάτωσης της διαφοράς των staled deliveries απο τον *ATTL*.

Γενικά αυτή η σειρά προσομοιώσεων μας δίνει μια σημαντική εικόνα όσον αφορά την επιρροή του λ στην απόδοση του *Odds* κανόνα παύσης, κάτι που πρέπει να ληφθεί υπόψιν σε μελλοντικές επεκτάσεις ή και διορθώσεις του *Vproxy* μηχανισμού.

Τέλος, παρουσιάζεται μια σειρά απο προσομοιώσεις όπου δεν χρησιμοποιείται ο *ATTL* ως υπόστρωμμα του *Vproxy*, αλλά αντιθέτως χρησιμοποιούνται σταθερά TTLs. Στα σχήματα 5.5(α′) - (γ′) παρουσιάζονται τα κόστη που προκύπτουν απο την χρήση μιας τέτοιας πολιτικής απόδοσης TTL.

Το συμπέρασμα που εύκολα προκύπτει απο τα παραπάνω σχήματα είναι η άσχημη απόδοση του *Vproxy*, καθότι δεν καταφέρνει να βελτιώσει σε μεγάλο βαθμό την αναμενόμενη άσχημη απόδοση του μηχανισμού συνέπειας που χρησιμοποιεί μια πάγια τακτική απόδοσης TTL. Η μείωση του αριθμού των stale deliveries παρότι εμφανής δεν είναι αρκετή, ώστε να βελτιώσει τον αριθμό αυτόν σε απόλυτη τιμή· ο αριθμός των stale deliveries παραμένει αρκετά υψηλός. Όσον αφορά τα slow hits και τα consistency misses οι διαφορές είναι μικρές και εντοπίζονται κυρίως στις μικρές τιμές του TTL. Άρα είναι ολοφάνερο οτι ο *Vproxy* δεν μπορεί να λειτουργήσει αποδοτικά δίχως την χρήση μια ορθολογικής πολιτικής απόδοσης TTL σαν αυτή του *ATTL*.



(α′) Stale deliveries

(β′) Slow hits

(γ′) Consistency misses

Σχήμα 5.5: Μέτρηση κόστους για διάφορες τιμές TTL.

5.3 Σύνοψη

Οι μέθοδοι που χρησιμοποιούνται στις μέρες μας για την αποτίμηση ενός μηχανισμού είναι τρείς· live measurments, web traces και benchmarking. Απο όλες αυτές τις μεθόδους η ακριβέστερη μα και λιρότερο εφαρμόσιμη είναι η πρώτη. Η δεύτερη μέθοδος, η μέθοδος δηλαδή προσομοιώσης ενός μηχανισμού με την χρήση web traces είναι μια αρκετά ακριβής και εύκολα εφαρμόσιμη μέθοδος, παρουσιάζει όμως το σημαντικό πρόβλημα της γενίκευσης των αποτελεσμάτων, μιας και τα χαρακτηριστικά ενός trace αφορούν την συμπεριφορά ενός συγκεκριμένου συστήματος, μιας δεδομένης χρονικής περιόδου. Τα προβλήματα αυτά προσπαθεί να επιλύσει η τρίτη μέθοδος, στην οποία γίνεται χρήση κατάλληλων benchmarks δηλαδή προγραμμάτων που προσπαθούνε να προσομοιώσουν την συμπεριφορά ενός συστήματος κάτω απο έντονες συνθήκες, αξιοποιώντας τη μετα-πληροφορία που εξάγουν απο κατάλληλα Web traces. Βέβαια και στην τρίτη μέθοδο τίθεται ξανα το ερώτημα κατα πόσο η πληροφορία και η μετα-πληροφορία που χρησιμοποιείται και αξιοποιείται απο τα benchmarks είναι χαρακτηριστική ενός συστήματος, και ακόμα περισσότερο αντιπροσωπευτική.

Η μέθοδος που χρησιμοποιήθηκε στον προσομοιωτή της παρούσας εργασίας είναι η δεύτερη. Έγινε, λοιπόν, κατάλληλη επεργασία και χρήση Web traces που προέρχονται απο την DEC και συγκρίθηκε η απόδοση του *Vproxy* με τις διάφορες επιλογές του σε μετρικές και κανόνες

παύσης, με αυτή του *ATTL*, αλλά και ενός απλοϊκού μηχανισμού που χρησιμοποιεί σταθερά TTLs.

Απο τις προσομοιώσεις προέκυψε το εξής ενδιαφέρον αποτέλεσμα: όταν ο *Vproxy* χρησιμοποιεί ως υπόστρωμα τον *ATTL*, χρησιμοποιεί δηλαδή τον *ATTL* ως μηχανισμό απόδοσης TTL στα αντικείμενα της cache, έχει καλύτερη απόδοση απο τον *ATTL* ως μηχανισμό συνέπειας, για κάθε επιλογή μετρικής και κανόνα παύσης. Η καλύτερη απόδοση συνίσταται στην μείωση του αριθμού των stale deliveries, στην μικρή αύξηση των consistency misses και στην ανεκτή αύξηση των slow hits, σε σχέση με τον *ATTL*[7]. Η αύξηση άλλωστε των slow hits είναι αναπόφευκτη μιας και είναι αποτελέσμα του tradeoff μεταξύ συνέπειας και συμφόρησης δικτύου που προσπαθεί κάθε ασθενής μηχανισμός να επιλύσει. Επιπλέον, το κόστος των slow hits είναι δυσανάλογα μικρό με αυτό των άλλων συμβάντων[8].

Το άλλο ενδιαφέρον αποτέλεσμα, που προέκυψε απο τις προσομοιώσεις, σχετίζεται με την περίπτωση χρήσης ως πολιτική απόδοσης TTL στα αντικείμενα της cache του απλοϊκού μηχανισμού. Σε αυτή την περίπτωση η απόδοση του *Vproxy* δεν ήταν η επιθυμητή, καθότι δεν κατάφερε σε καμία περίπτωση να βελτιώσει ικανοποιητικά το μεγάλο κόστος που παρήγαγε ο απλοΪκός μηχανισμός συνέπειας σε stale deliveries. Αυτό το αποτέλεσμα κατέδειξε με τον πλέον έντονο τρόπο την ισχυρή σύνδεση του *Vproxy* και κυρίως των κανόνων παύσης που διαθέτει στην φαρέτρα του, με την πολιτική απόδοσης TTL. Με απλά λόγια η απόδοση του *Vproxy* δεν είναι ικανοποιητική εάν η πολιτική απόδοσης TTL δεν είναι ορθολογική.

[7]Για παράδειγμα για $k = 0.2$ και $threshold = 432000$ παρατηρείται μείωση των stale deliveries περίπου 43%, με αντίστοιχη αύξηση των consistency misses 3.83% και αύξηση των slow hits 53.47%. Τα αποτελέσματα αυτά αφορούν τον *Vproxy* με *Odds* κανόνα παύσης και u_3 μετρική.

[8]Εξαιρείται βέβαια το fast hit, το οποίο δεν ενθυλακώνει κάποιο stale delivery.

6. ΣΥΜΠΕΡΑΣΜΑΤΑ

Στην παρούσα διπλωματική εργασία μελετήθηκε το πρόβλημα της διασφάλισης συνέπειας στον caching μηχανισμό του Web. Πιο συγκεκριμένα μελετήθηκε το πρόβλημα αυτό για την περίπτωση των cache servers, δηλαδή των proxies εκείνων που βρίσκονται ανάμεσα στους clients και τους origin servers και παρέχουν υπηρεσίες caching προσπαθώντας απο τη μια να μειώσουν το latency στις απαντήσεις που δέχονται οι clients, ενώ απο την άλλη να ελαχιστοποιήσουν τον φόρτο εργασίας που δέχεται ένας origin server και συνάμα να περιορίσουν την συμφόρηση του δικτύου, καθώς και την κατανάλωση bandwidth. Σκοπός όλου αυτού του μηχανισμού είναι η βελτίωση της απόδοσης του Web, κυρίως όμως η εξασφάλιση της επεκτασιμότητας του Διαδικτύου, αφού αποτελεί την αναγκαία συνθήκη για την προσδοκώμενη βελτίωση της απόδοσης.

Για να είναι αποτελεσματικός ένας Web caching μηχανισμός, πρέπει να επιτυγχάνει όλους τους παραπάνω στόχους και ταυτόχρονα να εξασφαλίζει μεγάλο βαθμό συνέπειας στα δεδομένα του. Άλλωστε αυτό είναι και το μεγαλύτερο πρόβλημα που αντιμετωπίζει οποιοσδήποτε caching μηχανισμός, το οποίο στην περίπτωση του Web καθίσταται ακόμα πιο πολύπλοκο. Υπενθυμίζεται οτι το πρόβλημα της συνέπειας σε έναν Web caching μηχανισμό συνίσταται στην διασπορά μεγάλου αριθμού αντιγράφων κάποιου αντικειμένου σε ένα μεγάλο αριθμό Web caches και στην ανάγκη κατάλληλης ενημερώσης όλων αύτων σε περίπτωση τροποποίησης του αντικειμένου, ώστε οι χρήστες του Web να μην βρεθούν στη δυσάρεστη θέση να λαμβάνουν έωλα αντικείμενα ή ακριβέστερα να λαμβάνουν έωλα αντικείμενα για μεγάλο χρονικό διάστημα.

Αυτή τη στιγμή υπάρχουν αρκετοί μηχανισμοί που προσπαθούν να λύσουν το πρόβλημα της συνέπειας με έναν αποδοτικό τρόπο. Ο μηχανισμός που ευρέως χρησιμοποιείται στο Web είναι ο *ATTL*, ο οποίος λειτουργεί μέσω της χρήσης κατάλληλα καθορισμένων TTLs και της αποστολής στον origin server σύγχρονων αιτήσεων επικύρωσης μετά την λήξη αυτών, προσφέροντας έτσι ένα μοντέλο διασφάλισης ασθενούς συνέπειας. Η απόδοση του μηχανισμού αυτού είναι αρκετά καλή, καθώς επιτυγχάνει χαμηλό αριθμό stale deliveries με αντίστοιχα χαμηλό αριθμό slow hits.

Ο μηχανισμός που προτάθηκε σε αυτή την εργασία και μελετήθηκε ονομάζεται *Vproxy*, και είναι ένας μηχανισμός που ικανοποιεί όλες τις ιδιότητες του *ATTL* μηχανισμού[1]. Ο *Vproxy* λειτουργεί κάνοντας εφαρμογή της θεωρίας Βέλτιστης Παύσης και αυτό που συγκεκριμένα μελετάται είναι η αποδοτικότητα ενός τέτοιου μηχανισμού. Στις προσομοιώσεις που διεξήχθησαν και οι οποίες περιγράφηκαν στο προηγούμενο κεφάλαιο βγήκαν ορισμένα σημαντικά συμπεράσματα όσον αφορά την μελέτη του *Vproxy*, καθώς και κάποιες ιδέες που μπορούν να σκιαγραφήσουν τις μελλοντικές κατευθύνσεις παραπέρα επέκτασης του.

Κατ΄ αρχάς παρατηρήθηκε μια αξιόλογη συμπεριφορά του μηχανισμού, όταν αυτός λειτούργησε έχοντας ως υποστρωμμά του τον *ATTL*[2]. Ο *Vproxy* παρουσίασε τεράστια μείωση στον αριθμό των stale deliveries σε σχέση με τον *ATTL*, όταν έκανε χρήση του $1/e$ κανόνα. Επειδή όμως ο κανόνας αυτός παρουσιάσε μεγάλη αύξηση στον αριθμό των slow hits, καθώς επίσης

[1] Είναι δηλαδή ασθενής, σύγχρονος και validation μηχανισμός.

[2] Η λειτουργία του *Vproxy* με υπόστρωμμα τον *ATTL* συνεπάγεται την χρήση του *ATTL* για τον καθορισμό ενός TTL και στη συνέχεια την επικείμενη πρόβλεψη συνέπειας απο τον *Vproxy*.

είναι ένας κανόνας που δεν λαμβάνει υπόψιν τα χαρακτηριστικά των αντικειμένων (δεν κάνει χρήση κάποιας μετρικής), στην συνέχια του κεφαλαίου αυτού θα δοθεί σημασία στον *Odds* κανόνα. Ο *Odds* κανόνας, λοιπόν, στην καλύτερη περίπτωση πέτυχε μείωση του αριθμού των stale deliveries εώς και 42.8% σε σχέση με τον $ATTL$ με ταυτόχρονη αύξηση του αριθμού των slow hits κατα 53.47%[3].

Αν και η αύξηση του αριθμού των slow hits είναι αξιοσημείωτη και το κόστος αυτής της αύξησης μεταφράζεται σε αύξηση της συμφόρησης δικτύου μεταξύ proxy και origin server, καθώς επίσης σε αύξηση της κατανάλωσης bandwidth σε αυτό το κομμάτι του δικτύου, δεν είναι αρκετή ώστε να προκαλεί ανησυχία. Άλλωστε το latency που προκύπτει απο ένα slow hit είναι αρκετά μικρότερο συγκρινόμενο με αυτό που προκύπτει σε περίπτωση ενός miss. Επιπλέον, το κόστος αυτό είναι αναμενόμενο και εν πολλοίς αναπόφευκτο, μιας και προέρχεται απο το εγγενές tradeoff μεταξύ της εξασφαλίσης υψηλότερου βαθμού συνέπειας μέσω της αποφυγής stale deliveries και της αποσυμφόρησης του δικτύου μέσω της μη αποστολής αιτήσεων επικύρωσης που θα συμβάλλουν σε slow hits.

Δεν παρατηρήθηκε, όμως, η ίδια αξιόλογη συμπεριφορά όταν ο *Vproxy* λειτούργησε έχοντας ως υπόστρωμμα μια πολιτική απόδοσης σταθερών και ρητών TTLs. Αν και βελτίωσε τον αριθμό των stale deliveries, η βελτίωση δεν ήταν αρκετή ούτε σε ποσοστό, μα ούτε και σε απόλυτο αριθμό· ο αριθμός των stale delliveries κινήθηκε σε υψηλές τιμές. Αυτό το αποτέλεσμα οδηγεί στο εύλογο συμπέρασμα της τεράστιας εξάρτησης του *Vproxy* απο το TTL. Ο ορθολογικός καθορισμός του TTL, όπως αυτός γίνεται στην περίπτωση του $ATTL$, είναι αναγκαίος για την αποδοτική λειτουργία του *Vproxy*. Θα μπορούσε, λοιπόν, να ειπωθεί σε αυτό το σημείο οτι μια μελλοντική κατεύθυνση θα αποτελούσε η διεξαγωγή έρευνας για την απεξάρτηση του *Vproxy* απο την τιμή του TTL και την αποδοτική του λειτουργία άσχετα απο αυτό.

Βέβαια το μεγαλύτερο πρόβλημα που αντιμετωπίζουν και οι δύο μηχανισμοί, και ο *Vproxy* αλλά και ο $ATTL$, είναι αυτό που ορίστηκε στο τέταρτο κεφάλαιο ως stale deliveries burst. Δηλαδή το φαινόμενο εκείνο κατα το οποίο ο μηχανισμός συνέπειας καθυστερεί να ανιχνεύσει μια τροποποίηση του αντικειμένου στον origin server και συνεχίζει να ικανοποιεί μεγάλο αριθμό εισερχόμενων αιτήσεων απο την cache, συμβάλλοντας με τον τρόπο αυτό σε ένα αντίστοιχα μεγάλο αριθμό απο stale deliveries. Ενώ θα περίμενε κάποιος το πρόβλημα αυτό να είναι συνυφασμένο με αντικείμενα τα οποία παρουσιάζουν μεγάλο βαθμό μεταβλητότητας η πραγματικότητα είναι αρκετά διαφορετική. Χειρότερη περίπτωση αποτελεί εκείνη η περίπτωση αντικειμένου που παρουσιάζει πολύ μικρό βαθμό μεταβλητότητας (δηλαδή παρατηρούνται τροποποιήσεις σε πολύ αραιά χρονικά διαστήματα), και ταυτόχρονα μεγάλο βαθμό δημοφιλίας. Παρότι ο *Vproxy* καταφέρνει να βελτιώσει το πρόβλημα αυτό σε σχέση με τον $ATTL$ προσαρμόζοντας κατάλληλα τον κανόνα παύσης[4], δεν καταφέρνει να μετριάσει την επίπτωση του φαινομένου[5]. Επομένως, μια επιπλέον μελλοντική κατεύθυνση θα αποτελούσε η διεξαγωγή έρευνας γύρω απο την αντιμετώπιση του προβλήματος αυτού, καθώς θα συνέβαλε σε ακόμα μεγαλύτερη μείωση του αριθμού των stale deliveries με επακόλουθη την ισχυροποιήση της συνέπειας.

Όλη η προηγούμενη ανάλυση καθώς και τα αποτελέσματα που προέκυψαν κατα την προσο-

[3]Περίπτωση χρήσης με u_3 μετρική, και $ATTL$ με $k = 0.2$ και $threshold = 43200$.

[4]Υπενθυμίζεται οτι εάν ο *Vproxy* θεωρήσει οτι ο αριθμός των αιτήσεων που έχει δεχτεί ένα αντικείμενο είναι αρκετά μεγάλος, αλλάζει τον κανόνα παύσης απο *Odds* σε $1/e$.

[5]Ακόμα και εάν ο κανόνας παύσης αλλάξει απο *Odds* σε $1/e$, εάν το TTL είναι αρκετά μεγάλο τότε ακόμα και ο $1/e$ θα αργήσει να αντιληφθεί την τροποποίηση.

μοίωση του *Vproxy* ενίσχυσε την πεποίθηση περι του καίρου ρόλου της μετρικής στην απόδοση του μηχανισμού. Πρώτον, η αμελητέα διαφορά στη συμπεριφορά του Odds κανόνα για τις δύο μετρικές κατέδειξε οτι ο ορισμός μια μετρικής δεν είναι απλή υπόθεση. Παρότι θα περίμενε κανείς οτι η u_4 μετρική θα συνέβαλε στην καλύτερη απόδοση του μηχανισμού, εφόσον χρησιμοποιεί επιπλέον πληροφορία για ένα αντικείμενο, αποδείχθηκε το αντίθετο. Σε όλες σχεδόν τις προσομοιώσεις η απόδοση του *Vproxy* με *Odds* κανόνα και u_4 μετρική παρέμεινε ίδια ή χειροτέρεψε ελάχιστα σε σύγκριση με την u_3 μετρική. Δεύτερον, είναι έντονη η διαίσθηση πως ένας καλύτερος ορισμός της μετρικής θα συμβάλλει στην αντίμετωπιση του stale deliveries burst προβλήματος. Είναι αναγκαίο, λοιπόν, να επεκταθεί η έρευνα σχετικά με τον ορισμό μιας μετρικής. Στα πλαίσια αυτής της εργασίας, έγινε η παραδοχή δίχως απώλεια της γενικότητας, οτι οι μετρικές ακολουθούν την ομοιόμορφη κατανόμή. Αυτό όμως μένει να διερευνηθεί, καθώς διάφορες μελέτες σχετικά με τη δημοφιλιά ενός αντικειμένου, το ποσοστό ευστοχίας μιας ιστοσελίδας, τη μεταβλητότα ενός αντικειμένου έχουν καταδείξει ενδιαφέροντα ευρήματα σχετικά με την κατανομή τους[6] [33, 77–79]. Είναι σημαντικό, λοιπόν, να υπάρξει περαιτέρω έρευνα γύρω απο το ζήτημα ορισμού μιας μετρικής. Να μελετηθούν ποια χαρακτηριστικά είναι τα πλέον απαραίτητα για την συνθεσή της, τον τρόπο συνθεσής της, τα βάρη που θα πάρει καθένα απο αυτά, κ.λπ.

Ένα άλλο ζήτημα που μελετήθηκε και παρουσιάστηκε στο προηγούμενο κεφάλαιο αφορά το λ. Παρατηρήθηκε, λοιπόν, η επίδραση του λ στην απόδοση του *Vproxy*, καθώς όταν εισήχθη ο $\frac{1}{\lambda}$ παράγοντας τα αποτελέσματα άλλαξαν σημαντικά. Συγκεκριμένα, μετά την εισαγωγή του $\frac{1}{\lambda}$ παράγοντα το ποσοστό μείωσης των stale deliveries σε σχέση με τον $ATTL$ μειώθηκε εώς και κατα το ήμισυ, ενώ η αύξηση των slow hits κυμάνθηκε πλέον σε πολύ χαμηλά επίπεδα[7]. Ίσως αυτό να οφείλεται στην παραδοχή της ομοιογένειας της Poisson διεργασίας που μοντελοποιεί τις τυχαίες αφίξεις των αιτήσεων προς ένα αντικείμενο της cache. Συνεπώς, θα ήταν σημαντικό να επεκταθεί η λειτουργία του *Vproxy* κάνοντας χρήση ενός μοντελού μη ομοιογενούς Poisson διεργασίας ή ακόμα και άλλων διεργασιών που μοντελοποιούν τις τυχαίες αφίξεις συμβάντων, ώστε να μελετηθεί η αποδοσή του.

Θα ήταν ακόμα σημαντικό να μελετηθεί περισσότερο η διαχείριση των συμβάντων που σχετίζονται με τον έλεγχο συνέπειας, όπως των slow hits, consistency misses και stale deliveries. Ενδεχόμενη ενσωματωσή τους στον ορισμό της μετρικής ή του κανόνα παύσης, ακόμα περισσότερο τυχόν bayesian ανάλυση αυτών μπορεί να οδηγήσει σε αποδοτικότερη λειτουργία του μηχανισμού. Σημαντική συνεισφορά θα αποτελούσε η ανάπτυξη ενός μηχανισμού που θα προσπαθεί να εκτιμήσει τον αριθμό των stale deliveries που έχουν συμβεί, ώστε να καθορίσει ένα σύστημα αποφάσεων που θα κινείται σε πιο συγκεκριμένο περιβάλλον. Η έλλειψη γνώσης σχετικά με τον αριθμό των stale deliveries που έχουν συμβεί ή έστω η μη ύπαρξη μιας καλής εκτίμησης του αριθμού αυτού είναι αιτία πολλών εσφαλμένων αποφάσεων απο ένα μηχανισμό συνέπειας.

Τέλος, είναι αναγκαίο να επισημανθεί πως η επιλογή της υλοποιήσης του *Vproxy* ως ένα

[6] Υπευθυμίζεται οτι όλα τα προηγούμενα αποτελούν συστατικά στοιχεία των δύο μετρικών που ορίστηκαν στα πλαίσια της παρούσας μελέτης, δηλαδή της u_3 και u_4. Συνεπώς η κατανομή αυτών μπορεί να οδηγήσει στην θεωρητική τεκμηρίωση της κατανομής των μετρικών. Βέβαια ο ακριβέστερος ορισμός της κατανομής τους, μάλλον, θα είναι εύρημα μια trace driven ανάλυσης, παρά μιας θεωρητικής τεκμηρίωσης.

[7] Το ποσοστό αύξησης των slow hits για τον *Vproxy* μηχανισμό με χρήση *Odds* κανόνα σε σύγκριση με τον $ATTL$, ήταν κατα μέσο όρο της τάξης του 10%.

σύγχρονο μηχανισμό βασίζεται στην παραδοχή οτι το latency που παρατηρεί ένας χρήστης του Web προέρχεται κυρίως απο την μεταφορά των δεδομένων μέσα στο δίκτυο, παρά απο την επεξεργασία της αίτησης του σε έναν proxy. Πιο συγκεκριμένα η ταχύτητα των επεξεργαστών είναι αρκετά μεγάλη, με αποτέλεσμα η συμβολή της επεξεργασίας των δεδομένων μιας εισερχόμενης αίτησης στο latency μιας απάντησης που λαμβάνει ένας Web χρήστης, είναι αμελητέα μπροστά στην καθυστέρηση που προκύπτει απο την μεταφορά των αντιστοίχων δεδομένων μέσα στο διαδίκτυο. Και ο λόγος που γίνεται η επισήμανση αυτή οφείλεται στο γεγονός της απαιτούμενης επεξεργασίας κάθε εισερχόμενης αίτησης στην cache απο τον *Vproxy*. Πράγματι παρότι ο *ATTL* κάνει μονάχα έναν απλό έλεγχο για κάθε αίτηση, ο *Vproxy* απαιτεί μια πιο πολύπλοκη επεξεργασία αυτής. Επιπλέον, ο *Vproxy* απαιτεί περισσότερη μνήμη απο τον *ATTL*, απαραίτητη για την κατάλληλη επεξεργασία της αίτησης. Όμως και αυτό δεν συνιστά μεγάλο πρόβλημα, καθώς το μέγεθος της μνήμης και η ταχυτητά της αυξάνονται διαρκώς τα τελευταία χρόνια. Εν κατακλείδι, εάν οι αιτήσεις κατανεμηθούν σε έναν μεγάλο αριθμό απο proxies οι οποίοι θα αποτελούν αρκετά ισχυρά υπολογιστικά συστήματα αφιερωμένα μονάχα στην παροχή caching λειτουργικότητας, τα παραπάνω ζητήματα δεν αποτελούν πρόβλημα.

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	Ελληνικός όρος
access log	αρχείο πρόσβασης
almost surely	σχεδόν σίγουρα
applicant	αιτών
application layer	επίπεδο εφαρμογής
Application Programming Interface	Διασύνδεση, Διεπαφή Προγραμματισμού Εφαρμογών
asynchronous	ασύγχρονος
backward induction	προς τα πίσω επαγωγή
bandwidth	εύρος ζώνης
best-choice problem	πρόβλημα βέλτιστης επιλογής
biased	μεροληπτικός
binomial distribution	διωνυμική κατανομή
block cache	κρυφή μνήμη μπλοκ
bottleneck	σημείο συμφόρησης (δικτύου)
browser	πρόγραμμα περιήγησης, φυλλομετρητής
browser cache	κρυφή μνήμη προγράμματος περιήγησης
buffer	προσωρινή μνήμη
buffer cache	κρυφή μνήμη προσωρινής αποθήκευσης
bug	σφάλμα
cache	κρυφή μνήμη
cache consistency problem	πρόβλημα συνέπειας κρυφής μνήμης
cache hit	ευστοχία κρυφής μνήμης
cache management	διαχείριση κρυφής μνήμης
cache miss	αστοχία κρυφής μνήμης
cache server	εξυπηρετητής κρυφής μνήμης
caching	χρήση της κρυφής μνήμης
candidate	υποψήφιος
Central Processor Unit	Κεντρική Μονάδα Επεξεργασίας
classical secretary prblem	κλασσικό πρόβλημα της γραμματέως
client	πελάτης
client list problem	πρόβλημα λίστας πελατών
collision chain	αλυσίδα σύγκρουσης
common physical clock	κοινό φυσικό ρολόι
communication	επικοινωνία
conditional request	υπο συνθήκη αίτηση
consistency	συνέπεια
consistent	συνεπές

Συνεχίζεται στην επόμενη σελίδα

Πίνακας Ορολογίας – συνέχεια από την προηγούμενη σελίδα

Ξενόγλωσσος όρος	Ελληνικός όρος
content provider	πάροχος περιεχομένου
copyright	πνευματική ιδιοκτησία, πνευματικά δικαιώματα
CPU cache	κρυφή μνήμη επεξεργαστή
default	προεπιλεγμένη (τιμή)
delayed updates dilemma	δίλημμα καθυστερημένων ενημερώσεων
design pattern	σχεδιαστικό πρότυπο
directive	οδηγία
distributed system	κατανεμημένο σύστημα
disk cache	κρυφή μνήμη σκληρού δίσκου
Dynamic Random Access Memory	Δυναμική Μνήμη Τυχαίας Προσπέλασης
enterprise	επιχείρηση
entity body	σώμα (οντότητας)
entity header	κεφαλίδα οντότητας
event	γεγονός, συμβάν
eviction	έξωση
failure	αποτυχία
fair	δίκαιος
fault tolerance	ανοχή σε σφάλματα
file system	σύστημα αρχείων
forward proxy	πληρεξούσιος προώθησης
gateway	πύλη εισόδου
general header	γενική κεφαλίδα
hardware	υλικό
hash function	συνάρτηση κατακερματισμού
hash table	πίνακας κατακερματισμού
header	κεφαλίδα
heterogeneity	ανομοιογένεια, ετερογένεια
heuristic	ευρετικός
hit	ευστοχία
hit ratio	ποσοστό ευστοχίας
homogeneous	ομοιογενής
house selling problem	πρόβλημα πώλησης σπιτιού
hypertext	υπερκείμενο
Hypertext Transfer Protocol	Πρωτόκολλο Μεταφοράς Υπερκειμένου
identically independently distributed	ομοιόμορφα ανεξάρτητα κατανεμημένα
inconsistent	ασυνεπής, μη συνεπής
increment	βήμα
indicator function	δείκτρια συνάρτηση
infinite	άπειρος, μη πεπερασμένος
information system	πληροφοριακό σύστημα
inhomogeneous	ανομοιογενής, μη ομοιογενής

Συνεχίζεται στην επόμενη σελίδα

Πίνακας Ορολογίας – συνέχεια από την προηγούμενη σελίδα

Ξενόγλωσσος όρος	Ελληνικός όρος
intensity	ένταση
interarrival time	χρόνος μεταξύ δύο αφίξεων
Internet	Διαδίκτυο
Internet Protocol Suite	Σουίτα Πρωτοκόλλων Διαδικτύου
Internet Service Provider	Πάροχος Υπηρεσιών Διαδικτύου
invalid	άκυρος, μη έγκυρος
invalidation	ακύρωση
joint distribution	κοινή κατανομή
keyword	λέξη-κλειδί
latency	καθυστέρηση απόκρισης
lease	μίσθωση
locality	αρχή της τοπικότητας
loop	βρόγχος
Markov chain	Markov αλυσίδα
Markov structure	Markov δομή
marriage problem	πρόβλημα επιλογής συζύγου
mean	μέσος
median	διάμεσος
memory hierarchy	ιεραρχία μνήμης
meta-information	μετα-πληροφορία
miss	αστοχία
modularity	τμηματοποίηση
naming	ονοματοδοσία
object mutability	μεταβλητότητα αντικειμένου
object oriented	αντικειμενοστραφής
object popularity	δημοτικότητα αντικειμένου
Operations Research	Επιχειρησιακή Έρευνα
Optimal Stopping	Βέλτιστη Παύση
origin server	εξυπηρετητής προέλευσης
path	μονοπάτι
payoff	ανταμοιβή
probability mass function	συνάρτηση μάζας πιθανότητας
process	διεργασία
proprietary	ιδιόκτητο
proxy	πληρεξούσιος
proxy cache	πληρεξούσιος κρυφής μνήμης
random variable	τυχαία μεταβλητή
randomized	τυχαιοκρατικός
randomized stopping rule	τυχαιοκρατικός κανόνας παύσης
rank	κατάταξη
rate parameter	παράμετρος ρυθμού
reference	αναφορά

Συνεχίζεται στην επόμενη σελίδα

Πίνακας Ορολογίας – συνέχεια από την προηγούμενη σελίδα

Ξενόγλωσσος όρος	Ελληνικός όρος
replication	αναπαραγωγή
request	αίτηση
request header	κεφαλίδα αίτησης
request time ratio	χρονικό ποσοστό αίτησης
response	απάντηση
response header	κεφαλίδα απάντησης
reverse proxy cache	αντίστροφος πληρεξούσιος κρυφής μνήμης
reward function	συνάρτηση ανταμοιβής
reward sequence	ακολουθία ανταμοιβής
scalability	επεκτασιμότητα
security	ασφάλεια
secretary problem	πρόβλημα της γραμματέως
server	εξυπηρετητής
shared memory	διαμοιραζόμενη μνήμη
simulator	προσομοιωτής
site hit ratio	ποσοστό ευστοχίας ιστοτόπου
software	λογισμικό
spatial locality	χωρική τοπικότητα
stage	φάση
stale	έωλος
state machine	μηχανή καταστάσεων
Static Random Access Memory	Στατική Μνήμη Τυχαίας Προσπέλασης
status code	κωδικός κατάστασης
status line	γραμμή κατάστασης
stochastic counting process	στοχαστική διεργασία καταμέτρησης
stopping rule	κανόνας παύσης
strong consistency	ισχυρή συνέπεια
strong law of large numbers	ισχυρός κανόνας των μεγάλων αριθμών
success	επιτυχία
synchronization	συγχρονισμός
synchronous	σύγχρονος
temporal locality	χρονική τοπικότητα
three-way handshake	χειραψία τριών βημάτων
threshlod	κατώφλι
threshlod rule	κανόνας κατωφλίου
timestamp	χρονοσφραγίδα
Time to Live	χρόνος ζωής
tradeoff	δίλημμα, ισορροπία
Transmission Control Protocol	Πρωτόκολλο Ελέγχου Μεταφοράς
transparency	διαφάνεια
transparent proxy cache	διάφανος πληρεξούσιος κρυφής μνήμης
transport layer	επίπεδο μεταφοράς

Συνεχίζεται στην επόμενη σελίδα

Πίνακας Ορολογίας – συνέχεια από την προηγούμενη σελίδα

Ξενόγλωσσος όρος	Ελληνικός όρος
truncation method	μέθοδος περικοπής
tunneling proxy	πληρεξούσιος διοχέτευσης
user interface	διεπαφή χρήστη
validation	επικύρωση
validation request	αίτηση επικύρωσης
validity	εγκυρότητα
version	έκδοση
weak consistency	ασθενή συνέπεια
Web	(Παγκόσμιος) Ιστός
Web cache	κρυφή μνήμη Παγκόσμιου Ιστού
Web caching	χρήση κρυφής μνήμης Παγκόσμιου Ιστού
Web caching consistency problem	πρόβλημα συνέπειας κρυφής μνήμης Παγκόσμιου Ιστού
Web caching proxy	πληρεξούσιος κρυφής μνήμης Παγκόσμιου Ιστού
Web client	πελάτης Παγκόσμιου Ιστού
Web proxy	πληρεξούσιος Παγκόσμιου Ιστού
Web server	εξυπηρετητής Παγκόσμιου Ιστού
website	ιστοσελίδα
workload	φόρτος εργασίας
World Wide Web	Παγκόσμιος Ιστός
write back	ετερόχρονη εγγραφή
write-through	άμεση, ταυτόχρονη εγγραφή
write-through cache	κρυφή μνήμη άμεσης, ταυτόχρονης εγγραφής

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ

Σύντμηση	Πλήρης Ανάπτυξη
API	Application Programming Interface
as	almost surely
ASCII	American Standard Code for Information Interchange
CPU	Central Processor Unit
CR	carriage return
CSP	classical secretary problem
DEC	Digital Equipment Corporation
DRAM	Dynamic Random Access Memory
DS	distributed system
FIFO	First In First Out
GB	gigabyte(s)
HTTP	Hypertext Transfer Protocol
iid	identically independently distributed
IS	information system
ISP	Internet Service Provider
k-sla	k-stages look ahead
LF	line feed
LRU	Least Recently Used
pmf	probability mass function
rv	random variable
sla	stages look ahead
SRAM	Static Random Access Memory
TCP	Transmission Control Protocol
TTL	Time to Live
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WWW	World Wide Web
ΔΠΕ	Διασύνδεση, Διεπαφή Προγραμματισμού Εφαρμογών
ΕΚΠΑ	Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
κ.α.	και άλλα
κ.λπ.	και λοιπά, και τα λοιπά
ΚΜΕ	Κεντρική Μονάδα Επεξεργασίας
κ.ο.κ.	και και ούτω καθεξής
ΚΣ	κατανεμημένο σύστημα
ΠΙ	Παγκόσμιος Ιστός
ΠΣ	πληροφοριακό σύστημα
τμ	τυχαία μεταβλητή

ΠΑΡΑΡΤΗΜΑ Ι

Στα πλαίσια αυτού του παραρτήματος παρουσιάζονται οι πίνακες με τα αποτελέσματα των προσομοιώσεων. Οι πίνακες αυτοί περιέχουν τα δεδομένα βάσει των οποίων προέκυψαν τα γραφήματα του πέμπτου κεφαλαίου. Στο σημείο αυτό, υπευθυμίζεται οτι οι προσομοιώσεις διεξήχθησαν σε έναν προσομοιωτή που διέθετε LRU cache μεγέθους 4 GB και ότι τα δεδομένα προήρθαν απο Web traces της DEC τα οποία ελήφθησαν σε μια χρονική περίοδο τρεισήμισι εβδομάδων το έτος 1996 και περιλαμβάνουν 24.477.674 αναφορές.

Ακολουθούν οι πίνακες της πρώτης σειράς προσομοιώσεων:

Stale deliveries για διάφορες τιμές threshold				
threshold	ATTL	1/e	odds-u3	odds-u4
86400	81815	31872	49374	51861
172800	112258	45808	83436	85391
259200	159290	63389	101807	102281
345600	141090	61275	115732	117939
432000	228532	76778	130618	132657
518400	186318	92413	137431	139371
604800	224392	82254	175050	176999

Πίνακας Ι.1: Stale deliveries για διάφορες τιμές threshold.

Slow hits για διάφορες τιμές threshold				
threshold	ATTL	1/e	odds-u3	odds-u4
86400	1042654	2115071	1721424	1682387
172800	643617	1560109	1024190	985668
259200	525561	1193146	823748	797554
345600	474122	1048318	736091	711660
432000	448252	976277	687963	666807
518400	435978	909948	668250	649092
604800	427926	878744	656930	638440

Πίνακας Ι.2: Slow hits για διάφορες τιμές threshold.

Consistency misses για διάφορες τιμές threshold				
threshold	ATTL	1/e	odds-u3	odds-u4
86400	199503	210287	208218	208386
172800	196750	207632	204650	204610
259200	195938	206295	203894	203853
345600	194858	205869	202470	202388
Συνεχίζεται στην επόμενη σελίδα				

Πίνακας I.3 – συνέχεια από την προηγούμενη σελίδα

Consistency misses για διάφορες τιμές threshold				
432000	194588	205743	202044	201894
518400	194334	205172	201698	201646
604800	194191	205124	201636	201502

Πίνακας I.3: Consistency misses για διάφορες τιμές threshold.

Stale deliveries για διάφορες τιμές k				
threshold	ATTL	1/e	odds-u3	odds-u4
0.1	138820	52463	118430	118832
0.2	321025	127145	207078	208730
0.3	363964	191909	274254	272808
0.4	454575	257984	353969	355912
0.5	493795	270129	388739	394361
0.6	448593	356187	474509	478080
0.7	526267	385166	514424	515934
0.8	581927	356568	516865	520677
0.9	607025	414261	563616	565051
1	620443	441454	581220	589632

Πίνακας I.4: Stale deliveries για διάφορες τιμές k.

Slow hits για διάφορες τιμές k				
threshold	ATTL	1/e	odds-u3	odds-u4
0.1	648178	1156949	982370	957484
0.2	407607	780294	633906	616045
0.3	302177	607645	475255	461718
0.4	242147	503269	382954	371617
0.5	202426	432507	321435	311949
0.6	173926	379133	277974	269758
0.7	152511	338238	244415	237229
0.8	136112	306442	218634	212540
0.9	123555	279995	198032	192269
1	112509	258920	180938	175266

Πίνακας I.5: Slow hits για διάφορες τιμές k.

Consistency misses για διάφορες τιμές k				
threshold	ATTL	1/e	odds-u3	odds-u4
0.1	202012	208799	206696	206444
0.2	193803	204202	201184	201061
0.3	187121	200927	196305	196155
0.4	181209	197904	191922	191288
Συνεχίζεται στην επόμενη σελίδα				

Πίνακας I.6 – συνέχεια από την προηγούμενη σελίδα

Consistency misses για διάφορες τιμές k				
0.5	176244	195091	187298	186795
0.6	171700	192182	183956	183483
0.7	167663	189519	180335	179631
0.8	164203	187360	177430	176912
0.9	161054	184933	174639	174121
1	158247	182932	172335	171515

Πίνακας I.6: Consistency misses για διάφορες τιμές k.

Στη συνέχεια ακολουθούνε οι πίνακες με τα αποτελέσματα των προσομοιώσεων, κάνοντας χρήση του παράγοντα $\frac{1}{\lambda}$:

Stale deliveries για διάφορες τιμές threshold και παράγοντα $\frac{1}{\lambda}$				
threshold	ATTL	1/e	odds-u3	odds-u4
86400	81815	31872	64667	70019
172800	112258	45808	97710	104255
259200	159290	63389	114051	120833
345600	141090	61275	130224	136865
432000	228532	76778	145075	151665
518400	186318	92413	150644	156138
604800	224392	82254	188784	194390

Πίνακας I.7: Stale deliveries για διάφορες τιμές threshold και παράγοντα $\frac{1}{\lambda}$.

Slow hits για διάφορες τιμές threshold και παράγοντα $\frac{1}{\lambda}$				
threshold	ATTL	1/e	odds-u3	odds-u4
86400	1042654	2115071	1295331	1190915
172800	643617	1560109	768315	720742
259200	525561	1193146	617986	584396
345600	474122	1048318	556426	527733
432000	448252	976277	524244	498576
518400	435978	909948	511482	486025
604800	427926	878744	501887	477190

Πίνακας I.8: Slow hits για διάφορες τιμές threshold και παράγοντα $\frac{1}{\lambda}$.

Consistency misses για διάφορες τιμές threshold και παράγοντα $\frac{1}{\lambda}$				
threshold	ATTL	1/e	odds-u3	odds-u4
86400	199503	210287	203453	202766
172800	196750	207632	200748	199890
259200	195938	206295	199308	198688
345600	194858	205869	198354	197640
Συνεχίζεται στην επόμενη σελίδα				

Πίνακας I.9 – συνέχεια από την προηγούμενη σελίδα

Consistency misses για διάφορες τιμές threshold και παράγοντα $\frac{1}{\lambda}$				
432000	194588	205743	197856	197256
518400	194334	205172	197711	197096
604800	194191	205124	197505	196896

Πίνακας I.9: Consistency misses για διάφορες τιμές threshold και παράγοντα $\frac{1}{\lambda}$.

Stale deliveries για διάφορες τιμές k και παράγοντα $\frac{1}{\lambda}$				
threshold	ATTL	1/e	odds-u3	odds-u4
0.1	138820	52463	126357	127067
0.2	321025	127145	220387	226090
0.3	363964	191909	284799	288914
0.4	454575	257984	371144	377261
0.5	493795	270129	411127	414963
0.6	448593	356187	493353	499719
0.7	526267	385166	532186	535708
0.8	581927	356568	535207	534658
0.9	607025	414261	580448	585289
1	620443	441454	603532	604179

Πίνακας I.10: Stale deliveries για διάφορες τιμές k και παράγοντα $\frac{1}{\lambda}$.

Slow hits για διάφορες τιμές k και παράγοντα $\frac{1}{\lambda}$				
threshold	ATTL	1/e	odds-u3	odds-u4
0.1	648178	1156949	753327	715453
0.2	407607	780294	480763	456549
0.3	302177	607645	360408	342015
0.4	242147	503269	289755	276012
0.5	202426	432507	243018	231816
0.6	173926	379133	209584	199314
0.7	152511	338238	184081	175823
0.8	136112	306442	164820	157168
0.9	123555	279995	148578	142115
1	112509	258920	135933	130690

Πίνακας I.11: Slow hits για διάφορες τιμές k και παράγοντα $\frac{1}{\lambda}$.

Consistency misses για διάφορες τιμές k και παράγοντα $\frac{1}{\lambda}$				
threshold	ATTL	1/e	odds-u3	odds-u4
0.1	202012	208799	204225	203667
0.2	193803	204202	197116	196517
0.3	187121	200927	191466	190677
0.4	181209	197904	186043	185500
Συνεχίζεται στην επόμενη σελίδα				

Πίνακας I.12 – συνέχεια από την προηγούμενη σελίδα

Consistency misses για διάφορες τιμές k και παράγοντα $\frac{1}{\lambda}$				
0.5	176244	195091	181663	181027
0.6	171700	192182	177390	176555
0.7	167663	189519	173753	173178
0.8	164203	187360	170957	170097
0.9	161054	184933	167953	167178
1	158247	182932	165225	164821

Πίνακας I.12: Consistency misses για διάφορες τιμές k και παράγοντα $\frac{1}{\lambda}$.

Τέλος, παρουσιάζονται οι πίνακες οι οποίοι περιέχουν τα δεδομένα της τρίτης σειράς προσομοιώσεων, κατα την οποία χρησιμοποιήθηκε ένας απλοϊκός αλγόριθμος απόδοσης σταθερών TTLs:

Stale deliveries για διάφορες τιμές TTL				
fixed TTL	ATTL	1/e	odds-u3	odds-u4
86400	378784	255087	326931	330955
172800	501820	327547	468114	473281
259200	605886	416984	553872	556925
345600	626424	457733	612759	614609
432000	752135	501442	668070	670870
518400	746941	560965	705204	707329
604800	807387	588089	731550	732287

Πίνακας I.13: Stale deliveries για διάφορες τιμές TTL.

Slow hits για διάφορες τιμές TTL				
fixed TTL	ATTL	1/e	odds-u3	odds-u4
86400	788487	1810925	1350625	1316668
172800	322932	1144503	532869	498938
259200	173870	679304	274798	260386
345600	107750	478002	161060	150158
432000	70958	372925	93032	86913
518400	52319	271512	63342	60585
604800	39755	219224	44710	43470

Πίνακας I.14: Slow hits για διάφορες τιμές TTL.

Consistency misses για διάφορες τιμές TTL				
fixed TTL	ATTL	1/e	odds-u3	odds-u4
86400	68451	99626	86617	86411
172800	41595	82005	54869	52301
259200	30639	60074	37804	37021
Συνεχίζεται στην επόμενη σελίδα				

Πίνακας I.15 – συνέχεια από την προηγούμενη σελίδα

Consistency misses για διάφορες τιμές TTL				
345600	25265	49459	29047	28341
432000	21405	44628	23194	22721
518400	19596	36834	20693	20474
604800	18382	33405	18863	18758

Πίνακας I.15: Consistency misses για διάφορες τιμές TTL.

ΑΝΑΦΟΡΕΣ

[1] "Internet World Stats", 30 Jun. 2012;
http://www.internetworldstats.com/stats.htm.
[Προσπελάστηκε 02/2/13]

[2] "The Domain Name Industry Brief", Dec. 2012;
http://www.verisigninc.com/assets/domain-name-brief-dec2012.pdf.
[Προσπελάστηκε 02/2/13]

[3] R. Fielding et al., *Hypertext Transfer Protocol – HTTP/1.1*, IETF RFC 2616, June 1999; http://tools.ietf.org/rfc/rfc2616.txt.

[4] Thomas S. Ferguson, "Optimal Stopping and Applications", UCLA,
http://www.math.ucla.edu/ tom/Stopping/Contents.html.
[Προσπελάστηκε 02/2/13]

[5] David A. Patterson and John L. Hennessy, *Computer Organization and Design, 3rd Edition*, Elesvier, 2005, pp. 486-510.

[6] Andrew S. Tanenbaum, *Modern Operating Systems, Second Edition*, Prentice Hall, 2001, pp 508-511.

[7] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach*, Elsevier, 2011, Appendix B *Review of Memory Hierarchy.*

[8] James A. O'Brien and George M. Marakas, *Introduction to Information Systems, 15th Edition*, McGraw-Hill, 2010, p. 4.

[9] Ajay D. Kshemkalyani and Mukesh Singhal, *Distributed Computing: Principles, Algorithms, and Systems*, Cambridge University Press, 2008, pp. 1-38.

[10] André B. Bondi, "Characteristics of scalability and their impact on performance," *Proc. 2nd Int'l workshop Software and performance (WOSP '00)*, ACM, 2000, pp. 195-203.

[11] Mark D. Hill, "What is scalability?," *SIGARCH Comput. Archit. News 18*, Dec. 1990, pp. 18-21.

[12] V. Jacobson, "How to kill the Internet," A presentation at SIGCOMM '95, Middleware Workshop.

[13] Silicon Press, "What is Web Caching?,"
http://www.silicon-press.com/briefs/brief.webcaching/index.html.
[Προσπελάστηκε 04/2/13]

[14] G. Huston. "Web Caching," *Cisco, The Internet Protocol Journal*, vol. 2, no. 3, Sep. 1999. [Προσπελάστηκε 04/2/13]

[15] Michael Rabinovich and Oliver Spatschek, *Web Caching and Replication*, Addison-Wesley Longman Publishing Co., 2002.

[16] Vinton G. Cerf and Robert E. Icahn, "A protocol for packet network intercommunication," *SIGCOMM Comput. Commun. Rev. Newsletter*, vol. 35, no. 2, Apr. 2005, pp. 71-82.

[17] J. Postel, *User Datagram Protocol*, IETF RFC 768, August 1980; http://tools.ietf.org/rfc/rfc768.txt.

[18] *Transmission Control Protocol*, IETF RFC 793, September 1981; http://tools.ietf.org/rfc/rfc793.txt.

[19] T. M. Kroeger, D. D. E. Long and J. C. Mogul, "Exploring the bounds of Web latency reduction from caching and prefetching," *Proc. USENIX Symp. Internet Technologies and Systems (USITS '97)*, USENIX Association, 1997, pp. 13-22.

[20] A. Feldmann et al., "Performance of Web proxy caching in heterogeneous bandwidth environments," *Proc. 18th Annu. Joint Conf. the IEEE Computer and Communications Societies (INFOCOM '99)*, IEEE CS, 1999, pp. 107-116.

[21] R. Cohen and S. Ramanathan, *Using proxies to enhance TCP performance over hybrid fiber coaxial networks*, tech. report HPL-97-81, Hewlett-Packard Labs, 1997.

[22] M. Arlitt, R. Friedrich and T. Jin, *Workload characterization of a Web proxy in a cable modem environment*, tech. report HPL-1999-48, Hewlett Packard Labs, 1999.

[23] A. Wolman et al., "On the scale and performance of cooperative Web proxy caching," *Proc. 17th ACM Symp. Operating System Principles (SOSP '99)*, ACM, 1999, pp. 16-31.

[24] T. Berners-Lee, R. Fielding and H. Frystyk, *Hypertext Transfer Protocol – HTTP/1.0*, IETF RFC 1945, May 1996; http://www.ietf.org/rfc/rfc1945.txt.

[25] T. Berners-Lee, L. Masinter and M. McCahill, *Uniform Resource Locators (URL)*, IETF RFC 1945, December 1996; http://www.rfc-editor.org/rfc/rfc1738.txt.

[26] Leon Shklar and Richard Rosen, *Web Application Architecture: Principles, protocols and practices*, John Wiley & Sons Ltd, 2003.

[27] G. H. Gonnet and R. Baeza-Yates, *Handbook of Algorithms and Data Structures: In Pascal and C, Second Edition*, Addison-Wesley Longman Publishing Co., 1991.

[28] R. Fielding, M. Nottingham and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Caching," IETF Internet draft, work in progress, Oct. 2012; http://tools.ietf.org/id/draft-ietf-httpbis-p6-cache-21.txt.

[29] S.V. Nagaraj, *Web Caching and Its Applications*, Springer, 2004.

[30] J. Wang, "A survey of web caching schemes for the Internet," *SIGCOMM Comput. Commun. Rev. 29*, Oct. 1999, pp. 36-46.

[31] J. Dilley, "The effect of consistency on cache response time," *The Magazine of Global Internetworking*, vol. 14, no. 3, May 2000, pp. 24-28.

[32] E. Cohen and H. Kaplan, "Web Content Delivery: The Time-to-Live Based Consistency Mechanism," *Web Content Delivery*, Web Information Systems Engineering and Internet Technologies Book Series 2, X. Tang, J. Xu and Samuel T. Chanson, eds., Springer, 2005, pp. 45-71.

[33] B.M. Duska, D. Marwood and M.J. Feeley, "The measured access characteristics of World-Wide-Web client proxy caches," *Proc. USENIX Symp. Internet Technologies and Systems (USITS '97)*, USENIX Association, 1997, pp. 23-35.

[34] V. Cate, "Alex - a global filesystem," *Proc. USENIX File Systems Workshop*, 1992, pp. 1-12.

[35] B. Krishnamurthy and C. E. Wills, "Study of piggyback cache validation for proxy caches in the World Wide Web," *Proc. USENIX Symp. Internet Technologies and Systems, (USITS '97)*, USENIX Association, pp. 1-12.

[36] C. G. Gray and D. R. Cheriton, "Leases: An efficient fault-tolerant mechanism for distributed file cache consistency," *Proc. 12th ACM Symp. Operating System Principles, (SOSP '89)*, ACM, 1989, pp. 202-210.

[37] V. Duvvuri, P. Shenoy and R. Tewari, "Adaptive leases: A strong consistency mechanism for the World Wide Web," *IEEE Trans. Knowledge and Data Engineering*, vol. 15, no. 5, Sep. 2003, pp. 1266-1276.

[38] J. Gwertzman and M. Seltzer, "World-wide web cache consistency," *Proc. the 1996 ann. conf. USENIX (ATEC '96)*, USENIX Association, 1996, p. 12.

[39] P. Cao and C. Liu, "Maintaining Strong Cache Consistency in the World Wide Web," *IEEE Transactions on Computers*, vol. 47 no. 4, Apr. 1998, pp. 445-457.

[40] A. Wald, "Sequential Tests of Statistical Hypotheses," *Ann. Math. Statist.*, vol. 16, no. 2, 1945, pp. 117-186.

[41] Abraham Wald, *Sequential Analysis*, John Wiley & Sons, 1947.

[42] Abraham Wald, *Statistical Decision Functions*, John Wiley & Sons, 1950.

[43] K. J. Arrow, D. Blackwell and M. A. Girshick, "Bayes and minimax solutions of sequential decision problems," *Econometrica*, vol. 17, no. 3/4, Jul. - Oct., 1949, 213-244.

[44] L. J. Snell, "Applications of martingale system theorems," *Trans. Amer. Math. Soc.*, vol. 73, no. 2, Sep. 1952, 293-312.

[45] Y. S. Chow and H. Robbins, "A martingale system theorem and applications," *Proc. 4th Berk. Symp. on Math. Statist. and Prob.*, vol. 1, 1961, pp. 93-104.

[46] Y. S. Chow and H. Robbins, "On optimal stopping rules," *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, vol 2, no. 1, 1963, pp. 33-49.

[47] Y. S. Chow, H. Robbins and D. Siegmund, *Great Expectations: The Theory of Optimal Stopping*, Houghton Mifflin Company, 1971.

[48] Pranab K. Sen and Julio M. Singer, *Large Sample Methods in Statistics: An Introduction with Applications*, Chapman & Hall, 1994.

[49] A. N. Shiryaev, "Statistical Sequential Analysis: Optimal Stopping Rules," *Statistical Sequential Analysis*, Translations of Mathematical Monographs vol. 38, American Mathematical Society, 1973.

[50] Thomas S. Ferguson, "Who Solved the Secretary Problem?," *Statistical Science*, vol. 4, no. 3, 1989, pp. 282-289.

[51] P. R. Freeman, "The Secretary Problem and Its Extensions: A Review," *International Statistical Review / Revue Internationale de Statistique*, vol. 51, no. 2, Aug. 1983, pp. 189-206.

[52] J. Gianini and S. M. Samuels, "The Infinite Secretary Problem," *Ann. Probab*, vol. 4, no. 3, 1976, pp. 418-432.

[53] A. L. Rocha, "The Infinite Secretary Problem with Recall," *Ann. Probab*, vol. 21, no. 4, 1993, pp. 898-916.

[54] F. T. Bruss, 'Sum the odds to one and stop," *Ann. Probab*, vol. 28, no. 3, 2000, pp. 1384-1391.

[55] F. T. Bruss, "A note on bounds for the odds theorem of optimal stopping," *Ann. Probab*, vol. 31, no. 4, 2003, pp. 1859-1961.

[56] F. T. Bruss, "A Unified Approach to a Class of Best Choice Problems with an Unknown Number of Options," *Ann. Probab*, vol. 12, no. 3, 1984, pp. 882-889.

[57] T. S. Ferguson, "The sum-the-odds theorem with application to a stopping game of Sakaguchi," preprint, 2008.

[58] F. T. Bruss and G. Louchard, "The odds algorithm based on sequential updating and its performance," *Adv. in Appl. Probab.*, vol. 41, no. 1, 2009, pp. 131-153.

[59] D. V. Lindley, "Dynamic Programming and Decision Theory," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 10, no. 1, 1961, pp. 39-51.

[60] E. L. Presman, I. M. Sonin, "The best choice problem for a random number of objects", *Teor. Veroyatnost. i Primenen.*, vol. 17, no. 4, 1972, pp. 695–706.

[61] R. Cowan and J. Zabczyk, "An optimal selection problem associated with the Poisson process," *Teor. Veroyatnost. i Primenen.*, vol. 23, no. 3 1978, pp. 606–614

[62] F. Thomas Bruss, "On an optimal selection problem of Cowan and Zabczyk," *J. Appl. Prob.*, vol. 24, no. 4, 1987, pp. 918-928.

[63] M. Sakaguchi, "Optimal stopping problems for randomly arriving offers," *Mathematicae Japonicae*, vol. 21, 1976, pp. 201-217.

[64] T. Bojdecki, "On optimal stopping of a sequence of independent random variables - probability maximizing approach," *Stochastic Processes and their Applications*, vol. 6, no. 2, January 1978, pp. 153–163.

[65] T. S. Ferguson, J. P. Hardwick and M. Tamaki, "Maximizing the duration of owning a relatively best object," *Strategies for Sequential Search and Selection in Real Time*, Contemporary Mathematics vol. 125, American Mathematical Society, 1992, pp. 37-57.

[66] V. Paxson and S. Floyd, "Why we don't know how to simulate the Internet," *Proc. 29th conf. Winter simulation (WSC '97)*, IEEE CS, 1997, pp. 1037-1044.

[67] Ramón Cáceres et al., "Web proxy caching: the devil is in the details," *ACM SIGMETRICS Performance Evaluation Rev. Newsletter*, vol. 26, no. 3, Dec. 1998, pp. 11-15.

[68] P. Barford and M. Crovella, "Generating representative Web workloads for network and server performance evaluation," *Proc. 1998 ACM SIGMETRICS joint Int'l conf. Measurement and modeling of computer systems (SIGMETRICS '98/PERFORMANCE '98)*, ACM, 1998, pp. 151-160.

[69] V. Almeida et al., "Characterizing reference locality in the WWW," *Proc. 4th Int'l conf. Parallel and distributed information systems (DIS '96)*, IEEE CS, 1996, pp. 92-107.

[70] G. Abdulla, "Analysis and Modeling of World Wide Web Traffic," doctoral dissertation, Virginia Polytechnic and State University, 1998.

[71] J. Almeida and P. Cao, "Measuring proxy performance with the Wisconsin Proxy benchmark," *Computer Networks and ISDN Systems*, vol. 30, no. 22-23, Nov. 1998, pp. 2179-2192.

[72] P. Owezarski and N. Larrieu, "A trace based method for realistic simulation," *2004 IEEE Int'l Conf. Communications*, IEEE CS, 2004, pp. 2236-2239.

[73] "Digital's Web Proxy Traces (V1.2 format)", Aug - Sep 1996; http://apotheca.hpl.hp.com/ftp/pub/dec/traces/proxy/webtraces.v1.2.html. [Προσπελάστηκε 07/3/13]

[74] Z. Zhu, Y. Mao and W. Shi, "Web Engineering: Workload Characterization of Uncacheable HTTP Content," *Web Engineering*, Lecture Notes in Computer Science, N. Koch, P. Fraternali and M. Wirsing, eds., Springer, 2004, pp. 391-395.

[75] X. Zhang, "Cachability of Web Objects," master's thesis, Dept. Computer Science, Boston University, 2000.

[76] M. Ghosh, N. Mukhopadhyay and Pranab K. Sen, *Sequential Estimation*, Wiley-Interscience, 1997.

[77] James E. Pitkow, "Summary of WWW characterizations," *World Wide Web*, vol. 2, no. 1-2, 1999, pp. 3-13.

[78] F. Douglis et al., "Rate of change and other metrics: a live study of the world wide web," *Proc. USENIX Symp. on Internet Technologies and Systems (USITS '97)*, USENIX Association, 1997, p. 14.

[79] L. Breslau et al., "Web Caching and Zipf-like Distributions: Evidence and Implications," *Proc. 18th Ann. Joint Conf. IEEE Computer and Communications Societies (INFOCOM '99)*, IEEE CS, 1999, pp. 126-134.