**ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ**

**«ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ»**

ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΑ ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

**ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ**

# «Εφαρμογή της Θεωρίας Βέλτισης Παύσης στον έλεγχο συνέπειας (consistency) σε WWW Caching Servers»

ΛΟΡΕΝΤΖΟΣ ΔΗΜΗΤΡΙΟΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ
ΧΑΤΖΗΕΥΘΥΜΙΑΔΗΣ ΕΥΣΤΑΘΙΟΣ

ΠΑΤΡΑ
ΣΕΠΤΕΜΒΡΙΟΣ 2011

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

# «Εφαρμογή της Θεωρίας Βέλτισης Παύσης στον έλεγχο συνέπειας (consistency) σε WWW Caching Servers»

ΛΟΡΕΝΤΖΟΣ ΔΗΜΗΤΡΙΟΣ

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να εκφράσω τις ευχαριστίες μου στον επιβλέποντα της εργασίας κ. Ευστάθιο Χατζηευθυμιάδη για τη σημαντική καθοδήγηση και καθοριστική βοήθειά του. Επίσης θα ήθελα να ευχαριστήσω τον κ. Μανώλη Σπανουδάκη για την πολύτιμη συμβολή του στην ολοκλήρωση της εργασίας.

# «Εφαρμογή της Θεωρίας Βέλτισης Παύσης στον έλεγχο συνέπειας (consistency) σε WWW Caching Servers»

**ΛΟΡΕΝΤΖΟΣ ΔΗΜΗΤΡΙΟΣ**

| Ονοματεπώνυμο Επιβλέποντα | Ονοματεπώνυμο Μέλους 1 | Ονοματεπώνυμο Μέλους 2 |
|---|---|---|
| **ΧΑΤΖΗΕΥΘΥΜΙΑΔΗΣ** | **ΜΟΣΧΟΛΙΟΣ** | **ΚΑΨΑΛΗΣ** |
| **ΕΥΣΤΑΘΙΟΣ** | **ΙΩΑΝΝΗΣ** | **ΒΑΣΙΛΕΙΟΣ** |

**Περίληψη**

Ο Παγκόσμιος Ιστός (World Wide Web) είναι αναμφισβήτητα πολύ δημοφιλής καθώς η χρήση του γίνεται καθημερινά από ανθρώπους σε όλο τον κόσμο. Ωστόσο, η χρηστικότητά του απειλείται από την διαρκώς αυξανόμενη δημοτικότητά του. Ο συνωστισμός στον Ιστό αυξάνεται συνεχώς λόγω της σταθερής ανόδου του αριθμού των χρηστών. Για τη βελτίωση της πρόσβασης στο Δίκτυο η τεχνολογία του Web Caching στοχεύει στη μείωση της διάδοσης περιττής πληροφορίας με τη χρήση αντιγράφων. Ένα πρόβλημα που εμφανίζει η χρήση Web αντιγράφων είναι ότι το αντικείμενο που γίνεται διαθέσιμο από ένα ενδιάμεσο διακομιστή μπορεί να μην είναι ενημερωμένο όπως το αυθεντικό αντικείμενο στον πηγαίο διακομιστή. Για τον περιορισμό του προβλήματος αυτού, στην παρούσα εργασία προτείνεται ένα μοντέλο που εφαρμόζει την Θεωρία της Βέλτιστης Παύσης και αναπτύσσεται πρόγραμμα σε Java για τον έλεγχό του.

**Λέξεις κλειδιά :** Συνέπεια, Ενδιάμεσος Εξυπηρετητής, Θεωρία Βέλτιστης Παύσης.

**Abstract**

The World Wide Web is without questioning very popular since it is being used every day by people all over the world. However, its utility is being threatened by its ever growing popularity. Traffic on the Web is continuously increasing due to the steady growth of people using it. To improve access to the Web, Web Caching technology aims at reducing transmission of redundant information by using copies. A problem that occurs when using Web copies is that the item obtained from a Web cache may not be up to date with the original item in the origin server. In order to limit this problem a model applying the Optimal Stopping Theory is proposed and a program in Java is implemented in this project.

**Keywords :** Consistency, Caching Server, Optimal Stopping Theory.

# ΠΕΡΙΕΧΟΜΕΝΑ

**ΠΕΡΙΕΧΟΜΕΝΑ ..... 7**

**1 Caching σε Συστήματα Μνήμης ..... 9**

**2 Web Caching ..... 11**

2.1 Γενικά ..... 11

2.2 Ενδιάμεσος Εξυπηρετητής (Web Proxy) ..... 13

2.3 Πλεονεκτήματα Web Caching ..... 15

2.4 Προκλήσεις Web Caching ..... 15

**3 Web Cache Consistency ..... 19**

3.1 Τύποι Cache Συνέπειας ..... 19

3.2. Μηχανισμοί Ασθενούς Συνέπειας ..... 20

3.2.1 Adaptive TTL ..... 20

3.2.2 Piggyback Cache Validation ..... 22

3.3 Μηχανισμοί Ισχυρής Συνέπειας ..... 23

3.3.1 Polling Every-Time ..... 23

3.3.2 Leases ..... 24

**4 Μοντέλο Ασθενούς Συνέπειας ..... 26**

**5 Θεωρία Βέλτιστης Παύσης (Optimal Stopping Theory) ..... 30**

5.1 Εισαγωγή ..... 30

5.2 Καθορισμός του Προβλήματος Παύσης ..... 31

5.3 Προβλήματα Πεπερασμένου Ορίζοντα - Πρόβλημα της Γραμματέως ..... 34

5.4 Αλγόριθμοι Βέλτιστης Χρονικής Παύσης ..... 36
5.4.1 Odds Algorithm ..... 36
5.4.2 Odds-Algorithm με σειριακή εκτίμηση των odds ..... 38

**6 Προσομοιώσεις - Αποτελέσματα ..... 41**
6.1 Caching Server - Αρχείο Αιτήσεων ..... 41
6.2 Γεννήτρια Αντικειμένων - Αρχείο Τροποποιήσεων ..... 43
6.3 Προσομοίωση μηχανισμού ασθενούς συνέπειας Adaptive TTL ..... 44
6.4 Προσομοίωση Odds Algorithm with Sequential Estimation ..... 47
6.4.1 Προσδιορισμός της Συνάρτησης U ..... 48
6.4.2 Εφαρμογή Αλγορίθμου Odds ..... 50

**7 Συμπεράσματα ..... 59**

**Βιβλιογραφία ..... 62**

**Παράρτημα ..... 63**

# 1 Caching σε Συστήματα Μνήμης

Η έννοια του caching, δηλαδή της δημιουργίας και αποθήκευσης αντιγράφων για τη βελτίωση της απόδοσης ενός συστήματος, είναι ευρέως διαδεδομένη στο χώρο της επιστήμης των υπολογιστών. Για παράδειγμα, αρχιτεκτονικές μνήμης κάνουν χρήση της τεχνικής caching ώστε να αυξηθεί η απόδοση ενός υπολογιστή [1]. Οι σχεδιαστές εκμεταλλεύονται το γεγονός ότι οι κεντρικές μονάδες επεξεργασίας (ΚΜΕ) λειτουργούν σε πολύ υψηλές ταχύτητες σε σχέση με τα συστήματα μνήμης. Βασισμένοι σε αυτό, σχεδιάζουν ένα μικρό κομμάτι μνήμης, που ονομάζεται μνήμη cache και λειτουργεί σε περίπου ίδια ταχύτητα όσο και η ΚΜΕ. Με αυτό τον τρόπο, όταν η ΚΜΕ εντοπίζει την πληροφορία που αναζητεί μέσα στη μνήμη cache (αναφέρεται ως cache hit), αποκτά άμεσα πρόσβαση σε αυτήν και δεν χρειάζεται να ανατρέξει στην κύρια μνήμη, μια διαδικασία σαφώς πιο χρονοβόρα.

Στην περίπτωση που η ΚΜΕ δεν βρίσκει την απαιτούμενη πληροφορία στη μνήμη cache (αναφέρεται ως cache miss), απευθύνεται στην κύρια μνήμη, επιβαρύνοντας έτσι την εκτέλεση των λειτουργιών. Τυπικά, με βάση τη λογική ότι ένα αντικείμενο που έχει πρόσφατα ζητηθεί έχει μεγαλύτερη πιθανότητα να ζητηθεί ξανά στο μέλλον, η ΚΜΕ αποθηκεύει τη νέα πληροφορία στη μνήμη cache.

Επιπλέον, η εφαρμογή της τεχνικής caching στα υπολογιστικά συστήματα συνοδεύεται και από προκλήσεις. Για παράδειγμα, το σύστημα πρέπει να προβλέπει και να αντιμετωπίζει το πρόβλημα, που είναι γνωστό ως πρόβλημα συνέπειας (coherence problem), ότι η πληροφορία που αποθηκεύεται στη μνήμη cache ενδέχεται να μην είναι πρόσφατη. Η χρήση μη ενημερωμένης πληροφορίας επιφέρει συνέπειες στην εκτέλεση των λειτουργιών απο την ΚΜΕ.

Ακόμη, η cache δεν είναι ανεξάντλητη. Κάποια στιγμή οι πόροι της εξαντλούνται, με αποτέλεσμα το σύστημα να πρέπει να εφαρμόσει αλγορίθμους αντικατάστασης ώστε να εξασφαλιστεί χώρος για καινούρια πληροφορία. Η αντιμετώπιση αυτών των ζητημάτων

γίνεται πάντα με στόχο την βελτιστοποίηση της λειτουργίας της μνήμης cache για την αύξηση της πιθανότητας να επιτευχθεί ένα cache hit.

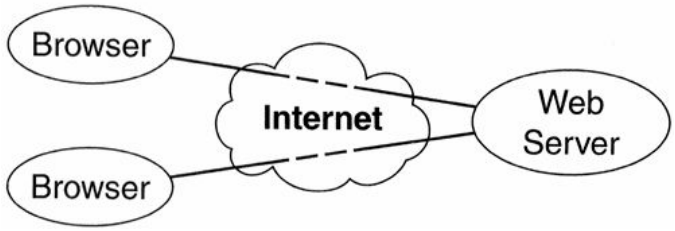
Ένας άλλος χώρος όπου έχει βρει εφαρμογή η έννοια του caching είναι το Διαδίκτυο (Internet) και πιο συγκεκριμένα ο Παγκόσμιος Ιστός (World Wide Web – WWW). Η βελτίωση των επιδόσεων στο Δίκτυο υπήρξε πεδίο μελέτης ήδη από τις αρχές του ’90. Προς αυτή την κατεύθυνση, η χρήση της μεθόδου caching στο Web έχει αναγνωριστεί ως μια πολύ αποτελεσματική τεχνική.

# 2 Web Caching

## 2.1 Γενικά

Ο Παγκόσμιος Ιστός (World Wide Web) μπορεί να θεωρηθεί ως ένα τεράστιο κατανεμημένο σύστημα πληροφοριών που παρέχει πρόσβαση σε διαμοιραζόμενα αντικείμενα [2]. Ένα τέτοιο σύστημα, στην πιο απλή μορφή του, αποτελείται από δύο μέρη που προσπαθούν να επικοινωνήσουν μεταξύ τους. Οι διακομιστές (servers), περιέχουν τα δεδομένα και τις πληροφορίες, ενώ οι πελάτες (clients) αιτούνται κομμάτια πληροφορίας, που ονομάζονται αντικείμενα (objects). Οποιαδήποτε συσκευή με πρόσβαση στο διαδίκτυο, π.χ. προσωπικός υπολογιστής, palmtops, κινητά τηλέφωνα, μπορεί να αποτελέσει ένα χρήστη, εφ'όσον για την επικοινωνία του με το διακομιστή βασίζεται σε καθορισμένα πρωτόκολλα, όπως π.χ. το πρωτόκολλο HTTP.

Μια απλουστευμένη παράσταση της αρχιτεκτονικής του διαδικτύου φαίνεται στην παρακάτω εικόνα [2]. Ο χρήστης δίνει εντολή στο πρόγραμμα περιήγησης να «κατεβάσει» ένα συγκεκριμένο αντικείμενο, π.χ. εισάγοντας ένα παγκοσμίως μοναδικό όνομα, το URL, που οδηγεί στο αντικείμενο αυτό. Το πρόγραμμα περιήγησης, με τη σειρά του, μεταβιβάζει την εντολή στον αντίστοιχο διακομιστή, που αναφέρεται στο εν λόγω URL. Ο διακομιστής εξυπηρετεί την εντολή και στέλνει το αντικείμενο πίσω στο προγραμμα περιήγησης, το οποίο τελικά προβάλλει το αντικείμενο στο χρήστη.




Εικόνα 2.1. Επικοινωνία προγράμματος περιήγησης και εξυπηρετητή [2]

Ωστόσο, η απλουστευμένη αρχιτεκτονική που παρουσιάζεται στην παραπάνω εικόνα περιγράφει αφηρημένα τον Παγκόσμιο Ιστό και ανταποκρίνεται μόνο στην περίπτωση που το εύρος ζώνης του δικτύου και η χωρητικότητα των διακομιστών είναι ανεξάντλητα. Οι πόροι, όμως, του δικτύου και των διακομιστών έχουν κάποια όρια.

Επιπλέον, το γεγονός ότι το διαδίκτυο περιέχει πληθώρα πληροφοριών που απειχούν σε ένα ευρύ πεδίο ενδιαφερόντων, π.χ. επιστημονική έρευνα, ταξίδια, αγορές, ειδησειογραφία, αθλητικά, καιρός, εκπαίδευση, κ.α. και ότι η πρόσβαση στο Διαδίκτυο είναι προσιτή στους περισσότερους, έχουν κάνει πολύ δημοφιλή τη χρήση του Ιστού, με αποτέλεσμα την εκρηκτική ανάπτυξη της δικτυακής κυκλοφορίας.

Με βάση τα παραπάνω, η εκθετική αύξηση της πληροφορίας στο Διαδίκτυο δεν ακολουθείται από ανάλογη αύξηση στην υποδομή. Η διόγκωση της διακίνησης αιτημάτων και δεδομένων μεταξύ χρηστών και διακομιστών στον Ιστό προκαλεί δύο σημαντικά προβλήματα, τη συμφόρηση του δικτύου και την υπερφόρτωση των διακομιστών. Το αποτέλεσμα είναι να αυξάνεται η καθυστέρηση πρόσβασης που αντιλαμβάνεται ο χρήστης.

Το πρόβλημα της κλιμάκωσης του Δικτύου, όπως συχνά αναφέρεται η επίτευξη επιτρεπτής απόδοσης και αξιοπιστίας παρά τη συνεχή ανάπτυξη του διαδικτύου, δεν μπορεί να αντιμετωπιστεί με αύξηση της χωρητικότητας του δικτύου και των διακομιστών των Εταιρειών Παροχής Υπηρεσιών Διαδικτύου (ISPs) [1]. Και αυτό γιατί, παράλληλα αυξάνεται τόσο η χρήση του δικτύου από τους χρήστες, όσο και η χρησιμοποίηση όλο και πιο απαιτητικών εφαρμογών. Έτσι, λοιπόν, αυξάνεται και η απαίτηση για ολοένα και περισσότερους πόρους για το δίκτυο και τους διακομιστές.

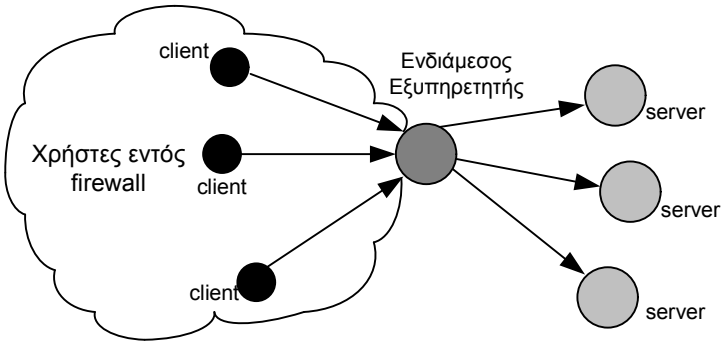
Μια λύση για την αποσυμφόρηση των υπηρεσιών διαδικτύου, τη μείωση της κυκλοφορίας στο Δίκτυο και την αρμονική ανάπτυξη του Ιστού προσφέρει η χρήση των web caches. Η στρατηγική αυτή έχει τις ρίζες της στην τεχνική caching που εφαρμόζεται στα συστήματα μνήμης και βασίζεται στην απλή ιδέα της δημιουργίας και αποθήκευσης

αντιγράφων αντικειμένων σε ένα εύκολα προσβάσιμο χώρο, για τη μελλοντική χρήση τους.

## 2.2 Ενδιάμεσος Εξυπηρετητής (Web Proxy)

Η τεχνική του Web caching είναι παρόμοια με την περίπτωση του caching σε ένα σύστημα μνήμης υπολογιστή. Αφορά στην αποθήκευση του αντιγράφου Web αντικειμένων που έχουν ζητηθεί από χρήστες (clients) σε ένα ενδιάμεσο διακομιστή (caching server), για την εξυπηρέτηση μελλοντικών αιτήσεων. Στην περίπτωση που υπάρχει μια τέτοια αίτηση από ένα χρήστη, το αντίγραφο του αντικειμένου παραδίδεται κατευθείαν από τον ενδιάμεσο διακομιστή, που αναφέρεται και ως Web caching proxy ή εν συντομία proxy, χωρίς να γίνεται επικοινωνία με τον πηγαίο εξυπηρετητή (origin server).

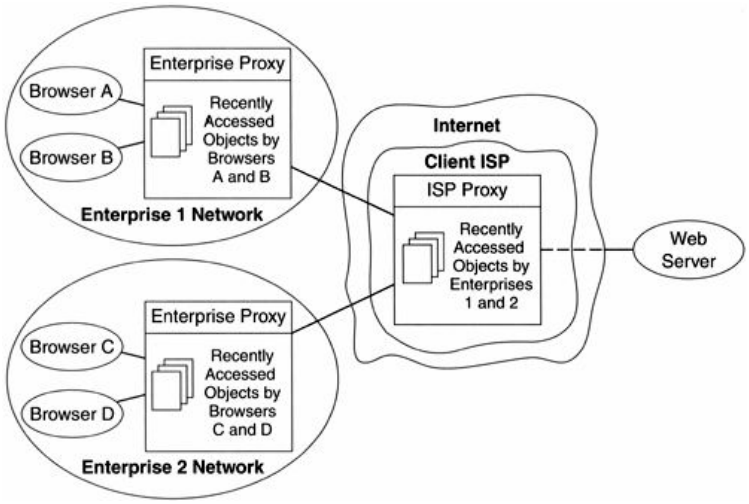
Η ιδέα για τη χρήση των διακομιστών proxy προέκυψε όταν είχαν πρωτοχρησιμοποιηθεί από οργανισμούς για να επιτρέπουν τη σύνδεση των χρηστών τους στο διαδίκτυο μέσω ενός firewall [3]. Για λόγους ασφαλείας, εταιρείες χρησιμοποιούσαν ειδικού τύπου HTTP ενδιάμεσους διακομιστές πάνω σε μηχανές firewall. Οι διακομιστές αυτοί αφού επεξεργάζονταν αιτήματα από τους χρήστες που βρίσκονταν πίσω από το firewall, τα προωθούσαν στους απομακρυσμένους εξυπηρετητές και στη συνέχεια συγκέντρωναν τις απαντήσεις και τις προωθούσαν κατάλληλα στους χρήστες.




Εικόνα 2.2. Σύνδεση χρηστών με διακομιστές μέσω ενδιάμεσου εξυπηρετητή που λειτουργεί πάνω σε firewall

Προέκυψε, λοιπόν, το ερώτημα κατά πόσο ήταν αποδοτική η αποθήκευση αρχείων στους proxies. Δεδομένου ότι, συνήθως, χρήστες που βρίσκονται πίσω από κοινό firewall ανήκουν στον ίδιο οργανισμό, έχουν πιθανόν και τα ίδια ενδιαφέροντα. Αποκτούν, επιπλέον, πρόσβαση στα ίδια αρχεία, στα οποία πιθανότατα επανέρχονται επανειλημμένα. Κατά συνέπεια, ένα αρχείο που έχει ζητηθεί και έχει αποθηκευτεί σε ένα proxy είναι πιθανό να ξαναζητηθεί. Επομένως, η χρήση των ενδιάμεσων διακομιστών για τη διαδικασία caching όχι μόνο εξοικονομεί τη χωρητικότητα του δικτύου, αλλά μειώνει και την καθυστέρηση πρόσβασης του αντικειμένου από τους χρήστες.

Ένας ενδιάμεσος διακομιστής, λοιπόν, εξυπηρετεί τα αιτήματα ενός χρήστη, αλλά ταυτόχρονα παίζει το ρόλο του χρήστη προωθώντας αιτήματα στον πηγαίο εξυπηρετητή. Για παράδειγμα, όπως φαίνεται στην παρακάτω εικόνα, οι περιηγητές Ιστού A, B, C, D, που ανήκουν σε διαφορετικά εταιρικά δίκτυα, χρησιμοποιούν τον εταιρικό proxy για να ικανοποιήσουν τα αιτήματά τους [2]. Αυτός με τη σειρά του χρησιμοποιεί τον proxy της Εταιρείας Παροχής Υπηρεσιών Διαδικτύου (ISP) για να εξυπηρετήσει τα αιτήματα που δεν μπορεί ο ίδιος να ικανοποιήσει. Στην περίπτωση που κάποια αιτήματα παραμένουν εντούτοις ανικανοποίητα, προωθούνται στον πηγαίο εξυπηρετητή, ο οποίος παρέχει το ζητούμενο αντικείμενο.




Εικόνα 2.3. Εξυπηρέτηση περιηγητών Ιστού μέσω πολλαπλών ενδιάμεσων εξυπηρετητών [2]

## 2.3 Πλεονεκτήματα Web Caching

Η εφαρμογή της τεχνικής caching με τη χρήση ενδιάμεσων διακομιστών επιφέρει σημαντικά πλεονεκτήματα γενικότερα σε όλο το Διαδίκτυο, αλλά και ειδικότερα σε καθένα από τα μέρη που απαρτίζουν τον Παγκόσμιο Ιστό, δηλαδή τους χρήστες, τους Πάροχους Υπηρεσιών Διαδικτύου (ISPs) και τους πηγαίους εξυπηρετητές [2].

Όσον αφορά το Διαδίκτυο με τη χρήση caching μειώνεται το συνολικό φορτίο στα επιμέρους δίκτυα. Το αποτέλεσμα είναι να επωφελείται η αρμονική ανάπτυξη του Διαδικτύου.

Για τον χρήστη, η χρήση του αποθηκευμένου αρχείου από τον proxy μειώνει την καθυστέρηση της απάντησης από έναν απομακρυσμένο server. Επιπλέον, ο χρήστης, λόγω της σύνδεσής του με τον ενδιάμεσο διακομιστή, δεν απαιτείται να πραγματοποιεί εκ νέου σύνδεση με τον πηγαίο εξυπηρετητή που κάθε φορά επισκέπτεται.

Όσο για τους ISPs, πέρα από τη μείωση της κίνησης και της συμφόρησης στο δίκτυο τους, τα οφέλη είναι και οικονομικά. Με τη χρήση του ενδιάμεσου εξυπηρετητή τα αιτήματα από τους χρήστες ικανοποιούνται μέσα στο δίκτυο του, με αποτέλεσμα να μειώνεται το εύρος ζώνης που δεσμεύει ο ISP από άλλους Πάροχους Υπηρεσιών Διαδικτύου.

Τέλος, για τον πηγαίο εξυπηρετητή η χρήση αποθηκευμένων αντιγράφων μειώνει το φορτίο που χειρίζεται, καθώς μειώνεται ο αριθμός των αιτημάτων που καλείται να ικανοποιήσει, και περιορίζει το κόστος συντήρησης.

## 2.4 Προκλήσεις Web Caching

Παρ' όλο που ο Παγκόσμιος Ιστός μπορεί γενικευμένα να θεωρηθεί ως ένα ευρύ σύστημα διαμοιραζόμενης πληροφορίας, η εφαρμογή του caching στο Web εμφανίζει διαφορές σε σχέση με την εφαρμογή του caching στα υπολογιστικά συστήματα. Οι

διαφορές αυτές πηγάζουν κυρίως από το τεράστιο εύρος του Internet, την ανομοιογένεια στο μέγεθος των Web αντικειμένων και τη δυνατότητα χρήσης αντιγράφων Web αντικειμένων [2]. Πράγματι, λόγω της ευρείας κλίμακας του Διαδικτύου υπεισέρχονται θέματα, όπως οι αποστάσεις μεταξύ των χρηστών και των εξυπηρετητών, η συμφόρηση του δικτύου και ο φόρτος του διακομιστή, που πρέπει να ληφθουν υπόψη.

Επιπλέον, όσον αφορά το μέγεθος των αντικειμένων, παρακολουθείται τόσο ο συνολικός ρυθμός επιτυχίας για τα αντικείμενα (object hit rate), δηλαδή το ποσοστό των αιτήσεων που ικανοποιούνται από τον cache διακομιστή, όσο και το ποσοστό των bytes που ικανοποιούνται από τον cache διακομιστή (byte hit rate).

Τέλος, ορισμένα Web αντικείμενα είναι δύσκολο ή ακόμα και ακατόρθωτο να αντιγραφούν σε ένα cache server, π.χ. όταν το υλικό αλλάζει δυναμικά ή είναι χαρακτηριστικό για ένα χρήστη.

Παρ'όλες τις σημαντικές διαφορές μεταξύ της εφαρμογής του caching σε ένα σύστημα μνήμης υπολογιστή και σε ένα caching server στο Διαδίκτυο, οι προκλήσεις παραμένουν ίδιες. Ένας cache διακομιστής έχει περιορισμένους πόρους και ενδέχεται να εξυπηρετεί ένα πολύ μεγάλο αριθμό αιτήσεων από χρήστες. Συνεπώς, πρέπει να προνοήσει για την εξασφάλιση του απαραίτητου χώρου. Αυτό γίνεται με τη χρήση αλγορίθμων αντικατάστασης για τη διαγραφή κάποιων από τα ήδη υπάρχοντα αντίγραφα αντικειμένων ώστε να αποθηκευτούν αντίγραφα καινούριων αντικειμένων. Μία ιδιαίτερα γνωστή προσέγγιση είναι η αντικατάσταση του αντικειμένου που χρησιμοποιήθηκε λιγότερο.

Ωστόσο, οι στόχοι της χρήσης των cache servers, όπως η μείωση του όγκου της πληροφορίας που ανταλλάσεται στο δίκτυο και της υστέρησης που αντιλαμβάνεται ο χρήστης, οδηγούν σε σύνθετες αποφάσεις για την αντικατάσταση του περιεχομένου σε ένα caching server. Αυτές οι αποφάσεις συνδέονται με τη χρησιμότητα διατήρησης ενός πόρου στην cache, που είναι συνάρτηση πολλών παραγόντων όπως:

- Το κόστος ανάκτησης του πόρου

- Το κόστος αποθήκευσης του πόρου
- Ο αριθμός των προσβάσεων στο πόρο κατά το παρελθόν
- Η πιθανότητα προσπέλασης του πόρου στο άμεσο μέλλον
- Ο χρόνος από την τελευταία μεταβολή του πόρου
- Ο χρόνος λήξης που προσδιορίζεται ευρεστικά

Μια ακόμη πρόκληση που καλείται να αντιμετωπίσει ένας caching server είναι το πρόβλημα συνέπειας (consistency). Ενδέχεται το αντίγραφο που διαθέτει ο cache διακομιστής να είναι ξεπερασμένο σε σχέση με αυτό που παρέχει ο πηγαίος εξυπηρετητής. Αυτό σημαίνει ότι η πληροφορία που διαμοιράζεται από τον caching server δεν ανταποκρίνεται στην πραγματικότητα, πράγμα που προκαλεί ευνόητα προβλήματα. Επομένως, πρέπει να υπάρχει ένας μηχανισμός που να προλαμβάνει την κατάσταση αυτή, να εξασφαλίζει δηλαδή ότι ο πόρος που είναι αποθηκευμένος στον cache εξυπηρετητή είναι έγκυρος και να παρέχει στο χρήστη όσο το δυνατό πιο πρόσφατα ενημερωμένη πληροφορία.

Η διατήρηση της συνέπειας εξαρτάται από τους πόρους που διαθέτει ο ενδιάμεσος εξυπηρετητής και από την πολιτική που ακολουθείται. Οι διακομιστές μπορεί απλά να επιστρέφουν ένα παλιό αποθηκευμένο αντίγραφο μαζί με μια αιτία για την απαξίωση του. Μια από τις αιτίες αυτές μπορεί να είναι η αδυναμία εγκατάστασης σύνδεσης προς τον πηγαίο διακομιστή ή ο υψηλός φόρτος του caching server. Σε άλλη περίπτωση, ο περιορισμός *only-if-cached* που υπεισέρχεται στην επικεφαλίδα *Cache-control* σε ένα HTTP αίτημα του πελάτη, υποχρεώνει τον caching server να επιστρέψει μία ήδη αποθηκευμένη πληροφορία χωρίς να ενδιαφέρει η εγκυρότητά της.

Επιπλέον, έλεγχος της εγκυρότητας της πληροφορίας που διαθέτει ο ενδιάμεσος διακομιστής γίνεται με τη χρήση επικεφαλίδων στις HTTP αιτήσεις του προς τον πηγαίο εξυπηρετητή, π.χ. με την αποστολή ενός *GET* ή *HEAD* αιτήματος με τη χρήση της επικεφαλίδας *if-modified-since*, που δείχνει τη χρονική στιγμή της τελευταίας μεταβολής του πόρου όπως υποδεικνύεται από τον origin server. Ο πηγαίος διακομιστής απαντά με

ένα επίκαιρο αντίγραφο του πόρου, όταν έχει υπάρξει τροποποίηση του πόρου από τη χρονική στιγμή της τελευταίας μεταβολής, ή με την απάντηση 304 Not Modified, όταν ο πόρος δεν έχει τροποποιηθεί.

# 3 Web Cache Consistency

## 3.1 Τύποι Cache Συνέπειας

Ένα από τα πλεονεκτήματα της χρήσης των ενδιάμεσων διακομιστών για την εφαρμογή του Web Caching είναι η μείωση στην καθυστέρηση πρόσβασης που αντιλαμβάνεται ο χρήστης στο Διαδίκτυο. Ωστόσο, υπάρχει η ανάγκη ύπαρξης μηχανισμών που εξασφαλίζουν την cache συνέπεια, δηλαδή εξασφαλίζουν ότι τα αντίγραφα αρχείων που είναι αποθηκευμένα σε ένα caching server ενημερώνονται ώστε να διατηρηθεί η συνέπεια με την πρωτότυπη πληροφορία στον πηγαίο εξυπηρετητή.

Υπάρχουν δύο βασικές προσεγγίσεις που αφορούν στη συνέπεια cache [2]. Με την προσέγγιση της ασθενούς συνέπειας (weak consistency) ο χρήστης είναι δυνατό να λάβει από ένα caching server, ως απάντηση στο αίτημά του, πληροφορία που δεν είναι ενημερωμένη. Και αυτό γιατί ο έλεγχος για την εγκυρότητα της πληροφορίας γίνεται μόνο περιοδικά. Σύμφωνα με τη διαδικασία που είναι γνωστή ως validation, ο ενδιάμεσος εξυπηρετητής επικοινωνεί με τον πηγαίο διακομιστή για να διακριβώσει την εγκυρότητα των αποθηκευμένων αντικειμένων. Στην πράξη, η επικοινωνία αυτή γίνεται τυπικά τη στιγμή που υπάρχει αίτημα για κάποιο αντικείμενο στον caching server, πράγμα που αυξάνει το χρόνο απόκρισης.

Επιπλέον, με τη χρήση του validation ο ρόλος του πηγαίου διακομιστή περιορίζεται στην επαλήθευση της εγκυρότητας του ζητούμενου αντικειμένου. Παρ'όλα αυτά στην περίπτωση που το αντικείμενο δεν έχει αλλάξει είναι δυνατό να αυξηθεί τόσο ο φόρτος του origin server όσο και η κυκλοφορία στο δίκτυο λόγω ανταλλαγής μη αναγκαίων μηνυμάτων μεταξύ του ενδιάμεσου και του πηγαίου διακομιστή.

Μια άλλη προσέγγιση που αφορά στη συνέπεια cache είναι η ισχυρή συνέπεια (strong consistency), η οποία εγγυάται ότι η πληροφορία που παραδίδεται από τον

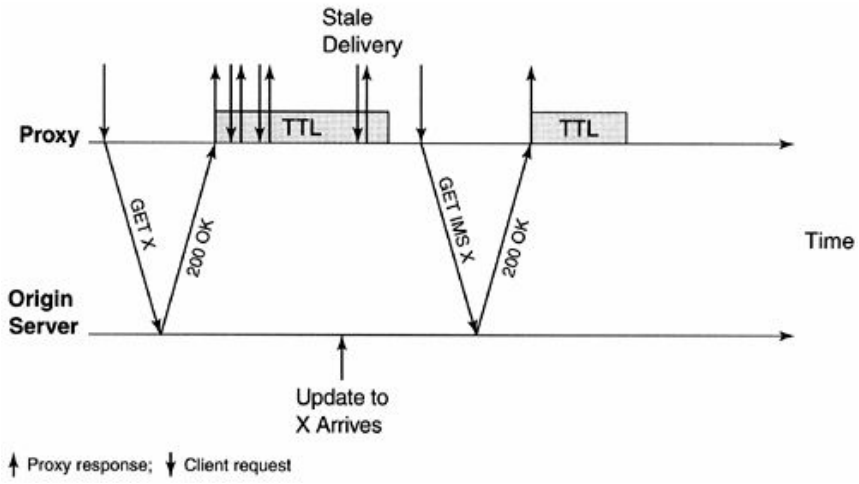
caching server στο χρήστη είναι έγκυρη και ενημερωμένη [2]. Αυτό μπορεί να γίνει όταν ο πηγαίος εξυπηρετητής ενημερώνει τους ενδιάμεσους διακομιστές όποτε συμβαίνει κάποια αλλαγή στα αντικείμενα που είναι αποθηκευμένα στον caching server. Με τον τρόπο αυτό τα αντίγραφα που διαθέτει στους χρήστες ο caching server είναι πάντα έγκυρα και ενημερωμένα. Η διαδικασία αυτή ονομάζεται invalidation και σύμφωνα με αυτή ο πηγαίος διακομιστής παρακολουθεί τα αντίγραφα της πληροφορίας που έχει διαθέσει στους caching servers. Σε αυτή την περίπτωση ο πηγαίος διακομιστής πρέπει να διατηρεί λίστες με τους ενδιάμεσους διακομιστές στους οποίους έχει στείλει κάθε αρχείο του.

Σε γενικές γραμμές μπορεί να αναφερθεί ότι η ισχυρή συνέπεια παρέχει πάντα ένα ενημερωμένο αντίγραφο στο χρήστη, με το τίμημα της πολυπλοκότητας, λόγω της διατήρησης της κατάστασης του πηγαίου εξυπηρετητή. Στον αντίποδα, η ασθενής συνέπεια εξοικονομεί εύρος ζώνης για το δίκτυο, με τίμημα την επιστροφή μη ενημερωμένης πληροφορίας στο χρήστη [7].

## 3.2. Μηχανισμοί Ασθενούς Συνέπειας

### 3.2.1 Adaptive TTL

Η προσέγγιση της ασθενούς συνέπειας προϋποθέτει ότι ο ενδιάμεσος εξυπηρετητής επικοινωνεί με τον πηγαίο διακομιστή για να εξακριβώσει αν τα αρχεία που διαθέτει στους χρήστες είναι έγκυρα. Η επικοινωνία του caching server με τον origin server συνοψίζεται στην παρακάτω εικόνα.




Εικόνα 3.1. Παράδειγμα Ασθενούς Συνέπειας [2]

Μόλις ο ενδιάμεσος εξυπηρετητής παραλάβει το αίτημα από το χρήστη, ελέγχει αν μπορεί να εξυπηρετήσει την αίτηση απ'ευθείας, διαφορετικά επικοινωνεί με τον πηγαίο διακομιστή για να αποκτήσει το ζητούμενο αρχείο. Κάθε αρχείο συνοδεύεται από ένα «χρόνο ζωής» (time to live – TTL) που μπορεί να επιβάλλεται ρητά (explicit TTL) από τον origin server ή να προσδιορίζεται έμμεσα (implicit TTL) [2]. Ο caching server θεωρεί το αρχείο έγκυρο και συνεπώς έχει το δικαίωμα να το διαμοιράζει στους χρήστες μέχρι τη λήξη του TTL. Σε ενδεχόμενη νέα αίτηση για το ίδιο αρχείο όταν πλέον έχει λήξει ο χρόνος αυτός, ο caching server, σύμφωνα με τη διαδικασία validation, πρέπει να ελέγξει την εγκυρότητα του αντιγράφου του επικοινωνώντας με τον origin server. Γίνεται προφανές από την εικόνα ότι αν ενημερωθεί το αρχείο στον origin server πριν τη λήξη του TTL, ο ενδιάμεσος εξυπηρετητής διαθέτει ένα μη έγκυρο αρχείο που ενδεχομένως να το παρέχει στους χρήστες.

Το χρονικό διάστημα explicit TTL είναι προκαθορισμένο και ορίζεται στα πεδία **`expires`** ή **`max-age`** της κεφαλίδας της HTTP απόκρισης. Επειδή, όμως, η πληροφορία αυτή πολλές φορές παραλείπεται, υπάρχει ανάγκη για ορισμό του implicit TTL χρησιμοποιώντας ευρετικές μεθόδους. Η χρήση ενός σταθερού TTL δεν είναι αξιόπιστη,

καθώς η επιλογή μικρής τιμής οδηγεί σε ανταλλαγή μη αναγκαίων μηνυμάτων για την εξακρίβωση της εγκυρότητας των αρχείων, ξοδεύοντας τους πόρους του συστήματος και του δικτύου. Από την άλλη μεριά, η επιλογή μεγάλης τιμής οδηγεί σε παράδοση μη ενημερωμένης πληροφορίας στους χρήστες [8].

Για το λόγο αυτό έχει υοθετηθεί ο μηχανισμός του adaptive TTL, που καθορίζει το «χρόνο ζωής» ενός αντικειμένου σύμφωνα με την παρατήρηση ότι όταν ένα αρχείο δεν έχει αλλάξει για πολύ καιρό, το πιο πιθανό είναι να μην αλλάξει και στο μέλλον. Ο μηχανισμός αυτός, που προέρχεται από το σύστημα αρχείων Alex, υπολογίζει το TTL ως ένα ποσοστό της «σχετικής ηλικίας» του αρχείου, δηλαδή του χρονικού διαστήματος από τη στιγμή που ενημερώθηκε τελευταία φορά (η πληροφορία αυτή περιέχεται στην κεφαλίδα `last-modified` της HTTP απόκρισης) μέχρι τη στιγμή που στάλθηκε από τον origin server στον caching server (η πληροφορία αυτή περιέχεται στην κεφαλίδα `date` της HTTP απόκρισης) [2].

Σύμφωνα με το [5] η λειτουργία του caching server που πρωτοχρησιμοποιήθηκε στο CERN βασιζόταν στην τεχνική του adaptive TTL, ενώ, σύμφωνα με το [4], τα αποτελέσματα προσομοίωσης, που στηρίχτηκε σε ίχνη, έδειξαν ότι το ποσοστό της μη ενημερωμένης πληροφορίας με τη χρήση του adaptive TTL μπορεί να μειωθεί σε λιγότερο από 5%. Επιπλέον, σύμφωνα με το [8] για το ποσοστό που υπεισέρχεται στον υπολογισμό του TTL το πρωτόκολλο HTTP/1.1 προτείνει ένα ανώτατο όριο 10%, ενώ οι Chankhunthod et al. προτείνουν το 50%. Τέλος, στο [6] αναφέρεται ο σχεδιασμός του ενδιάμεσου εξυπηρετητή Harvest, που χρησιμοποιεί adaptive TTL με διάρκεια TTL καθορισμένη στο 50% του χρόνου από την τελευταία φορά που ενημερώθηκε το αντικείμενο.

### 3.2.2 Piggyback Cache Validation

Άλλος ένας μηχανισμός ασθενούς συνέπειας είναι ο Piggyback Cache Validation. Σύμφωνα με αυτή την τεχνική όταν ο ενδιάμεσος εξυπηρετητής πρέπει να επικοινωνήσει με τον πηγαίο εξυπηρετητή για να αποστείλει μια HTTP αίτηση, ελέγχει αν διαθέτει άλλα αντίγραφα αντικειμένων από τον ίδιο origin server η διάρκεια των οποίων έχει λήξει ή κοντεύει να λήξει [2]. Στη συνέχεια, στην αρχική αίτηση, περιλαμβάνει αιτήσεις επιβεβαίωσης της εγκυρότητας για τα παραπάνω αντικείμενα χρησιμοποιώντας τη μέθοδο HEAD. Με αυτό τον τρόπο, η απόκριση από τον origin server επιστρέφει μεταπληροφορία για το αντικείμενο, δηλαδή μόνο τις επικεφαλίδες (και τα πεδία **`last-modified`** και **`date`**) και όχι το ίδιο το αντικείμενο, ακόμα και αν αυτό έχει αλλάξει. Έπειτα ανάλογα με το αν το αντικείμενο έχει μεταβληθεί ή όχι στον origin server, ο caching server ακυρώνει το αποθηκευμένο αντίγραφο ή ανανεώνει τον χρόνο εγκυρότητας του (TTL).

## 3.3 Μηχανισμοί Ισχυρής Συνέπειας

### 3.3.1 Polling Every-Time

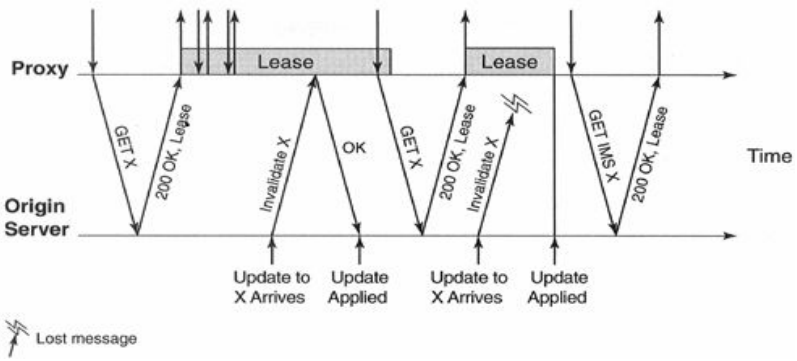
Σύμφωνα με το μηχανισμό αυτό, ο ενδιάμεσος εξυπηρετητής αντιμετωπίζει τα αποθηκευμένα αντικείμενα ως μη έγκυρα και κάθε φορά που παραλαμβάνει αίτηση από χρήστη για κάποιο αντίγραφο επικοινωνεί πρώτα με τον πηγαίο εξυπηρετητή για να επιβεβαιώσει την εγκυρότητα του αντιγράφου και στη συνέχεια το στέλνει στο χρήστη. Είναι προφανές ότι στην περίπτωση που το αντικείμενο δεν μεταβληθεί, η μέθοδος αυτή οδηγεί σε μεγάλο αριθμό μη αναγκαίων μηνυμάτων μεταξύ του ενδιάμεσου και του πηγαίου διακομιστή. [2]

### 3.3.2 Leases

Ένας ακόμη μηχανισμός που εξασφαλίζει ισχυρή συνέπεια είναι η ενοικίαση (lease), δηλαδή ένα «συμβόλαιο» μεταξύ του origin και του caching server που εξασφαλίζει ότι ο πρώτος θα ενημερώνει το δεύτερο για οποιαδήποτε αλλαγή στο αποθηκευμένο αντικείμενο για όσο χρονικό διάστημα ισχύει το «συμβόλαιο» [2].

Σύμφωνα με το μηχανισμό αυτό, ο origin server πρέπει να διατηρεί λίστα με τους ενδιάμεσους εξυπηρετητές στους οποίους έχει στείλει αντικείμενα. Παρ'όλα αυτά, υποχρεούται να διατηρεί τον caching server στη λίστα μόνο για όσο χρόνο διαρκεί το «συμβόλαιο». Για την επίτευξη ισχυρής συνέπειας, κατά τη διάρκεια του «συμβολαίου», όποτε υπάρχει ενημέρωση στο πρωτότυπο αντικείμενο, ο πηγαίος εξυπηρετητής πρέπει να στείλει μήνυμα ακύρωσης στους ενδιάμεσους διακομιστές που περιέχονται στη λίστα του και να επιβάλλει τις αλλαγές στο αντικείμενο μόνο όταν επιβεβαιωθεί ότι όλοι οι caching servers έχουν ακυρώσει την παλιά εκδοχή του αντικειμένου. Επιπλέον, αφού λήξει το «συμβόλαιο» ο ενδιάμεσος διακομιστής πρέπει να επικυρώσει την εγκυρότητα του αντιγράφου του με την πρώτη αίτηση χρήστη που παραλαμβάνει.

Στην παρακάτω εικόνα παρουσιάζεται το σχήμα των «συμβολαίων». Η πρώτη αίτηση που φτάνει στον ενδιάμεσο εξυπηρετητή τον αναγκάζει να επικοινωνήσει με τον πηγαίο εξυπηρετητή για να αποκτήσει το αντικείμενο X. Συνάπτεται, συνεπώς, ένα «συμβόλαιο» και για όσο χρόνο διαρκεί αυτό, ο origin server ενημερώνει τον caching server κάθε φορά που αλλάζει το πρωτότυπο αντικείμενο.




Εικόνα 3.2. Σχήμα μηχανισμού «συμβολαίου» [2]

Μόλις επιβεβαιωθεί ότι ο caching server έλαβε την ειδοποίηση, ο origin server εφαρμόζει τις αλλαγές στο πρωτότυπο αντικείμενο. Την επόμενη φορά που υπάρχει αίτηση από χρήστη, ο ενδιάμεσος διακομιστής ανακτά το έγκυρο αντικείμενο και συνάπτει νέο «συμβόλαιο» με τον πηγαίο διακομιστή.

# 4 Μοντέλο Ασθενούς Συνέπειας

Στο κεφάλαιο αυτό περιγράφεται το μοντέλο ασθενούς συνέπειας που θα υλοποιηθεί στη συνέχεια. Σύμφωνα με το [2], στο μηχανισμό adaptive TTL ο χρόνος ζωής ενός αντικειμένου ορίζεται ως ένα ποσοστό k της «σχετικής ηλικίας» του, δηλαδή του χρονικού διαστήματος που μεσολαβεί από τη στιγμή που ο πόρος ενημερώθηκε τελευταία φορά μέχρι τη στιγμή που στάλθηκε από τον πηγαίο διακομιστή στον ενδιάμεσο διακομιστή. Προσομοίωση που στηρίχθηκε σε πραγματικά ίχνη [2] έδειξε ότι, χρησιμοποιώντας adaptive TTL με k ίσο με 0.2, η παράδοση μη φρέσκου αντικειμένου ανήλθε στο 0.22% του συνόλου των αντικειμένων. Επιπλέον, υπήρξε διακίνηση μη απαραίτητων μηνυμάτων, καθώς σε ποσοστό σχεδόν 59% τα αντικείμενα παρέμειναν αμετάβλητα και οι αιτήσεις επιβεβαίωσης της εγκυρότητας του πόρου που αιτούταν ο caching server είχαν ως αποτέλεσμα τη λήψη αποκρίσεων “Not Modified”. Αν, λοιπόν, ο ενδιάμεσος διακομιστής γνώριζε πότε να επικαιροποιήσει το αντίγραφό του, τότε θα εξαλειφόταν όχι μόνο το ενδεχόμενο διάθεσης ενός πεπαλαιωμένου πόρου στον εκάστοτε χρήστη αλλά και η διακίνηση αχρείαστων μηνυμάτων για την επιβεβαίωση της συνέπειας του αντικειμένου που διαθέτει ο ενδιάμεσος διακομιστής.

Προς αυτή την κατεύθυνση, προτείνεται ένα σχήμα που αξιοποιεί την θεωρία της βέλτιστης παύσης (optimal stopping) και χρησιμοποιεί μια συνάρτηση U πάνω στην οποία θα εφαρμοστεί το κριτήριο της παύσης (stopping rule). Η συνάρτηση αυτή εκφράζει κατά πόσο επιτακτική είναι η ανανέωση ενός πόρου. Επιπλέον, αφού κάθε πόρος ανήκει σε ένα ιστότοπο (site), το πόσο συχνά γίνεται η ανανέωση του μπορεί να θεωρηθεί ότι εξαρτάται τόσο από την πληροφορία που σχετίζεται με το συγκεκριμένο πόρο (π.χ. η ρητή χρονική στιγμή, που ορίζεται στο πεδίο expires της κεφαλίδας της HTTP απόκρισης, μέχρι την οποία μπορεί να θεωρηθεί έγκυρο το αντικείμενο), όσο και από την «κίνηση» των υπόλοιπων πόρων που ανήκουν στο ίδιο site. Πιο συγκεκριμένα,

για τη σύνθεση της συνάρτησης U για ένα αντικείμενο i που ανήκει στον ιστότοπο x λαμβάνονται υπόψη οι παρακάτω παράγοντες :

- Ρυθμός επιτυχίας (hit rate) $\mathbf{h_i}$, που ορίζεται ως $h_i = \frac{\text{επιτυχείς αναζητήσεις αντικειμένων του site x}}{\text{αναφορές για το site x}}$. Η μέγιστη συνεισφορά του παράγοντα h είναι 1, όταν όλες οι αναφορές σε αντικείμενα του ίδιου site είναι επιτυχείς.
- Δημοτικότητα $\mathbf{k_i}$, που ορίζεται από τη σχέση $k_i = \frac{\text{αναφορές για το site x}}{\text{σύνολο αναφορών}}$.
- Παράγοντας $\boldsymbol{\rho_i}$, που δηλώνει την εξάρτηση από το χρόνο παραμονής του αντικειμένου στον ενδιάμεσο διακομιστή και από το χρονικό διάστημα (expiration_time) για το οποίο θεωρούνται έγκυρα τα άλλα αντικείμενα που ανήκουν στο ίδιο site. Ο παράγοντας ρ ορίζεται από τον τύπο $\rho_i = \frac{(\text{current\_time} - \text{load\_time})_i}{\text{average\_expiration\_time}}$, όπου current_time είναι η τρέχουσα χρονική στιγμή, load_time η χρονική στιγμή στην οποία αποθηκεύτηκε το αντικείμενο στον caching server και average_expiration_time το μέσο χρονικό διάστημα για το οποίο θεωρούνται έγκυρα τα αντικείμενα από το ίδιο site. Η τιμή του παράγοντα ρ τείνει στο 1 όταν ο χρόνος παραμονής του αντικειμένου στον caching server πλησιάζει το μέσο expiration time των αντικειμένων στο ίδιο site, γεγονός που επιτάσσει την ανανέωση του πόρου. Στην περίπτωση που ο χρόνος παραμονής είναι μηδέν, δηλαδή το αντικείμενο έχει μόλις «φορτωθεί» στον ενδιάμεσο διακομιστή, η τιμή του παράγοντα ρ είναι μηδέν, δηλαδή δεν υπάρχει ανάγκη για ανανέωση του πόρου.
- Παράγοντας $\boldsymbol{\xi_i}$, που δηλώνει την εξάρτηση από τη χρονική στιγμή (expires_time) μέχρι την οποία ο πόρος i μπορεί να θεωρείται έγκυρος και δίνεται από τη

σχέση $\xi_i = 1 - \frac{(\text{exp ires\_time} - \text{current\_time})_i}{(\text{exp ires\_time} - \text{load\_time})_i}$, όπου current_time είναι η τρέχουσα χρονική στιγμή και load_time η χρονική στιγμή στην οποία αποθηκεύτηκε το αντικείμενο στον caching server. Η τιμή του παράγοντα ξ τείνει στο 1 όταν η τρέχουσα χρονική στιγμή πλησιάζει την χρονική στιγμή μέχρι την οποία ισχύει η εγκυρότητα του αντικειμένου, με αποτέλεσμα να γίνεται ολοένα και πιο επιτακτική η ανάγκη για ενημέρωση του. Αναλόγως, η τιμή του ξ λαμβάνει την μικρότερη τιμή (το μηδέν) όταν το αντικείμενο έχει μόλις αποθηκευτεί στον ενδιάμεσο εξυπηρετητή, δηλαδή όταν current_time και load_time έχουν την ίδια τιμή. Στην περίπτωση αυτή δεν είναι αναγκαία η ανανέωση του πόρου.

Με βάση τα παραπάνω, η συνάρτηση U για τον πόρο i μπορεί να γραφτεί με την μορφή $U_i = w_1 \cdot h_i + w_2 \cdot k_i + w_3 \cdot \rho_i + w_4 \cdot \xi_i$, όπου $w_1$, $w_2$, $w_3$ και $w_4$ είναι τα βάρη που καθορίζουν την επίδραση καθενός από τους παραπάνω παράγοντες. Θεωρείται ότι οι παραπάνω παράγοντες έχουν ισοπίθανη επίδραση στο τελικό αποτέλεσμα και επομένως τα βάρη που αντιστοιχούν στον καθένα είναι $w_1 = w_2 = w_3 = w_4 = ¼ = 0.25$. Ωστόσο, είναι πιθανό, για κάποιο πόρο το πεδίο expires της HTTP απόκρισης να είναι κενό, με αποτέλεσμα να μην είναι διαθέσιμη η πληροφορία για το χρονικό διάστημα για το οποίο μπορεί να θεωρηθεί έγκυρος ο πόρος. Στην περίπτωση αυτή, η επίδραση του παράγοντα ξ θεωρείται μηδενική με αποτέλεσμα τα βάρη $w_1$, $w_2$, $w_3$ και $w_4$ να πάρουν τις τιμές 1/3, 1/3, 1/3 και 0 αντίστοιχα.

Οι τιμές που παίρνει η συνάρτηση U βρίσκονται στο διάστημα [0,1]. Η ελάχιστη τιμή 0 αναφέρεται στην περίπτωση για την οποία δεν χρειάζεται να γίνει ανανέωση του πόρου από τον caching server. Ωστόσο, καθώς η τιμή της U αυξάνεται γίνεται ολοένα και πιο αναγκαία η χρήση ενός ενημερωμένου αντιγράφου από τον ενδιάμεσο διακομιστή. Μάλιστα, για τιμές που πλησιάζουν στο μέγιστο 1 ο caching server πρέπει οπωσδήποτε να προχωρήσει σε ενημέρωση του αντικειμένου.

Στο σημείο αυτό δημιουργείται το ερώτημα ποια είναι η σωστότερη χρονική στιγμή για να γίνει η επικαιροποίηση του πόρου, πότε δηλαδή πρέπει να επικοινωνήσει ο ενδιάμεσος εξυπηρετητής με τον πηγαίο εξυπηρετητή για να ελέγχξει αν έχει τροποποιηθεί ο πόρος. Το πρόβλημα αυτό μπορεί να αντιμετωπιστεί με τη βοήθεια της θεωρίας της βέλτιστης παύσης και την εφαρμογή ενός κριτηρίου παύσης στις τιμές της συνάρτησης U.

# 5 Θεωρία Βέλτιστης Παύσης (Optimal Stopping Theory)

## 5.1 Εισαγωγή

Το βασικό πλαίσιο πολλών προβλημάτων αφορά στη λήψη μιας απόφασης έπειτα από την παρατήρηση μιας διαδικασίας που εξελίσσεται χρονικά με τυχαίο τρόπο. Ο παρατηρητής καλείται, βασισμένος μόνο σε διαδοχικά παρατηρηθείσες τυχαίες μεταβλητές, να επιτύχει το καλύτερο αποτέλεσμα. Δεδομένης, λοιπόν, της ακολουθίας των παρατηρήσεων $X_1, X_2, ..., X_n$ ο χρόνος παύσης αποτελεί τη φόρμουλα που ορίζει αν ο παρατηρητής πρέπει να σταματήσει στο βήμα *n*. Ως παραδείγματα χρόνου παύσης μπορούν αν αναφερθούν τα παρακάτω :

(1) η $5^{η}$ επίσκεψη στην κατάσταση x,

(2) 10 λεπτά μετά τη $2^{η}$ επίσκεψη στην κατάσταση y, ή

(3) η στιγμή κατά την οποία το άθροισμα $X_1 + X_2 + \cdots + X_n$ ξεπερνά το 100 κ.ο.κ.

Με την θεωρία της βέλτιστης χρονικής παύσης (optimal stopping) μελετάται το πρόβλημα της επιλογής της χρονικής στιγμής για τη λήψη μιας απόφασης, ή την εκτέλεση μιας συγκεκριμένης λειτουργίας, ώστε να μεγιστοποιηθεί το αναμενόμενο κέρδος ή να ελαχιστοποιηθεί το αναμενόμενο κόστος. Προβλήματα αυτού του είδους συναντώνται στον τομέα της στατιστικής, όπου η απόφαση μπορεί να αφορά στην εξέταση μιας υπόθεσης ή στην εκτίμηση μιας παραμέτρου, καθώς και στον τομέα της λειτουργικής έρευνας, όπου η απόφαση μπορεί να αφορά στην αντικατάσταση μιας μηχανής, την πρόσληψη μιας γραμματέως ή την επαναπαραγγελία αποθέματος κ.λ.π. [9].

Ιστορικά, οι ευρύτερες πρακτικές εφαρμογές των προβλημάτων της βέλτιστης παύσης έγιναν εμφανείς κατά τη διάρκεια του Β' Παγκοσμίου Πολέμου, κατά τον οποίο αναπτύχθηκε το πεδίο της ακολουθιακής ανάλυσης στατιστικών παρατηρήσεων, όταν η λήψη αποφάσεων σε στρατιωτικό και βιομηχανικό επίπεδο ήταν στρατηγικής σημασίας

και ενέπλεκαν έναν τεράστιο αριθμό ανθρώπων και υλικών [10]. Το πρόβλημα της βέλτιστης παύσης ανέκυψε με τη θεωρία του Wald για το κριτήριο του λόγου ακολουθιακής πιθανότητας (1945) και τα ακόλουθα βιβλία "Sequential Analysis" (1947) και "Statistical Decision Functions" (1950). Στη δεκαετία του '60, οι Chow και Robbins, το 1961 και 1963 αντίστοιχα, έδωσαν ώθηση στη ταχεία ανάπτυξη του θέματος, ενώ το βιβλίο "Great Expectations: The Theory of Optimal Stopping" από τους Chow, Robbins και Siegmund (1971) συνοψίζει αυτήν την ανάπτυξη [9].

Επιπλέον, μέσα στη δεκαετία του '70 η θεωρία της βέλτιστης παύσης αναδύθηκε ως ένα μείζον εργαλείο στην οικονομική επιστήμη, όταν οι Fischer Black και Myron Scholes ανακάλυψαν έναν πρωτοποριακό τύπο για την εκτίμηση των δικαιωμάτων προαίρεσης (stock options) που μεταμόρφωσε τις παγκόσμιες οικονομικές αγορές [10]. Η φόρμουλα Black-Scholes παραμένει το κλειδί για την τιμολόγηση των σύγχρονων μετοχών, ενώ οι έννοιες της βέλτιστης παύσης που περιλαμβάνει αποτελούν πεδίο έρευνας στην ακαδημαϊκή κοινότητα και στη βιομηχανία. Ακόμα, όμως, και τα στοιχειώδη εργαλεία της θεωρίας της βέλτιστης παύσης προσφέρουν ισχυρές, πρακτικές και πολλές φορές εντυπωσιακές λύσεις.

## 5.2 Καθορισμός του Προβλήματος Παύσης

Τα προβλήματα παύσης ορίζονται από δύο αντικείμενα [9] :

1) Μια ακολουθία τυχαίων μεταβλητών $X_1$, $X_2$, ..., των οποίων η κοινή κατανομή θεωρείται γνωστή, και

2) Μια ακολουθία συναρτήσεων ανταμοιβής με πραγματικές τιμές,

$$y_0, y_1(x_1), y_2(x_1, x_2), \ldots, y_\infty(x_1, x_2, \ldots).$$

Με βάση τα αντικείμενα αυτά, το πρόβλημα παύσης περιγράφεται ως ακολούθως [9]. Παρατηρείται η ακολουθία $X_1$, $X_2$, ... για ένα συγκεκριμένο χρονικό διάστημα. Για κάθε $n = 1, 2, \ldots$, μετά τις παρατηρήσεις $X_1 = x_1$, $X_2 = x_2$, …, $X_n = x_n$, ο παρατηρητής είτε

επιλέγει να σταματήσει και να λάβει το γνωστό αντίτιμο $y_n(x_1, ... , x_n)$, είτε επιλέγει να συνεχίσει και να λάβει την παρατήρηση $X_{n+1}$. Αν η επιλογή του είναι να μην διαλέξει καμία παρατήρηση, τότε λαμβάνει το σταθερό ποσό $y_0$. Αν δεν σταματήσει ποτέ, τότε λαμβάνει το ποσό $y_\infty(x_1, x_2, ...)$.

Το πρόβλημα αφορά στην επιλογή της κατάλληλης χρονικής στιγμής στην οποία πρέπει να σταματήσει ο παρατηρητής ώστε να μεγιστοποιηθεί το αναμενόμενο κέρδος. Επιτρέπεται η χρήση τυχαίων αποφάσεων. Δηλαδή, αν ο παρατηρητής φτάσει στο στάδιο n έχοντας παρατηρήσει τα $X_1 = x_1$, $X_2 = x_2$,…, $X_n = x_n$ μπορεί να επιλέξει να σταματήσει με μια πιθανότητα που να εξαρτάται από αυτές τις παρατηρήσεις. Η πιθανότητα αυτή δηλώνεται ως $\varphi_n (x_1, ..., x_n)$. [9]

Ένας κανόνας σταματήματος/παύσης (stopping rule) αποτελείται από την ακολουθία των συναρτήσεων $\varphi = (\varphi_0, \varphi_1(x_1), \varphi_2(x_1, x_2), ...)$ για όλα τα n και τα $x_1, x_2, ..., x_n$, με $0 \leq \varphi_n (x_1, ..., x_n) \leq 1$. Συνεπώς το $\varphi_0$ αναπαριστά την πιθανότητα κατά την οποία δεν έχει γίνει καμία παρατήρηση. Στην περίπτωση που λαμβάνεται η πρώτη παρατήρηση $X_1 = x_1$, τότε το $\varphi_1 (x_1)$ αναπαριστά την πιθανότητα να σταματήσει ο παρατηρητής μετά την πρώτη παρατήρηση, κ.ο.κ.. Με βάση τον κανόνα παύσης φ και την ακολουθία παρατηρήσεων $X = (X_1, X_2, \ldots) = (x_1, x_2, \ldots)$ καθορίζεται η τυχαία χρονική στιγμή N, με $0 < N \leq \infty$, στην οποία θα πραγματοποιηθεί η βέλτιστη παύση. [9]

Η συνάρτηση μάζας πιθανότητας του N δηλώνεται με $\psi = (\psi_0, \psi_1, \psi_2, \ldots, \psi_\infty)$, όπου $\psi_n = (x_1, x_2, \ldots, x_n) = P( N = n \mid X = x)$ για $n = 0, 1, 2, \ldots$ και

$\psi_\infty = (x_1, x_2, \ldots) = P( N = \infty \mid X = x)$. Η συσχέτισή της με τον κανόνα παύσης φ γίνεται ως εξής:

$$\psi_0 = \varphi_0$$

$$\psi_1 (x_1) = (1 - \varphi_0) * \varphi_1 (x_1)$$

.

.

$$\psi_n(x_1, x_2, \ldots, x_n) = [\prod_{1}^{n-1}(1 - \varphi_j(x_1, x_2, \ldots, x_j))] \cdot \varphi_n(x_1, x_2, \ldots, x_n)$$

.

.

$$\psi_\infty(x_1, x_2, \ldots) = 1 - \sum_{0}^{\infty} \psi_j(x_1, x_2, \ldots, x_j) \ ,$$

όπου $\psi_\infty(x_1, x_2, \ldots)$ είναι η πιθανότητα να μην γίνει παύση, αφού έχουν ληφθεί όλες οι παρατηρήσεις. [9]

Επομένως το πρόβλημα μετασχηματίζεται στην επιλογή του κανόνα παύσης φ που θα οδηγεί σε μεγιστοποίηση του κέρδους V(φ), που ορίζεται ως $V(\phi) = E_{yN}(X_1,...,X_N) = E\sum_{j=0}^{=\infty} \psi_j(X_1,...,X_j) y_j(X_1,...,X_j)$. Σαν συνάρτηση του τυχαίου χρόνου παύσης, ο κανόνας παύσης φ μπορεί να εκφραστεί ως
$\varphi_n = (X_1, X_2, \ldots, X_n) = P(N = n \mid N \geq n, X = x)$ για n = 0, 1, 2, ....

Σε κάποιες εφαρμογές, η ακολουθία ανταμοιβής περιγράφεται πιο ρεαλιστικά ως μια ακολουθία από τυχαίες μεταβλητές $Y_0, Y_1, ..., Y_\infty$ των οποίων η κοινή κατανομή με τις παρατηρήσεις $X_1, X_2, ...$ είναι γνωστή. Η πραγματική τιμή του Yn μπορεί να μην είναι γνωστή τη χρονική στιγμή n όταν πρέπει να ληφθεί η απόφαση για παύση ή συνέχιση των παρατηρήσεων.

Ωστόσο, το να επιτρέπεται να είναι τυχαίο το κέρδος δεν αποτελεί γενικότητα, καθώς, δεδομένου ότι η απόφαση για παύση τη χρονική στιγμή n μπορεί να εξαρτάται από τα $X_1, ..., X_n$, μπορεί να αντικατασταθεί η ακολουθία των τυχαίων ανταμοιβών $Y_n$ από την ακολουθία των συναρτήσεων ανταμοιβής $y_n(x_1, ..., x_n)$, με n = 0, 1,..., ∞ , όπου $y_n(x_1, ..., x_n) = E\{Y_n \mid X_1 = x_1, ..., X_n = x_n\}$. Οποιοσδήποτε κανόνας παύσης φ για την ακολουθία κέρδους (payoff) $Y_0, Y_1, ..., Y_\infty$ θα έδινε την ίδια αναμενόμενη τιμή για την ακολουθία y0, y1. . . , y∞. [9]

## 5.3 Προβλήματα Πεπερασμένου Ορίζοντα - Πρόβλημα της Γραμματέως

Μια κατηγορία προβλημάτων παύσης είναι τα προβλήματα πεπερασμένου ορίζοντα. Ένα πρόβλημα παύσης θεωρούμε ότι έχει πεπερασμένο ορίζοντα όταν υπάρχει ένα γνωστό άνω όριο ως προς τον αριθμό των σταδίων στα οποία μπορεί να σταματήσει ο παρατηρητής. Εάν λαμβάνει τις παρατηρήσεις $X_1, ..., X_T$ και πρέπει να σταματήσει μετά από την παρατήρηση $X_T$, τότε λέμε ότι το πρόβλημα έχει ορίζοντα T. Ένα πρόβλημα πεπερασμένου ορίζοντα μπορεί να ληφθεί ως μια ειδική περίπτωση του γενικού προβλήματος που παρουσιάστηκε προηγουμένως θέτοντας $y_{T+1} = \cdots = y_\infty = -\infty$.[9]

Σε γενικές γραμμές, τα προβλήματα αυτά μπορούν να λυθούν με τη μέθοδο της οπισθοδρομικής επαγωγής (backward induction). Δεδομένου ότι ο παρατηρητής πρέπει να σταματήσει στο στάδιο T, βρίσκεται αρχικά ο βέλτιστος κανόνας στο στάδιο $T-1$. Κατόπιν, γνωρίζοντας τον κανόνα αυτό, βρίσκεται ο βέλτιστος κανόνας στο προηγούμενο στάδιο $T-2$, κ.ο.κ μέχρι το αρχικό στάδιο 0.[9]

Διάφορα προβλήματα αξιολογούνται ως προβλήματα πεπερασμένου ορίζοντα, με πιο γνωστό από αυτά το πρόβλημα της γραμματέως. Το πρόβλημα της γραμματέως και οι παραλλαγές του αποτελούν μια σημαντική τάξη των προβλημάτων πεπερασμένου ορίζοντα. Το πρόβλημα αυτό αναφέρεται, συνήθως, στην επιλογή της καλύτερης γραμματέως από ένα σύνολο υποψηφίων. Πολλές φορές, όμως, αναφέρεται και στην επιλογή της καλύτερης συζύγου ή του μεγαλύτερου αριθμού από ένα σύνολο άγνωστων αριθμών (Googol).

Το κλασσικό πρόβλημα της γραμματέως (Classical Secretary Problem, CSP) μπορεί να περιγραφεί ως εξής [9] :

1. Υπάρχει μόνο μια διαθέσιμη θέση γραμματέως.

2. Υπάρχουν n αιτούντες (applicants) για την θέση, όπου το n είναι γνωστός αριθμός.

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 κάθε συνέντευξη, ο αιτών είτε προσλαμβάνεται για τη θέση, οπότε το πρόβλημα απόφασης λήγει, είτε απορρίπτεται, οπότε έρχεται η σειρά του επόμενου εξεταζόμενου.

6. Το αν θα γίνει αποδεκτός ή θα απορριφθεί ένας υποψήφιος βασίζεται μόνο στη σχετική διάταξη των υποψηφίων που έχουν περάσει από συνέντευξη μέχρι εκείνη τη στιγμή.

7. Από τη στιγμή που ο αιτών απορριφθεί δεν μπορεί αργότερα να γίνει αποδεκτός.

8. Ο στόχος είναι η επιλογή του καλύτερου αιτούντος, οπότε στην περίπτωση αυτή το κέρδος είναι 1, διαφορετικά είναι 0 .

Θεωρούνται οι παρατηρήσεις $X_1, X_2, \ldots, X_n$ όπου $X_j$ είναι η κατάταξη του j αιτούντα στην μέχρι τώρα διάταξη, με το 1 να είναι ο βέλτιστος αιτών. Σύμφωνα με την υπόθεση 4, οι τυχαίες μεταβλητές $X_j$ είναι ανεξάρτητες και έχουν ομοιόμορφη κατανομή από το 1 μέχρι το j. Δηλαδή, για 2 αιτούντες $X_1$ και $X_2$ η πιθανότητα ο $X_2$ να είναι ο $1^{ος}$ στην κατάταξη είναι ίδια με την πιθανότητα να είναι $2^{ος}$ στην κατάταξη ( $P(X_2=1) = P(X_2=2) = = 1/2$ ) κ.τ.λ.

Πρέπει να σημειωθεί ότι ένας αιτών γίνεται αποδεκτός μόνο αν είναι σχετικά καλύτερος από αυτούς που έχουν ήδη περάσει. Ένας σχετικά καλύτερος (βέλτιστος) αιτών ονομάζεται υποψήφιος (candidate). Επομένως, ο j αιτών είναι υποψήφιος αν και μόνο αν $X_j = 1$. Η πιθανότητα ο υποψήφιος στο στάδιο j να είναι η συνολικά βέλτιστη επιλογή σε σύνολο *n* αιτούντων είναι j/n. Επομένως

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

Σημειώνουμε οτι για $y_0 = 0$ και για $j \geq 1$, το $y_j$ εξαρτάται μόνο από το $x_j$.

Ένας βέλτιστος κανόνας για την επιλογή του καλύτερου υποψηφίου είναι, για κάποιο $r \geq 1$, να απορριφθούν οι πρώτοι r-1 αιτούντες και στη συνέχεια να προσληφθεί ο επόμενος σχετικά βέλτιστος υποψήφιος (αν υπάρχει). Αποδεικνύεται ότι για ένα μεγάλο αριθμό αιτούντων είναι βέλτιστο να απορριφθεί ένα ποσοστό ίσο με το $e^{-1} = 36.8$ % και να επιλεχθεί ο πρώτος υποψήφιος με τη σχετικά καλύτερη επίδοση μέχρι εκείνη τη στιγμή. Η πιθανότητα να επιλεχθεί ο καλύτερος υποψήφιος είναι τότε ίση με $e^{-1}$. [9]

## 5.4 Αλγόριθμοι Βέλτιστης Χρονικής Παύσης

### 5.4.1 Odds Algorithm

Σε ορισμένα προβλήματα βέλτιστης παύσης επιδιώκεται η μεγιστοποίηση της πιθανότητας να αναγνωριστεί ένα συγκεκριμένο γεγονός που παρατηρείται τελευταίο μέσα από μια ακολουθία διαδοχικών και ανεξάρτητων γεγονότων. Η αναγνώριση πρέπει να γίνει τη στιγμή της παρατήρησης, ενώ δεν είναι δυνατή η ανάκληση περασμένων παρατηρήσεων. Ο ορισμός του συγκεκριμένου γεγονότος γίνεται από τον παρατηρητή και αφορά στην επιλογή του κατάλληλου συμβάντος για παύση, ώστε να ληφθεί μια απόφαση. [11]

Παράδειγμα τέτοιου προβλήματος αποτελεί η πώληση ενός αυτοκινήτου στον πελάτη που θα κάνει την καλύτερη προσφορά [11]. Ο πωλητής πρέπει να ανταποκριθεί άμεσα στην προσφορά που δέχεται από τον κάθε υποψήφιο αγοραστή (n αγοραστές στο σύνολο) και να τη δεχτεί ή να την απορρίψει. Μια προσφορά ορίζεται ως ενδιαφέρουσα

και συμβολίζεται με 1, αν είναι καλύτερη από όλες τις προηγούμενες προσφορές, διαφορετικά συμβολίζεται με 0. Συνεπώς, δημιουργείται μια τυχαία ακολουθία από 0 και 1. Ο πωλητής προβληματίζεται μόνο για τις ενδιαφέρουσες προσφορές, δηλαδή για τα 1, καθώς κάθε φορά που δέχεται μια τέτοια προσφορά ενδέχεται να είναι η τελευταία. Αν, όμως, επιλέξει το τελευταίο 1 στην δυαδική ακολουθία θα έχει επιλέξει την καλύτερη προσφορά. Επομένως, ο στόχος είναι η μεγιστοποίηση της πιθανότητας να επιλεγεί το τελευταίο 1 μέσα σε μια τυχαία ακολουθία.

Η στρατηγική που ακολουθείται στο εν λόγω πρόβλημα συνίσταται στην παρατήρηση των διαδοχικών γεγονότων και στην παύση στο πρώτο ενδιαφέρον συμβάν μετά από το στάδιο s, που ορίζεται ως κατώφλι παύσης [11]. Στην περίπτωση που υπάρχουν n διαδοχικά και ανεξάρτητα μεταξύ τους γεγονότα, αντιστοιχίζεται σε κάθε συμβάν μια τιμή $I_k$, το 1 ή το 0, που δηλώνει αν η παρατήρηση είναι ενδιαφέρουσα ή μη ενδιαφέρουσα αντίστοιχα. Δημιουργείται συνεπώς μια ακολουθία $I_1, I_2, ..., I_n$ με τιμές 0 και 1. Αν $p_k$ είναι η πιθανότητα η k παρατήρηση να είναι ενδιαφέρουσα, προσδιορίζεται η πιθανότητα $q_k = 1 - p_k$ και ο λόγος $r_k = \frac{p_k}{q_k}$. Σύμφωνα με την διαδικασία του odds algorithm υπολογίζεται το άθροισμα των λόγων $r_i$ με την αντίστροφη φορά, δηλαδή $r_n + r_{n-1} + r_{n-2} + \ldots$, μέχρι τη στιγμή που το άθροισμα αυτό αποκτήσει τιμή ίση ή μεγαλύτερη του 1 για πρώτη φορά, οπότε καθορίζεται το κατώφλι παύσης s. Το παραπάνω άθροισμα παίρνει, τότε, τη μορφή $R_s = r_n + r_{n-1} + r_{n-2} + \ldots + r_s$. Υπολογίζεται επίσης το $Q_s = q_n \cdot q_{n-1} \cdot \ldots \cdot q_s$ και το γινόμενο $w = Q_s \cdot R_s$ δίνει την πιθανότητα επιτυχίας. Στην περίπτωση που το άθροισμα $R_s$ δεν παίρνει την τιμή 1, τότε το s λαμβάνεται ίσο με 1.

Με βάση την τεχνική που αναφέρθηκε μεγιστοποιείται η πιθανότητα να γίνει παύση στο τελευταίο 1 της ακολουθίας. Επίσης, προσδιορίζεται η πιθανότητα επιτυχίας από τον τύπο $w = Q_s \cdot R_s$ και παρέχεται ένα ελάχιστο όριο για αυτήν, ίσο με $1/e = 0.368$, όταν το $R_s$ είναι μεγαλύτερο από ή ίσο με 1. [11]

Μια παραλλαγή του αλγορίθμου αυτού είναι ο αλγόριθμος Odds-Algorithm με σειριακή εκτίμηση των odds (Odds-Algorithm with sequential estimation of odds). Η παραλλαγή αυτή χρησιμοποιείται στο πρόγραμμα που έχει συνταχθεί.

### 5.4.2 Odds-Algorithm με σειριακή εκτίμηση των odds

Όπως προαναφέρθηκε ο odds-algorithm επιλύει το πρόβλημα της εύρεσης του χρόνου παύσης μιας διαδικασίας ώστε να μεγιστοποιείται η πιθανότητα να σταματήσουμε στην τελευταία τιμή με $I_K = 1$ της διαδικασίας I. Η εφαρμογή του αλγορίθμου απαιτεί τη γνώση του λόγου $r_k = \frac{p_k}{1-p_k}$, όπου $p_k = E(I_k)$ με k = 1, 2, ..., n. Οι λόγοι $r_{k+1}, r_{k+2}, \ldots, r_n$ πρέπει να εκτιμώνται με βάση τις παρατηρήσεις $I_1, I_2, ..., I_\kappa$. Το πρόβλημα είναι ότι σε πολλές εφαρμογές της θεωρίας της βέλτιστης παύσης το $p_k$, που εκφράζει την πιθανότητα η k παρατήρηση να είναι «ενδιαφέρουσα», δηλαδή να ισχύει $I_\kappa = 1$, είναι άγνωστο και πρέπει να εκτιμηθεί σειριακά. Η αντιμετώπιση των προβλημάτων αυτού του είδους αποτελεί αντικείμενο του [12]. Οι συγγραφείς του [12] καταλήγουν σε μια νέα εκδοχή για τον odds-algorithm, εξετάζουν το θέμα από τη σκοπιά της ταχύτητας και της επίδοσης και προτείνουν τη χρήση του νέου αλγορίθμου στα προβλήματα αυτής της φύσεως.

Σύμφωνα με το μοντέλο που προτείνεται στο [12], η πιθανότητα $p_k$, και συνεπώς ο λόγος $r_k$, θεωρείται ότι είναι συνάρτηση μίας μόνο άγνωστης σταθερής παραμέτρου p, δηλαδή ισχύει $p_k = p*f_k$, με το $f_k$ να είναι γνωστό. Ο προσδιορισμός των «μελλοντικών» τιμών των $p_k$, $q_k$ και $r_k$ προκύπτουν με βάση προγενέστερες παρατηρήσεις. Αν δεχτούμε ότι το $I_\kappa$ παριστάνει την μετρούμενη τιμή μιας διαδικασίας κατά τη χρονική στιγμή k και ότι για κάθε επιτυχή παρατήρηση είναι $I_\kappa(p) = 1$, τότε μια εκτίμηση για την τιμή της

παραμέτρου p στη χρονική στιγμή s προκύπτει από τον τύπο $\hat{p}(s,p) = \frac{\sum_{1}^{s} I_k(p)}{\sum_{1}^{s} f_k}$, όπου $\sum_{1}^{s} I_k(p)$ είναι ο αριθμός των επιτυχημένων ή «ενδιαφέροντων» παρατηρήσεων μέχρι την χρονική στιγμή s.

Σε περίπτωση που αρχικά δεν υπάρχουν επιτυχείς παρατηρήσεις, η τιμή του $\hat{p}$ είναι μικρή και ενδεχομένως η παύση να γίνει νωρίς. Για την αποφυγή του προβλήματος αυτού είναι δυνατό να παραληφθεί ένας αριθμός τιμών προτού εφαρμοστεί ο αλγόριθμος, δηλαδή η εφαρμογή να ξεκινήσει από το βήμα $s = s_d$. Όταν το $s_d$ είναι 1 προφανώς δεν έχει εισαχθεί κάποια καθυστέρηση.

Για την ακολουθία $f_k$ στο [12] γίνονται δύο επιλογές. Στη περίπτωση ανεξάρτητων και όμοια κατανεμημένων τυχαίων παρατηρήσεων (independent and identically distributed – i.i.d.) επιλέγεται $f_k = 1$ για όλες τις τιμές του k. Στην άλλη περίπτωση, όπου κάθε παρατήρηση θεωρείται ισοπίθανα καλύτερη, δεύτερη καλύτερη κ.λ.π., χρησιμοποιείται $f_k = 1/k$, με αποτέλεσμα να προκύπτουν διαφορετικοί λόγοι $r_k$ για κάθε περίπτωση. Επιπλέον, η επιλογή αυτή σύμφωνα με το [12], δεν εξυπηρετεί κάποιο ιδιαίτερο σκοπό, παρά το ότι επιλύει μια νέα εκδοχή του προβλήματος της γραμματέως όπου ο υποψήφιος είναι διαθέσιμος με άγνωστη πιθανότητα. Ειδικότερα, αν σε μια ακολουθία υποψηφίων όλοι είναι το ίδιο πιθανό να είναι στην κατάταξη πρώτοι, δεύτεροι κ.λ.π. και ο k υποψήφιος είναι διαθέσιμος, ανεξαρτήτως κατάταξης, με πιθανότητα p, τότε ο υποψήφιος αυτός είναι η καλύτερη και ταυτόχρονα διαθέσιμη επιλογή με πιθανότητα p/k.

Όσον αφορά το σύνολο των τιμών n που λαμβάνονται υπόψη στον αλγόριθμο προτείνεται το μέγεθος n του δείγματος να είναι 15. Εξάλλου, όπως αναφέρεται, σε πολλές σημαντικές εφαρμογές το n δεν παίρνει πολύ μεγάλη τιμή, αλλά κυμαίνεται στο

10 με 15. Ωστόσο, για μικρές τιμές του n ($\leq$ 6) οι εκτιμήσεις των $p_k$, $q_k$ και $r_k$ είναι αναξιόπιστες.

Ο αλγόριθμος Odds-Algorithm με σειριακή εκτίμηση των odds παρουσιάζεται παρακάτω.

---

**Αλγόριθμος : Odds-algorithm με διαδοχική εκτίμηση των $p_k$, $q_k$ και $r_k$**

---

**Είσοδος :** προϋπολογισμός καθυστέρησης $s_d$ (εφόσον χρησιμοποιείται)
**Έξοδος :** το βήμα s για τη βέλτιστη παύση

```
s := sd
cont := true
while cont do
      ν := Σ_1^s I_k ,   p̂(s) = ν / Σ_1^s f_k
      if Σ_{s+1}^n r_k (p̂(s)) < 1  then
            cont := false
      else
            s := s +1
            if s = n then
                  cont := false
            end if
      end if
end while
return s
```

---

# 6 Προσομοιώσεις - Αποτελέσματα

Το πρόβλημα της αποθήκευσης αντικειμένων σε έναν caching server και του διαμοιρασμού έγκυρης πληροφορίας στους clients μπορεί να θεωρηθεί ως ένα πρόβλημα βέλτιστης παύσης, δηλαδή πρόβλημα προσδιορισμού της βέλτιστης χρονικής στιγμής στην οποία ο caching server πρέπει να ανανεώσει τον αποθηκευμένο πόρο. Για το λόγο αυτό, η αντιμετώπιση του προβλήματος της cache συνέπειας γίνεται με βάση το μοντέλο που προτάθηκε στο κεφάλαιο 4 και με τη χρήση ενός αλγορίθμου που εφαρμόζεται σε προβλήματα βέλτιστης παύσης και ειδικότερα τον αλγόριθμο Odds με διαδοχική εκτίμηση (sequential estimation) των λόγων (odds) $r_k$, όπως αναφέρθηκε στο κεφάλαιο 5.

Το σύστημα που υλοποιήθηκε αποτελείται από ένα ενδιάμεσο εξυπηρετητή (caching server) που δέχεται αιτήσεις για αντικείμενα από διάφορους προορισμούς (sites). Τα αντικείμενα αυτά τροποποιούνται από μια πηγή, με αποτέλεσμα να υπάρχει πιθανότητα μετάδοσης μη έγκυρης πληροφορίας από τον εξυπηρετητή.

Στο κεφάλαιο αυτό υλοποιείται ο μηχανισμός ασθενούς συνέπειας που κάνει χρήση του adaptive TTL, στη συνέχεια εφαρμόζεται ο μηχανισμός ασθενούς συνέπειας που αξιοποιεί τον αλγόριθμο Odds που βασίζεται σε διαδοχική εκτίμηση (sequential estimation) και παρουσιάζονται τα αποτελέσματα.

## 6.1 Caching Server - Αρχείο Αιτήσεων

Ο caching server δέχεται ανά χρονικά διαστήματα αιτήσεις προς εξυπηρέτηση. Οι αιτήσεις αυτές αφορούν σε ένα συγκεκριμένο αντικείμενο που προέρχεται από κάποιο site. Αρχικά, ο εξυπηρετητής είναι κενός και κάθε αίτηση για νέο αντικείμενο προϋποθέτει ότι ο ίδιος «φορτώνει» το αντικείμενο αυτό. Αιτήσεις που αφορούν αντικείμενα που ήδη υπάρχουν στον διακομιστή εξυπηρετούνται από αυτόν, με

αποτέλεσμα να έχουμε «επιτυχία» (hit). Ωστόσο, ο ενδιάμεσος εξυπηρετητής είναι περιορισμένης χωρητικότητας και όταν φτάσει η πληρότητά του ένα συγκεκριμένο όριο εφαρμόζεται κατάλληλος μηχανισμός για την εξασφάλιση περισσότερου χώρου.

Το ιστορικό των αιτήσεων που καταφθάνουν στον ενδιάμεσο εξυπηρετητή καταγράφονται σε ένα αρχείο. Σε κάθε εγγραφή του αρχείου αυτού περιλαμβάνεται πληροφορία για το αντικείμενο, τον ιστότοπο προέλευσης και το χρόνο αποθήκευσης στο διακομιστή, αν πρόκειται για νέο αντικείμενο, ή το χρόνο εξυπηρέτησης αν πρόκειται για αντικείμενο που ήδη υπάρχει στην cache.

Ο ιστότοπος στον οποίο αναφέρεται η κάθε αίτηση παράγεται από μια γεννήτρια ομοιόμορφα κατανεμημένων τυχαίων αριθμών, ωστόσο τα αντικείμενα του site που αναζητώνται πρέπει να ακολουθούν την κατανομή Zipf. Ο νόμος Zipf καθορίζει την πιθανότητα P(i) πρόσβασης στο i-ιοστό στοιχείο μίας ταξινομένης, ως προς την συχνότητα προσπέλασης, cache memory ή ενός WWW site. Η συνάρτηση πυκνότητας πιθανότητας P(i) προσδιορίζει το ποσοστό των αιτήσεων που αφορούν το στοιχείο i μετά την ταξινόμηση των στοιχείων ως προς την δημοτικότητα τους (popularity ranking). Στη γενική μορφή ισχύει:

$$P(i) = ki^{-Z}$$

Η σταθερά k εξασφαλίζει ότι το άθροισμα των πιθανοτήτων, για όλα τα θεωρούμενα στοιχεία-πόρους, είναι 1. Η σταθερά Z είναι η σταθερά Zipf η οποία για τις περιπτώσεις των WWW caches έχει προσδιοριστεί στην περιοχή 0.8 – 1.

Από το νόμο του Zipf προκύπτει ότι $i = (\frac{k}{P(i)})^{\frac{1}{Z}}$, συνεπώς λαμβάνοντας ως Z το 0.8 και αφού πρώτα υπολογιστεί η τιμή του k που αντιστοιχεί σε κάθε site, μπορούμε να προσδιορίσουμε το αντικείμενο από κάθε site για το οποίο γίνεται μια αίτηση.

Όσον αφορά στο χρόνο στον οποίο γίνεται μια αίτηση μπορούμε να πούμε τα εξής. Οι χρόνοι μεταξύ δυο διαδοχικών αιτήσεων που καταφθάνουν στον caching server ακολουθούν την κατανομή Pareto. Σύμφωνα με τον ορισμό της κατανομής Pareto, αν X

είναι μια τυχαία μεταβλητή που ακολουθεί την εν λόγω κατανομή, τότε η πιθανότητα η τιμή της X να είναι μεγαλύτερη από ένα αριθμό x είναι

$$P_r(X > x) = \begin{cases} (\frac{x_m}{x})^a, x \geq x_m \\ 1 \qquad , x \leq x_m \end{cases}$$

, όπου xm είναι η μικρότερη τιμή της X και a μια θετική παράμετρος.

Επομένως, ο χρόνος μεταξύ των διαδοχικών αφίξεων δίνεται από τον τύπο $\text{time} = \frac{t_{min}}{P_r^{\frac{1}{a}}}$, όπου $t_{min}$ είναι η ελάχιστη τιμή που μπορεί να πάρει ο χρόνος μεταξύ διαδοχικών αιτήσεων. Με επιλογή των τιμών 9 και 1.2 για τις παραμέτρους $t_{min}$ και a αντίστοιχα προσδιορίζονται οι χρόνοι αφίξεως των αιτήσεων στον ενδιάμεσο εξυπηρετητή.

Οι αιτήσεις προς τον caching server περιλαμβάνονται στο αρχείο **caching_server_traces.txt**, που χρησιμοποιείται στις προσομοιώσεις που περιγράφονται παρακάτω. Παρείχθησαν συνολικά 10000 αιτήσεις με μια μέση χρονική απόσταση μεταξύ διαδοχικών αιτήσεων ίση με 38.57 λεπτά.

## 6.2 Γεννήτρια Αντικειμένων - Αρχείο Τροποποιήσεων

Τα αντικείμενα που καταγράφονται στο αρχείο των αιτήσεων προς τον ενδιάμεσο εξυπηρετητή, τροποποιούνται ανά χρονικά διαστήματα από μια πηγή αντικειμένων. Το ιστορικό των αλλαγών αυτών περιέχεται σε κατάλληλο αρχείο, στο οποίο καταγράφεται πληροφορία για το αντικείμενο που μεταβάλλεται, τον ιστότοπο προέλευσης του και τη χρονική στιγμή πραγματοποίησης της αλλαγής.

Από γεννήτρια ομοιόμορφα κατανεμημένων τυχαίων αριθμών προκύπτει το site και το αντικείμενο στο οποίο αναφέρεται η τροποποίηση. Χρησιμοποιήθηκαν 7 sites με διαφορετικό πλήθος αντικειμένων, όπως φαίνεται παρακάτω.

| Πλήθος αντικειμένων | 200 | 300 | 400 | 500 | 650 | 800 | 200 |
|---|---|---|---|---|---|---|---|
| Site | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Οι χρονικές αφίξεις των τροποποιήσεων στην πηγή υπακούουν στην εκθετική κατανομή, που είναι της μορφής $f(x) = e^{-\lambda \cdot x}$. Συνεπώς, ο χρόνος μεταξύ διαδοχικών τροποποιήσεων ακολουθεί την αντίστροφη συνάρτηση της εκθετικής, σύμφωνα με τον τύπο $\text{time} = \frac{\ln P(i)}{\lambda}$, όπου λ είναι ο ρυθμός αφίξεων και θεωρήθηκε ίσος με 0.0026 αιτήσεις ανά λεπτό.

Για την περίπτωση της υλοποίησης του adaptive TTL, που περιγράφεται στην επόμενη υποενότητα, γίνεται χρήση του αρχείου τροποποιήσεων των αντικειμένων **source_traces_attl.txt**, που χρησιμοποιείται στις προσομοιώσεις που περιγράφονται παρακάτω. Η μέση χρονική απόσταση μεταξύ διαδοχικών τροποποιήσεων είναι 391 λεπτά για το ίδιο χρονικό διάστημα για το οποίο παράγονται αιτήσεις προς τον caching server. Η τιμή αυτή είναι σχεδόν 10πλάσια από την μέση χρονική απόσταση μεταξύ διαδοχικών αιτήσεων (38.57 λεπτά) στον ενδιάμεσο εξυπηρετητή. Αυτό σημαίνει ότι για κάθε τροποποίηση αντικειμένου στην πηγή έχουν ήδη γίνει 10 αιτήσεις στον εξυπηρετητή, προσεγγίζοντας με αυτό τον τρόπο ρεαλιστικά δεδομένα.

## 6.3 Προσομοίωση μηχανισμού ασθενούς συνέπειας Adaptive TTL

Σύμφωνα με το μοντέλο που εφαρμόζεται στην περίπτωση αυτή, κάθε φορά που ο caching server παραλαμβάνει μια αίτηση για ένα αντικείμενο που δεν είναι αποθηκευμένο σε αυτόν, αποδίδει σε αυτό ένα χρονικό διάστημα για το οποίο θεωρεί ότι

η πληροφορία που διαθέτει είναι έγκυρη. Το χρονικό διάστημα υπολογίζεται με βάση το adaptive TTL και προσδιορίζεται από τον τύπο

$$k \cdot (load\_time - last\_modified),$$

όπου *load_time* είναι η χρονική στιγμή αποθήκευσης του αντικειμένου στον εξυπηρετητή (περιέχεται στο αρχείο των αιτήσεων), *last_modified* είναι η χρονική στιγμή της τελευταίας τροποποίησης του αντικειμένου (περιέχεται στο αρχείο των τροποποιήσεων από την πηγή) και *k* είναι μια σταθερά ίση με 0.2.

Γίνεται προφανές πως για τον προσδιορισμό του χρόνου ζωής είναι απαραίτητο να προηγείται της αίτησης για ένα συγκεκριμένο αντικείμενο μια τουλάχιστον τροποποίηση. Για το λόγο αυτό, θεωρήθηκε ότι η παραλαβή των αιτήσεων από τον caching server ξεκινάει τη χρονική στιγμή 1200000 και τερματίζει τη χρονική στιγμή 1585680 (για το 10000$^{ο}$ αντικείμενο). Υπάρχει δηλαδή ένα χρονικό διάστημα 1200000 λεπτών, 3πλάσιο περίπου του χρονικού διαστήματος για το οποίο ο caching server παραλαμβάνει αιτήσεις, κατά το οποίο τα αντικείμενα τροποποιούνται από την πηγή. Δίνεται έτσι η δυνατότητα να υπολογιστεί ο χρόνος ζωής για κάθε αντικείμενο για το οποίο φτάνει μια αίτηση στον εξυπηρετητή μέσω του παραπάνω τύπου.

Όσον αφορά στη χρησιμότητα του χρόνου ζωής, κάθε φορά που γίνεται επιτυχημένη αναζήτηση ενός αντικειμένου στο διακομιστή χωρίς να έχει λήξει ο χρόνος ζωής του, καταγράφεται η χρονική στιγμή της τελευταίας αναφοράς (*last_access*) και ελέγχεται αν η διαθέσιμη πληροφορία είναι μη έγκυρη (stale hit), δηλαδή αν το αντικείμενο έχει ήδη μεταβληθεί στην πηγή από τη χρονική στιγμή της τελευταίας τροποποίησής του, *last_modified,* μέχρι τη χρονική στιγμή της αναφοράς του στο αρχείο των αιτήσεων, *last_access*. Στην περίπτωση που ο χρόνος ζωής του αντικειμένου έχει εκπνεύσει όταν φθάνει η αίτηση στον caching server, ελέγχεται αν το αντικείμενο έχει τροποποιηθεί στην πηγή από τη χρονική στιγμή της τελευταίας τροποποίησής του και αναπροσαρμόζεται κατάλληλα ο χρόνος ζωής σύμφωνα με τον παραπάνω τύπο.

Ένα άλλο ζήτημα που λαμβάνεται υπόψη αφορά στη χωρητικότητα του caching server. Η χωρητικότητα του ενδιάμεσου εξυπηρετητή θεωρείται πεπερασμένη. Για την περίπτωση που εξετάζεται, το μέγεθος του διακομιστή λαμβάνεται αρχικά ίσο με 400, ενώ χρησιμοποιούνται και οι τιμές 300, 250, 200, 150 και 100. Θεωρείται επίσης ότι για κάθε αίτηση προς το διακομιστή που αφορά ένα καινούριο αντικείμενο, αποδίδεται σε αυτό μέγεθος ίσο με τη μονάδα. Για την εξασφάλιση χώρου όταν ο caching server είναι πλήρης, εφαρμόζεται η τεχνική LRU (Least Recently Used), κατά την οποία πρώτα διαγράφονται τα αντικείμενα για τα οποία γίνεται πιο σπάνια αναφορά. Πιο συγκεκριμένα, όταν η πληρότητα του caching server φτάσει το 80% εφαρμόζεται ο αλγόριθμος αντικατάστασης LRU μέχρις ότου να μειωθεί η πληρότητα στο 60%.

Οι πληροφορίες που ενδιαφέρουν για την αξιολόγηση των αποτελεσμάτων είναι ο συνολικός αριθμός των επιτυχέντων αναζητήσεων (hits) και ο αριθμός των επιτυχέντων αναζητήσεων για τις οποίες η πληροφορία του εξυπηρετητή δεν είναι έγκυρη (stale hits). Επιπλέον, παρακολουθείται ο αριθμός των ελέγχων που πραγματοποιούνται μετά την εκπνοή του χρόνου ζωής των αντικειμένων για να διαπιστωθεί αν παραμένουν έγκυρα ή αν έχουν τροποποιηθεί πριν την αναπροσαρμογή του χρόνου ζωής. Λαμβάνοντας υπόψη τα όσα προαναφέρθηκαν, η προσομοίωση δίνει τα παρακάτω αποτελέσματα.

| **Μέγεθος caching server** | 400 | 300 | 250 | 200 | 150 | 100 |
|---|---|---|---|---|---|---|
| **Σύνολο Αντικειμένων** | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 |
| **Σύνολο Hits** | 9548 | 9479 | 9410 | 9293 | 9078 | 8666 |
| **Stale Hits** | 509 | 505 | 499 | 480 | 411 | 343 |
| **Ποσοστό stale επί των hits** | 5.33% | 5.328% | 5.303% | 5.165% | 4.527% | 3.958% |
| **Ποσοστό stale επί των αντικειμένων** | 5.09% | 5.05% | 4.99% | 4.8% | 4.11% | 3.43% |
| **Αντικείμενα που ελέγχθηκαν μετά την λήξη του TTL και βρέθηκε ότι άλλαξαν όσο ίσχυε το TTL** | 39 | 30 | 25 | 20 | 11 | 3 |
| **Αντικείμενα που ελέγχθηκαν μετά την λήξη του TTL και βρέθηκε ότι δεν άλλαξαν όσο ίσχυε το TTL** | 428 | 377 | 341 | 298 | 243 | 169 |
| **Συνολικός αριθμός ελέγχων** | 467 | 407 | 366 | 318 | 254 | 172 |

Πίνακας 1 : Αποτελέσματα adaptive TTL ως συνάρτηση της χωρητικότητας του εξυπηρετητή.

## 6.4 Προσομοίωση Odds Algorithm with Sequential Estimation

Στο μοντέλο που εφαρμόζεται στην περίπτωση αυτή, όπως και προηγουμένως, ο caching server καταχωρεί κάθε νέο αντικείμενο για το οποίο δέχεται αίτηση. Ειδικότερα, στο αρχείο των τροποποιήσεων που περιγράφηκε στην παράγραφο 6.2, περιλαμβάνονται 2 επιπλέον στήλες που περιέχουν μια εκτίμηση για τη χρονική στιγμή και το χρονικό διάστημα μέχρι τα οποία θεωρείται έγκυρο ένα αντικείμενο. Η εκτίμηση αυτή στηρίζεται στο ότι ο χρόνος ζωής $ttl$ για τα αντικείμενα θεωρείται 200 φορές μεγαλύτερος από το χρόνο μεταξύ διαδοχικών τροποποιήσεων στο αρχείο της πηγής. Με βάση αυτή την παραδοχή το χρονικό διάστημα για το οποίο η πληροφορία θεωρείται έγκυρη εκτιμάται για κάθε αντικείμενο ως ένα πολλαπλάσιο του χρόνου ζωής, δηλαδή $k \cdot ttl$, με το k να

παίρνει ομοιόμορφα κατανεμημένες τυχαίες τιμές από 1 έως 2. Στη συνέχεια, χρησιμοποιώντας την τιμή που προέκυψε, προσδιορίζεται και η χρονική στιγμή μέχρι την οποία το αντικείμενο θεωρείται έγκυρο. Το αρχείο τροποποιήσεων των αντικειμένων που χρησιμοποιείται στην περίπτωση αυτή είναι το **source_traces_odds.txt**.

Η εκτίμηση των χρόνων που προαναφέρθηκαν είναι απαραίτητη για τον υπολογισμό του μεγέθους U, στις τιμές του οποίου εφαρμόζεται σε τακτά χρονικά διαστήματα ο αλγόριθμος Odds με διαδοχική εκτίμηση. Με τον τρόπο αυτό προσδιορίζεται το χρονικό σημείο στο οποίο πρέπει να ελεγχθεί αν το αντικείμενο έχει τροποποιηθεί, επικοινωνώντας με την πηγή.

Παράλληλα, σε κάθε επιτυχή αναζήτηση ενός αντικειμένου ελέγχεται η εγκυρότητα της πληροφορίας που διέθεσε ο ενδιάμεσος εξυπηρετητής.

### 6.4.1 Προσδιορισμός της Συνάρτησης U

Σύμφωνα με τον αλγόριθμο Odds παρακολουθούνται οι τιμές που λαμβάνει μια συνάρτηση και με βάση προγενέστερες τιμές της γίνεται η εκτίμηση των odds. Στην περίπτωσή μας η συνάρτηση U, που παρουσιάστηκε στο $4^{ο}$ κεφάλαιο, παίζει το ρόλο της συνάρτησης αυτής. Οι τιμές της υπολογίζονται σε συγκεκριμένες χρονικές στιγμές για κάθε αντικείμενο που βρίσκεται στον caching server. Για τον υπολογισμό της U προσδιορίζονται τα παρακάτω μεγέθη :

- οι επιτυχείς αναζητήσεις και ο αριθμός των αιτήσεων που αφορούν τον ιστότοπο στον οποίο ανήκει το κάθε αντικείμενο,
- ο συνολικός αριθμός των αιτήσεων που φθάνουν στον caching server,
- η χρονική στιγμή στην οποία αποθηκεύτηκε το αντικείμενο στον caching server (load_time), που λαμβάνεται ίση με το χρόνο άφιξης της αίτησης για ένα νέο αντικείμενο,

- η χρονική στιγμή (expires_time) μέχρι την οποία το αντικείμενο θεωρείται έγκυρο, μια εκτίμηση της οποίας αντλείται από το αρχείο των τροποποιήσεων της πηγής,
- το μέσο χρονικό διάστημα για το οποίο θεωρούνται έγκυρα τα αντικείμενα από τον ίδιο ιστότοπο (average_expiration_time), που στηρίζεται στην εκτίμηση του expires_time και
- η τρέχουσα χρονική στιγμή (current_time), που είναι η στιγμή κατά την οποία υπολογίζεται η συνάρτηση U για όλα τα αντικείμενα.

Αξίζει να σημειωθεί ότι μετά τη λήξη του χρονικού διαστήματος expires_time, για το οποίο θεωρείται έγκυρη η πληροφορία του caching server, οι παράγοντες $\rho$ και $\xi$ της συνάρτησης U αποκτούν τιμές μεγαλύτερες του 1 και για το λόγο αυτό λαμβάνονται ίσοι με τη μονάδα, μιας και αυτή είναι η μέγιστη συνεισφορά για τον κάθε παράγοντα. Εξάλλου, μετά τη λήξη του συγκεκριμένου χρονικού διαστήματος η ανάγκη για έλεγχο της εγκυρότητας του αντικειμένου γίνεται άκρως επιτακτική.

Πιο συγκεκριμένα, κάθε φορά που ο caching server λαμβάνει μια αίτηση για ένα νέο αντικείμενο, καταχωρεί το αντικείμενο, τον ιστότοπο στον οποίο ανήκει, το χρόνο άφιξης της αίτησης, τη χρονική στιγμή και το χρονικό διάστημα μέχρι τα οποία θεωρείται έγκυρο το αντικείμενο. Επιπλέον, κάθε φορά που γίνεται hit καταγράφεται η χρονική στιγμή της τελευταίας αναφοράς (*last_access*) και ελέγχεται αν η πληροφορία που διαθέτει ο εξυπηρετητής είναι έγκυρη ή αν στο μεταξύ έχει αλλάξει, οπότε μιλάμε για ένα *stale hit*. Σε τακτά χρονικά διαστήματα γίνεται υπολογισμός της συνάρτησης U, με βάση τα δεδομένα που αναφέρθηκαν παραπάνω, και αφού συμπληρωθεί το απαραίτητο πλήθος τιμών της U για κάθε αντικείμενο, εφαρμόζεται ο αλγόριθμος Odds. Όταν ο αλγόριθμος αποφασίσει ότι πρέπει να ενημερωθεί το αντικείμενο, δηλαδή όταν το άθροισμα των λόγων $r_k$ γίνει μικρότερο της μονάδας, ελέγχεται αν το αντικείμενο έχει ήδη τροποποιηθεί στην πηγή οπότε προχωρά στην ενημέρωση του αντιγράφου του caching server.

### 6.4.2 Εφαρμογή Αλγορίθμου Odds

Όπως προαναφέρθηκε, ο αλγόριθμος Odds με διαδοχική εκτίμηση βασίζεται στον αριθμό των επιτυχημένων παρατηρήσεων μιας διαδικασίας. Επίσης, ο ορισμός της επιτυχίας εξαρτάται από τον ίδιο τον παρατηρητή και έχει να κάνει με το πρόβλημα που αντιμετωπίζει. Στη συγκεκριμένη περίπτωση η παρατήρηση θεωρείται επιτυχής όταν η τιμή της συνάρτησης U ξεπεράσει μια συγκεκριμένη τιμή.

Ειδικότερα, για την εφαρμογή του αλγορίθμου Odds με διαδοχική εκτίμηση, υπεισέρχονται 3 παράμετροι :

- η τιμή της συνάρτησης U για την οποία η παρατήρηση θεωρείται επιτυχής,
- το πλήθος των τιμών της συνάρτησης U για κάθε αντικείμενο για τις οποίες εφαρμόζεται ο αλγόριθμος και
- η χρονική στιγμή εφαρμογής του αλγορίθμου.

Κατά την εκτέλεση του αλγορίθμου, οι τιμές της συνάρτησης U που είναι μεγαλύτερες από ένα κατώφλι $I_k$ θεωρούνται ως επιτυχημένες παρατηρήσεις. Ως τιμές κατωφλίου ελήφθησαν οι **0.8**, **0.75**, **0.7**, **0.6**, **0.5** και **0.25**. Επιπλέον, χρησιμοποιήθηκαν 3 διαφορετικά πλήθη τιμών της U πάνω στα οποία γίνεται η εφαρμογή του αλγορίθμου. Αυτά είναι **15**, **10** και **7**. Όσον αφορά το χρόνο εκτέλεσης του αλγορίθμου χρησιμοποιήθηκαν 3 χρονικά διαστήματα **1000**, **3000** και **6000** λεπτών.

Οι πληροφορίες που ενδιαφέρουν για την αξιολόγηση των προσομοιώσεων είναι ο αριθμός των μη έγκυρων επιτυχέντων αναζητήσεων (**stale hits**), ο αριθμός των αντικειμένων που μετά την εκτέλεση του αλγορίθμου και τον έλεγχο στην πηγή (όταν το άθροισμα των λόγων $r_k$ γίνει μικρότερο του 1) βρέθηκαν τροποποιημένα (**updated**), ο αριθμός των αντικειμένων που μετά την εκτέλεση του αλγορίθμου και τον έλεγχο στην πηγή (όταν το άθροισμα των λόγων $r_k$ γίνει μικρότερο του 1) βρέθηκε ότι δεν έχουν αλλάξει (**not changed**) και ο συνολικός αριθμός των αντικειμένων για τα οποία

εκτελέστηκε ο αλγόριθμος (**Odds runs**). Επίσης, μετράται ο συνολικός αριθμός των αντικειμένων για τα οποία υπολογίστηκε η συνάρτηση U (**U calc**).

Τέλος, το μέγεθος του caching server λαμβάνεται ίσο με 400. Στην περίπτωση αυτή, όπως βρέθηκε και στην προσομοίωση του adaptive TTL, ο αριθμός των επιτυχημένων αναζητήσεων είναι 9548.

Σύμφωνα με όσα έχουν προαναφερθεί εφαρμόζεται το μοντέλο και τα αποτελέσματα παρουσιάζονται στους παρακάτω πίνακες.

| $I_k = 0.8$ | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Χρονικό διάστημα** | 6000 | | | **Χρονικό διάστημα** | 3000 | | | **Χρονικό διάστημα** | 1000 | | |
| **Πλήθος** | 15 | 10 | 7 | **Πλήθος** | 15 | 10 | 7 | **Πλήθος** | 15 | 10 | 7 |
| **stale hits** | 1750 | 1750 | 1750 | **stale hits** | 1750 | 1750 | 1750 | **stale hits** | 1750 | 1750 | 1750 |
| **Odds sum<1 Updated** | 0 | 0 | 0 | **Odds sum<1 Updated** | 0 | 0 | 0 | **Odds sum<1 Updated** | 0 | 0 | 0 |
| **Odds sum<1 not changed** | 0 | 0 | 0 | **Odds sum<1 not changed** | 0 | 0 | 0 | **Odds sum<1 not changed** | 0 | 0 | 0 |
| **Odds runs** | 880 | 1409 | 2098 | **Odds runs** | 1920 | 2961 | 4354 | **Odds runs** | 6138 | 9312 | 13402 |
| **U calc** | 15964 | 15964 | 15964 | **U calc** | 31807 | 31807 | 31807 | **U calc** | 95218 | 95218 | 95218 |

Πίνακας 2 : Αποτελέσματα μοντέλου με τιμή κατωφλίου $I_k = 0.8$ για τις επιτυχείς παρατηρήσεις που καταγράφει ο αλγόριθμος Odds.

| $I_k = 0.75$ | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Χρονικό διάστημα** | 6000 | | | **Χρονικό διάστημα** | 3000 | | | **Χρονικό διάστημα** | 1000 | | |
| **Πλήθος** | 15 | 10 | 7 | **Πλήθος** | 15 | 10 | 7 | **Πλήθος** | 15 | 10 | 7 |
| **stale hits** | 950 | 825 | 689 | **stale hits** | 696 | 610 | 610 | **stale hits** | 573 | 553 | 557 |
| **Odds sum<1 Updated** | 42 | 49 | 49 | **Odds sum<1 Updated** | 50 | 49 | 51 | **Odds sum<1 Updated** | 52 | 53 | 54 |
| **Odds sum<1 not changed** | 317 | 617 | 833 | **Odds sum<1 not changed** | 734 | 1143 | 1553 | **Odds sum<1 not changed** | 2248 | 3401 | 4847 |
| **Odds runs** | 880 | 1409 | 2098 | **Odds runs** | 1920 | 2961 | 4354 | **Odds runs** | 6138 | 9312 | 13402 |
| **U calc** | 15964 | 15964 | 15964 | **U calc** | 31807 | 31807 | 31807 | **U calc** | 95218 | 95218 | 95218 |

Πίνακας 3 : Αποτελέσματα μοντέλου με τιμή κατωφλίου $I_k = 0.75$ για τις επιτυχείς παρατηρήσεις που καταγράφει ο αλγόριθμος Odds.

| $I_k = 0.7$ | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Χρονικό διάστημα** | 6000 | | | **Χρονικό διάστημα** | 3000 | | | **Χρονικό διάστημα** | 1000 | | |
| **Πλήθος** | 15 | 10 | 7 | **Πλήθος** | 15 | 10 | 7 | **Πλήθος** | 15 | 10 | 7 |
| **stale hits** | 949 | 681 | 686 | **stale hits** | 678 | 529 | 487 | **stale hits** | 481 | 467 | 452 |
| **Odds sum<1 Updated** | 43 | 49 | 49 | **Odds sum<1 Updated** | 50 | 49 | 51 | **Odds sum<1 Updated** | 52 | 53 | 54 |
| **Odds sum<1 not changed** | 379 | 660 | 861 | **Odds sum<1 not changed** | 939 | 1375 | 1880 | **Odds sum<1 not changed** | 2743 | 4068 | 5849 |
| **Odds runs** | 880 | 1409 | 2098 | **Odds runs** | 1920 | 2961 | 4354 | **Odds runs** | 6138 | 9312 | 13402 |
| **U calc** | 15964 | 15964 | 15964 | **U calc** | 31807 | 31807 | 31807 | **U calc** | 95218 | 95218 | 95218 |

Πίνακας 4 : Αποτελέσματα μοντέλου με τιμή κατωφλίου $I_k = 0.7$ για τις επιτυχείς παρατηρήσεις που καταγράφει ο αλγόριθμος Odds.

| $I_k = 0.6$ | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Χρονικό διάστημα** | 6000 | | | **Χρονικό διάστημα** | 3000 | | | **Χρονικό διάστημα** | 1000 | | |
| **Πλήθος** | 15 | 10 | 7 | **Πλήθος** | 15 | 10 | 7 | **Πλήθος** | 15 | 10 | 7 |
| **stale hits** | 547 | 348 | 355 | **stale hits** | 320 | 137 | 116 | **stale hits** | 61 | 66 | 57 |
| **Odds sum<1 Updated** | 62 | 72 | 70 | **Odds sum<1 Updated** | 73 | 73 | 74 | **Odds sum<1 Updated** | 79 | 80 | 81 |
| **Odds sum<1 not changed** | 704 | 1192 | 1499 | **Odds sum<1 not changed** | 1549 | 2253 | 3283 | **Odds sum<1 not changed** | 4700 | 7039 | 10023 |
| **Odds runs** | 880 | 1409 | 2098 | **Odds runs** | 1920 | 2961 | 4354 | **Odds runs** | 6138 | 9312 | 13402 |
| **U calc** | 15964 | 15964 | 15964 | **U calc** | 31807 | 31807 | 31807 | **U calc** | 95218 | 95218 | 95218 |

Πίνακας 5 : Αποτελέσματα μοντέλου με τιμή κατωφλίου $I_k = 0.6$ για τις επιτυχείς παρατηρήσεις που καταγράφει ο αλγόριθμος Odds.

| $I_k = 0.5$ | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Χρονικό διάστημα** | 6000 | | | **Χρονικό διάστημα** | 3000 | | | **Χρονικό διάστημα** | 1000 | | |
| **Πλήθος** | 15 | 10 | 7 | **Πλήθος** | 15 | 10 | 7 | **Πλήθος** | 15 | 10 | 7 |
| **stale hits** | 533 | 347 | 317 | **stale hits** | 177 | 118 | 94 | **stale hits** | 38 | 47 | 39 |
| **Odds sum<1 Updated** | 65 | 73 | 74 | **Odds sum<1 Updated** | 75 | 74 | 76 | **Odds sum<1 Updated** | 80 | 81 | 82 |
| **Odds sum<1 not changed** | 815 | 1261 | 1870 | **Odds sum<1 not changed** | 1704 | 2700 | 3939 | **Odds sum<1 not changed** | 5549 | 8390 | 12060 |
| **Odds runs** | 880 | 1409 | 2098 | **Odds runs** | 1920 | 2961 | 4354 | **Odds runs** | 6138 | 9312 | 13402 |
| **U calc** | 15964 | 15964 | 15964 | **U calc** | 31807 | 31807 | 31807 | **U calc** | 95218 | 95218 | 95218 |

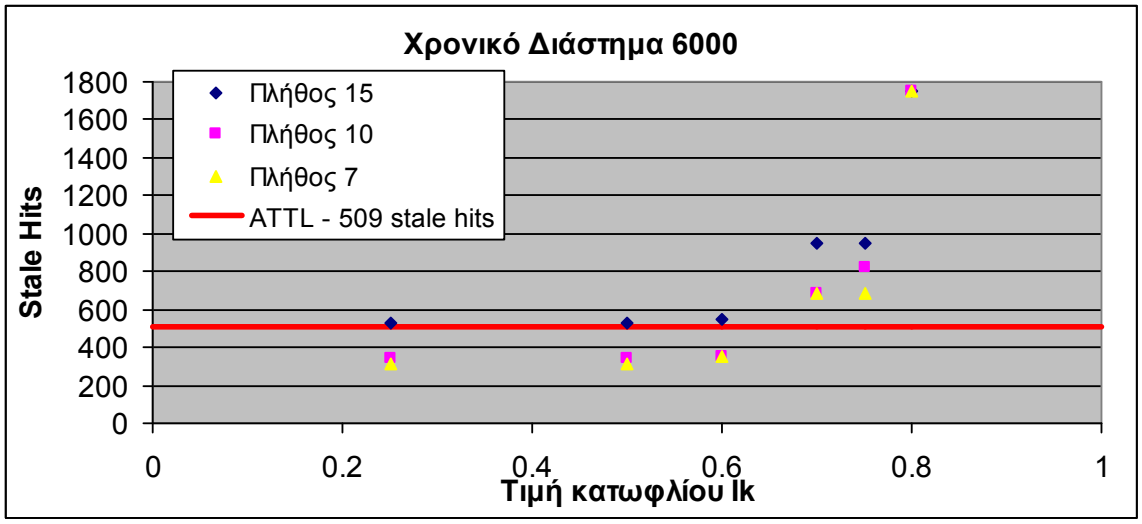
Πίνακας 6 : Αποτελέσματα μοντέλου με τιμή κατωφλίου $I_k = 0.5$ για τις επιτυχείς παρατηρήσεις που καταγράφει ο αλγόριθμος Odds.

| $I_k = 0.25$ | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Χρονικό διάστημα** | 6000 | | | **Χρονικό διάστημα** | 3000 | | | **Χρονικό διάστημα** | 1000 | | |
| **Πλήθος** | 15 | 10 | 7 | **Πλήθος** | 15 | 10 | 7 | **Πλήθος** | 15 | 10 | 7 |
| **stale hits** | 533 | 339 | 310 | **stale hits** | 171 | 116 | 89 | **stale hits** | 34 | 42 | 33 |
| **Odds sum<1 Updated** | 65 | 73 | 74 | **Odds sum<1 Updated** | 76 | 76 | 76 | **Odds sum<1 Updated** | 81 | 82 | 83 |
| **Odds sum<1 not changed** | 815 | 1336 | 2024 | **Odds sum<1 not changed** | 1844 | 2885 | 4278 | **Odds sum<1 not changed** | 6057 | 9229 | 13314 |
| **Odds runs** | 880 | 1409 | 2098 | **Odds runs** | 1920 | 2961 | 4354 | **Odds runs** | 6138 | 9312 | 13402 |
| **U calc** | 15964 | 15964 | 15964 | **U calc** | 31807 | 31807 | 31807 | **U calc** | 95218 | 95218 | 95218 |

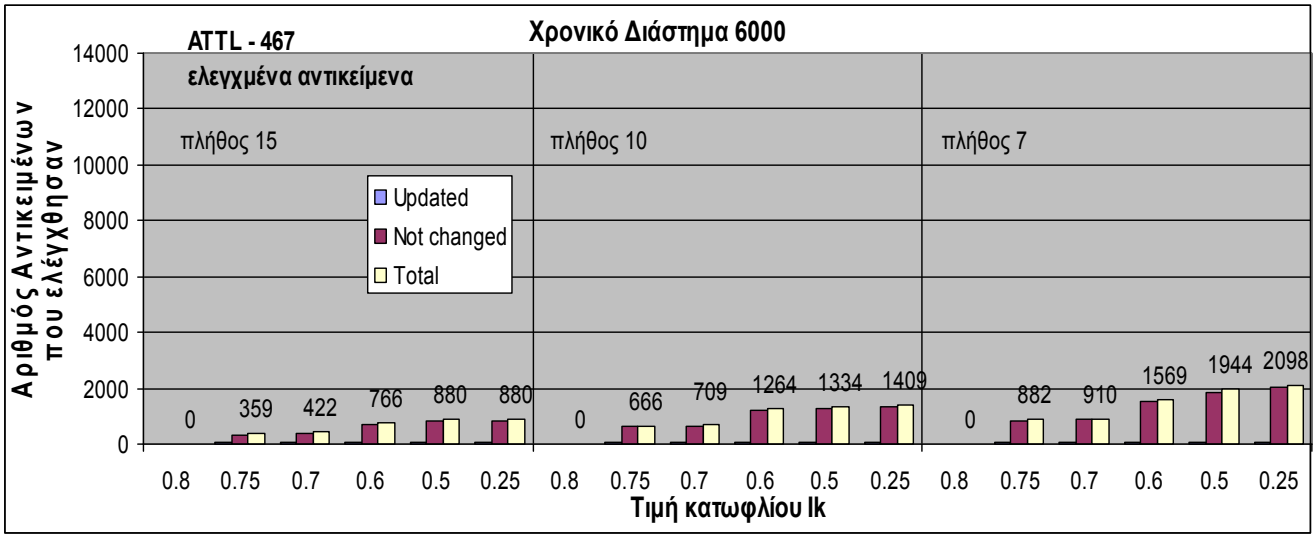
Πίνακας 7 : Αποτελέσματα μοντέλου με τιμή κατωφλίου $I_k = 0.25$ για τις επιτυχείς παρατηρήσεις που καταγράφει ο αλγόριθμος Odds.

Τα παραπάνω αποτελέσματα μπορούν να παρουσιαστούν γραφικά στα διαγράμματα που ακολουθούν. Πιο συγκεκριμένα, στα διαγράμματα παρουσιάζονται τα stale hits σαν συνάρτηση της τιμής κατωφλίου $I_k$ για τις διαφορετικές τιμές του χρονικού διαστήματος εκτέλεσης του αλγορίθμου και συγκρίνονται με τα stale hits στην περίπτωση του

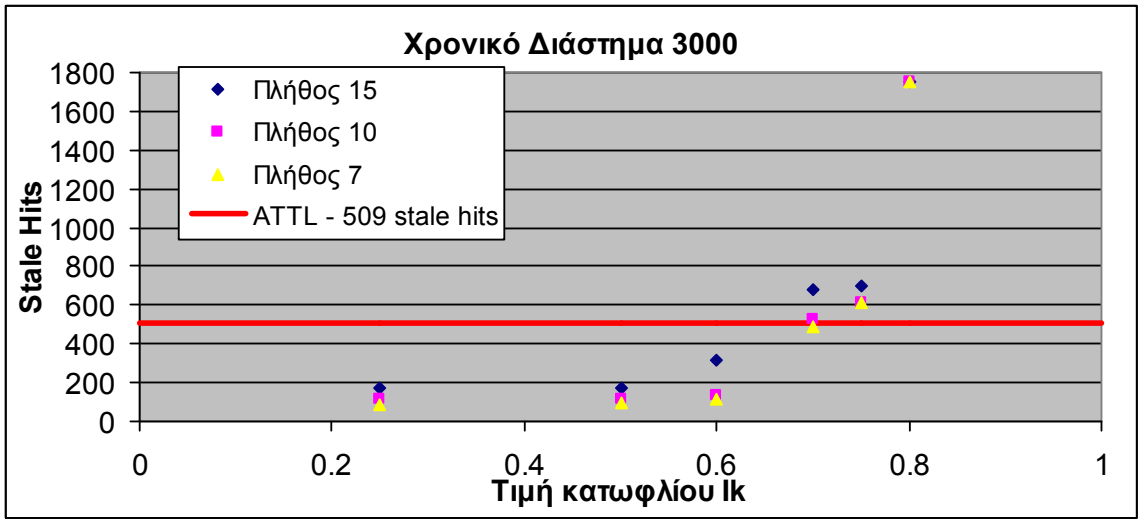
adaptive TTL. Επιπλέον, για κάθε μια περίπτωση παρουσιάζεται ο αριθμός των αντικειμένων που, μετά από απόφαση του αλγορίθμου Odds (όταν δηλαδή το άθροισμα των λόγων $r_k$ προκύπτει μικρότερο του 1), ελέγχθηκαν για να διαπιστωθεί αν έχουν τροποποιηθεί στην πηγή ή όχι. Ο αριθμός αυτός συγκρίνεται με τον αντίστοιχο αριθμό ελέγχων που πραγματοποιήθηκαν στο μηχανισμό του adaptive TTL.



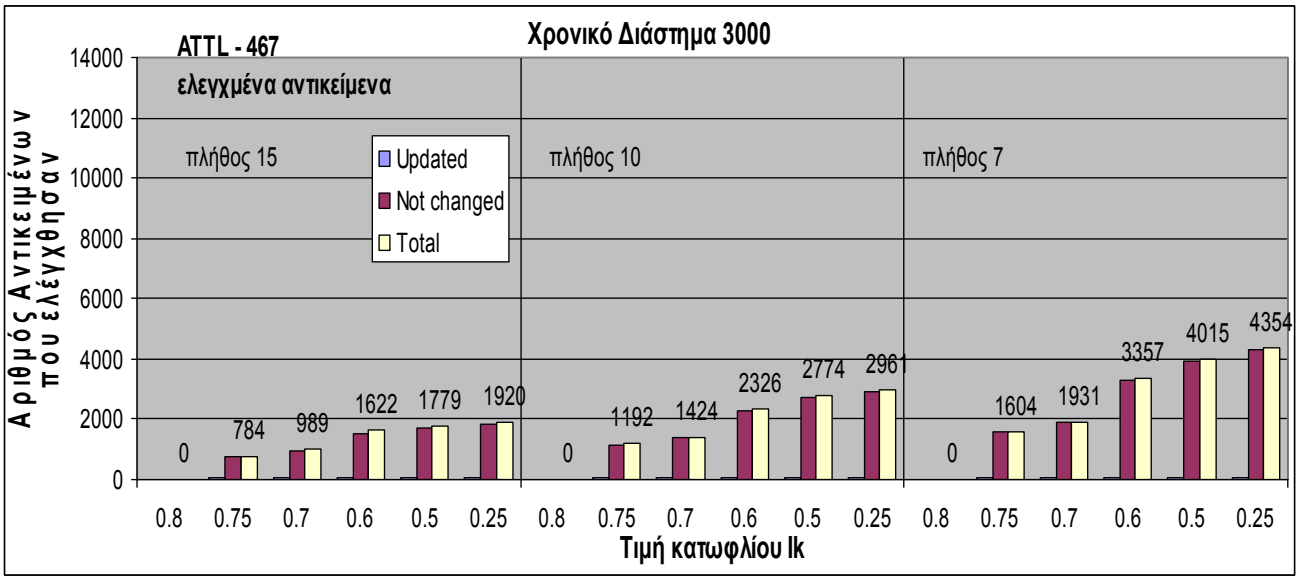

Διάγραμμα 1 : Stale Hits σε συνάρτηση της τιμής κατωφλίου Ik, για πλήθος τιμών 15, 10 και 7 με χρονικό διάστημα 6000 λεπτών μεταξύ εκτελέσεων του αλγορίθμου




Διάγραμμα 2 : Αριθμός αντικειμένων που ελέγχθηκαν για το αν έχουν τροποποιηθεί ή όχι για πλήθος τιμών 15, 10 και 7 με χρονικό διάστημα 6000 λεπτών μεταξύ εκτελέσεων του αλγορίθμου

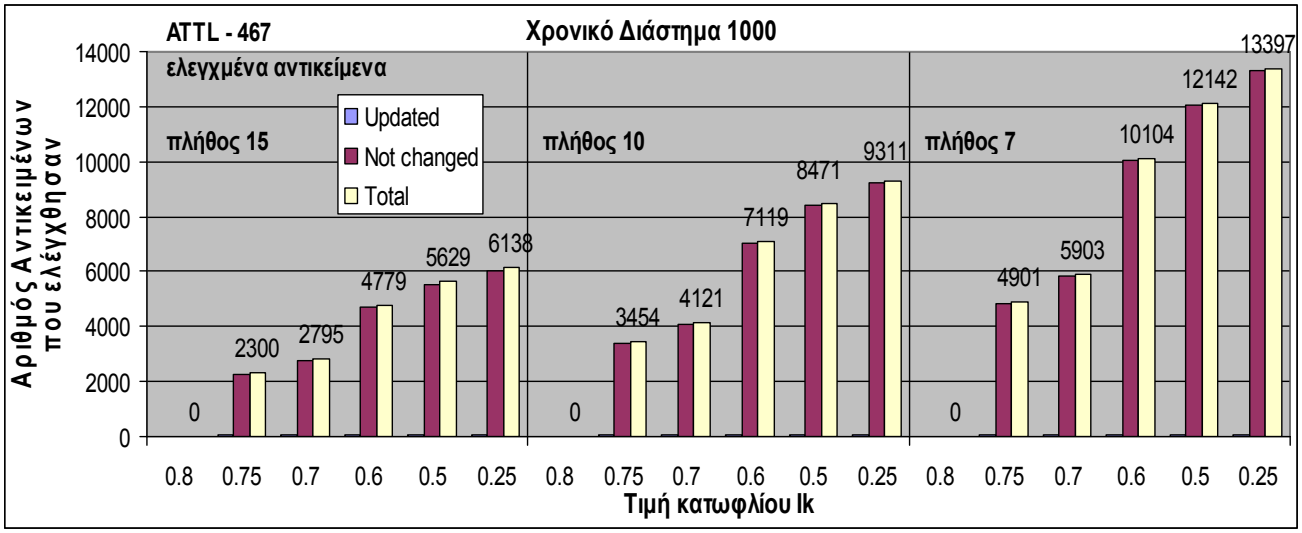



Διάγραμμα 3 : Stale Hits σε συνάρτηση της τιμής κατωφλίου Ik, για πλήθος τιμών 15, 10 και 7 με χρονικό διάστημα 3000 λεπτών μεταξύ εκτελέσεων του αλγορίθμου




Διάγραμμα 4 : Αριθμός αντικειμένων που ελέγχθηκαν για το αν έχουν τροποποιηθεί ή όχι για πλήθος τιμών 15, 10 και 7 με χρονικό διάστημα 3000 λεπτών μεταξύ εκτελέσεων του αλγορίθμου


Διάγραμμα 5 : Stale Hits σε συνάρτηση της τιμής κατωφλίου Ik, για πλήθος τιμών 15, 10 και 7 με χρονικό διάστημα 1000 λεπτών μεταξύ εκτελέσεων του αλγορίθμου




Διάγραμμα 6 : Αριθμός αντικειμένων που ελέγχθηκαν για το αν έχουν τροποποιηθεί ή όχι για πλήθος τιμών 15, 10 και 7 με χρονικό διάστημα 1000 λεπτών μεταξύ εκτελέσεων του αλγορίθμου

# 7 Συμπεράσματα

Με βάση τα αποτελέσματα των προσομοιώσεων που παρουσιάστηκαν στο προηγούμενο κεφάλαιο μπορούν να βγουν κάποια συμπεράσματα. Όσον αφορά στην υλοποίηση του μηχανισμού του adaptive TTL, υπάρχει σαφής σχέση του αριθμού των μη έγκυρων αντικειμένων που διαθέτει ο εξυπηρετητής με τη χωρητικότητα του. Όσο μικραίνει η χωρητικότητά του, μικραίνει και το ποσοστό των μη έγκυρων δεδομένων. Επίσης, σύμφωνα με τον πίνακα 1, το ποσοστό των stale hits κυμαίνεται γύρω στο 5% για την πλειοψηφία των περιπτώσεων (μέγεθος cache 400, 300, 250, 200), ενώ για μεγέθη ίσα με 150 και 100 κυμαίνεται γύρω στο 4% (4.5% και ~4% αντίστοιχα). Τα συγκεκριμένα ποσοστά γίνονται ακόμα μικρότερα αν το ποσοστό της μη έγκυρης πληροφορίας υπολογιστεί στο συνολικό αριθμό των αντικειμένων. Ωστόσο, η τάση το ποσοστό των stale αντικειμένων να είναι κοντά στο 5% δεν αλλάζει.

Πιο συγκεκριμένα για την περίπτωση που μας ενδιαφέρει, όπου η χωρητικότητα του caching server λαμβάνεται ίση με 400, βρέθηκαν 509 stale hits. Επίσης, διαπιστώνεται ότι μετά το πέρας του χρόνου ζωής τους, η συντριπτική πλειοψηφία των αντικειμένων που ελέγχθηκαν δεν είχαν τροποποιηθεί. Ειδικότερα, από τον πίνακα 1 φαίνεται ότι συνολικά ελέγχθηκαν 467 αντικείμενα, από τα οποία βρέθηκε ότι τα 39 είχαν ήδη τροποποιηθεί ένω τα 428 είχαν παραμείνει ίδια.

Για την αξιολόγηση της υλοποίησης του adaptive TTL μπορεί να αναφερθεί ότι τα αποτελέσματα συμφωνούν με το [13], σύμφωνα με το συγγραφέα του οποίου το adaptive TTL, με βάση παλαιότερες μελέτες των Cate, Gwertzman και Seltzer, περιορίζει την πιθανότητα για μη έγκυρη πληροφορία στο 5% περίπου. Ωστόσο, σύμφωνα με το [2] οι Feldmann et al., στηριγμένοι σε ίχνη από την εταιρία AT&T, έδειξαν ότι το adaptive TTL με k = 0.2 εμφανίζει ποσοστό μη έγκυρης πληροφορίας ίσο με 0.8% των hits ή 0.22% στο σύνολο των αντικειμένων.

Όσον αφορά στην υλοποίηση του μοντέλου με τον αλγόριθμο Odds, τόσο από τα διαγράμματα, όσο και από τους πίνακες προκύπτει ότι ο αριθμός των stale hits εξαρτάται από 3 παράγοντες : το χρονικό διάστημα μεταξύ των εκτελέσεων του αλγορίθμου, το πλήθος των τιμών της συνάρτησης U στις οποίες εφαρμόζεται ο αλγόριθμος και την τιμή για την οποία θεωρείται μια παρατήρηση επιτυχής. Ελέγχθηκαν 3 χρονικά διαστήματα (6000, 3000 και 1000) για τα οποία, με βάση τα διαγράμματα 1, 3 και 5, προκύπτει το συμπέρασμα ότι όσο το χρονικό διάστημα μεταξύ των εκτελέσεων του αλγορίθμου μειώνεται, τόσο πιο μικρός είναι ο αριθμός των stale hits. Ειδικά στην περίπτωση των 1000, όπως φαίνεται στο διάγραμμα 5, όλα σχεδόν τα αποτελέσματα δίνουν μικρότερο stale rate από το adaptive TTL. Αυτό οφείλεται στο ότι όσο πιο μικρό είναι το χρονικό διάστημα, τόσο πιο συχνά εκτελείται ο αλγόριθμος με αποτέλεσμα να ελέγχονται και να ενημερώνονται περισσότερα αντικείμενα. Πράγματι, από τους πίνακες φαίνεται ότι για πλήθος π.χ. 10 τιμών ο αλγόριθμος εφαρμόζεται σε 1409 αντικείμενα όταν το χρονικό διάστημα είναι 6000, σε 2961 αντικείμενα όταν το χρονικό διάστημα είναι 3000 και σε 9312 αντικείμενα όταν είναι 1000. Επιπλέον, για την ίδια τιμή κατωφλίου $I_k$, π.χ. 0.6, και για το ίδιο πλήθος, π.χ. 10, ο αριθμός των αντικειμένων που διαπιστώνεται ότι έχουν αλλάξει, και κατά συνέπεια έχουν ενημερωθεί εγκαίρως, αυξάνεται από 72 σε 73 και σε 80 με τη μείωση του χρονικού διαστήματος από 6000 σε 3000 και σε 1000 αντίστοιχα.

Παράλληλα όμως, όπως φαίνεται στα διαγράμματα 2, 4 και 6, για όλες τις περιπτώσεις όσο μικραίνει το χρονικό διάστημα αυξάνεται ο αριθμός των αντικειμένων (Total) που ελέγχονται για τον αν έχουν τυχόν τροποποιηθεί στην πηγή. Αυτό φαίνεται χαρακτηριστικά συγκρίνοντας τα διαγράμματα 2, 4 και 6. Ειδικά για χρονικό διάστημα 1000 (διάγραμμα 6), ο αριθμός των ελεγχμένων αντικειμένων είναι πολλαπλάσιος από ότι στις άλλες περιπτώσεις. Μάλιστα, στη συντριπτική τους πλειοψηφία τα αντικείμενα που ελέγχθηκαν δεν είχαν υποστεί αλλαγές, όπως φαίνεται στα αντίστοιχα διαγράμματα. Το αντίτιμο δηλαδή για ένα μειωμένο αριθμό stale hits είναι ο αυξημένος αριθμός ελέγχων.

Για το πλήθος των τιμών της συνάρτησης U για τις οποίες εφαρμόζεται ο αλγόριθμος χρησιμοποιήθηκαν 3 τιμές, 15, 10 και 7. Από τα διαγράμματα 1 και 3 προκύπτει ότι όσο μικραίνει το πλήθος, τόσο μικραίνει και ο αριθμός των stale hits. Στο διάγραμμα 5 δεν είναι εμφανής η διαφορά, αλλά η τάση επιβεβαιώνεται από τους πίνακες 2-7. Η τάση αυτή οφείλεται στο ότι όσο πιο μικρό είναι το πλήθος των τιμών της U, τόσο πιο συχνά εκτελείται ο αλγόριθμος για κάποιο αντικείμενο. Για παράδειγμα, από τους πίνακες 2 – 7 φαίνεται ότι για το ίδιο χρονικό διάστημα, π.χ. 6000, ο αλγόριθμος εφαρμόζεται σε 880, αντικείμενα όταν το πλήθος των τιμών είναι 15, σε 1409 αντικείμενα όταν το πλήθος των τιμών είναι 10 και σε 2098 αντικείμενα για πλήθος τιμών 7. Για το λόγο αυτό όμως, αυξάνεται ο αριθμός των ελέγχων που γίνονται για να διαπιστωθεί αν ένα αντικείμενο έχει τροποποιηθεί από την πηγή ή όχι. Η αύξηση αυτή φαίνεται ξεκάθαρα στα διαγράμματα 2, 4 και 6.

Τέλος, σημαντική για τη διαμόρφωση των αποτελεσμάτων είναι η επίδραση της τιμής κατωφλίου για την οποία θεωρείται μια παρατήρηση επιτυχής. Όσο πιο μικρή είναι αυτή η τιμή, τόσο πιο πολλές παρατηρήσεις θεωρούνται επιτυχείς, με αποτέλεσμα τα αθροίσματα των odds στον αλγόριθμο να καταλήγουν σε τιμή μικρότερη της μονάδας πιο γρήγορα. Η ελάττωση της τιμής κατωφλίου προκαλεί μείωση στον αριθμό των stale hits, όπως φαίνεται τόσο στους πίνακες 2-7 όσο και στα διαγράμματα 1, 3 και 5, αλλά συνοδεύεται από αύξηση των ελέγχων για τροποποιήσεις των αντικειμένων όπως γίνεται φανερό στα διαγράμματα 2, 4 και 6. Ενδεικτικά αναφέρεται ότι για χρονικό διάστημα 1000, με πλήθος τιμών 7 και τιμή κατωφλίου 0.25 ο αριθμός των stale hits μειώνεται αισθητά στα 33, σε σχέση με το adaptive TTL που είναι 509. Ωστόσο, ο αριθμός των ελέγχων στην πρώτη περίπτωση είναι 13397, κατά πολύ μεγαλύτερος από τη δεύτερη περίπτωση που είναι 467.

# Βιβλιογραφία

1. A Web Caching Primer, Brian D. Davison
2. Web Caching and Replication, Michael Rabinovich, Oliver Spatscheck, Addison Wesley, 2001
3. A Survey of Web Caching Schemes for the Internet, Jia Wang
4. World-Wide Web Cache Consistency: Gwertzman, Seltzer (1996)
5. Intelligent Caching for World Wide Web Objects, Duane Wessels
6. A Hierarchical Internet Object Cache: Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, Kurt J. Worrell
7. Maintaining Strong Cache Consistency in the World Wide Web: Cao, Liu
8. An Update-Risk Based Approach to TTL Estimation in Web Caching, Jeong-Joon Leey, Kyu-Young Whangy, Byung Suk Leez, Ji-Woong Changy
9. Thomas S. Ferguson, "Optimal Stopping and Applications", Mathematics Department UCLA, http://www.math.ucla.edu/~tom/Stopping/Contents.html.
10. Knowing When To Stop, Theodore Hill, Άρθρο του American Scientist, March-April 2009, Volume 97, Number 2, Page: 126, http://www.americanscientist.org/issues/feature/knowing-when-to-stop
11. http://www.thefullwiki.org/Odds_algorithm
12. The Odds-algorithm based on sequential updating and its performance, F.Thomas Bruss, Guy Louchard, Universit´ e Libre de BruxellesApril 21, 2008
13. Web Caching and Its Applications, S. V. Nagaraj, 2004

# Παράρτημα

Για κάθε μοντέλο χρησιμοποιήθηκαν 5 αρχεία. Κοινό κώδικα για τις προσομοιώσεις ATTL και Odds Algorithm with sequential estimation αποτελούν τα αρχεία **entryUrl.java**, **RequestData.java**, και **cachentry.java**. Τα αρχεία που διαφέρουν είναι τα **Main.java** και **CacheHTable.java**.

Κοινός Κώδικας **entryUrl.java**, **RequestData.java**, και **cachentry.java** για προσομοίωση ATTL και Odds algorithm with sequential estimation

## entryUrl.java

```
/*
 * Created by Dimitris Lorentzos
 */
public class entryUrl
{
      private String site;
      private String item;

      public entryUrl(String site, String item) {
                  this.site=site;
                  this.item=item;
          }

       public String getSite() {
             return site;
       }

       public String getItem() {
             return item;
       }

       public boolean UrlcontainsSite (entryUrl url,String site) {
              return (url.getSite().equals(site));
       }

   @Override
   public boolean equals(Object obj) {
      if (obj == null) {
         return false;
      }
      if (getClass() != obj.getClass()) {
         return false;
      }
```

```
        final entryUrl other = (entryUrl) obj;
        if ((this.site == null) ? (other.site != null) : !this.site.equals(other.site)) {
            return false;
        }
        if ((this.item == null) ? (other.item != null) : !this.item.equals(other.item)) {
            return false;
        }
        return true;
    }
    @Override
    public int hashCode() {
        int hash = 7;
        hash = 97 * hash + (this.site != null ? this.site.hashCode() : 0);
        hash = 97 * hash + (this.item != null ? this.item.hashCode() : 0);
        return hash;
    }
}
```

## RequestData.java

```
/*
 * Created by Manos Spanoudakis
 * Changed by Dimitris Lorentzos
 */

public class RequestData
{
        private double time;
        private String userId;

    public RequestData(double t)
        {
                this.time=t;
        }

        /**
         * @return
         */
        public double getTime()
        {
                return time;
        }

        /**
         * @return
         */
        public String getUserId()
        {
                return userId;
```

```
        }

        /**
         * @param calendar
         */
        public void setTime(double calendar)
        {
                time = calendar;
        }

        /**
         * @param string
         */
        public void setUserId(String string)
        {
                userId = string;
        }

}
```

## cachentry.java

```
import java.util.Vector;
import java.util.*;
/*
 * Created by Manos Spanoudakis
 * Chnaged by Dimitris Lorentzos
 */
public class cachentry
{
        private entryUrl url;
        private long size;
        private long numRequests;
        private long hits;
        private double created;
        private double lastAccess;
        private boolean deleted;
        private Vector history;

      private long SitenumRequests;
      private long SitenumHits;
      private boolean inCache;
      private ArrayList<Double> FunctionU;
      private double expires;
      private double ExpirationTime;
      private double lastModified;

        public cachentry(entryUrl url, long size, double created)
        {
```

```
                this.url=url;
                this.size=size;
                numRequests=1;
                hits=0;
                this.created=created;
                history=new Vector();
                lastAccess=created;
                deleted=false;
        expires = -1;

        SitenumRequests = 1;
        SitenumHits=0;
        inCache=false;
        FunctionU = new ArrayList<Double>();
}

public void advanceSiteNumRequests()
    {
                SitenumRequests++;
    }

public void advanceSiteNumHits()
    {
                SitenumHits++;
    }

public void advanceNumRequests()
    {
                numRequests++;
    }

    public void advanceHits()
    {
                hits++;
    }

/* Get and Set methods */

public double getExpires()
{
   return expires;
}

public double getLastModified()
{
   return lastModified;
}
     /**
      * @return */
```

```
    public double getCreated()
    {
            return created;
    }

    /**
     * @return
     */
    public double getLastAccess()
    {
            return lastAccess;
    }

    /**
     * @return
     */
    public long getNumRequests()
    {
            return numRequests;
    }

 public long getSiteNumRequests()
    {
            return SitenumRequests;
    }

 public long getSiteNumHits()
    {
            return SitenumHits;
    }

public ArrayList<Double> getListFunctionU()
    {
            return FunctionU;
    }

    /**
     * @return
     */
    public long getSize()
    {
            return size;
    }

    /**
     * @return*/
 public entryUrl getUrl()
    {
            return url;
```

```
    }

    /**
     * @param calendar
     */
    public void setCreated(double calendar)
    {
            created = calendar;
    }
public void setExpires(double TimeToLive)
    {
            expires = TimeToLive;
    }

 public void setExpirationTime(double TimeToLive)
    {
            ExpirationTime = TimeToLive;
    }

    /**
     * @param calendar
     */
    public void setLastAccess(double calendar)
    {
            lastAccess = calendar;
    }

public void setLastModified(double modified)
{
   lastModified = modified;
}

    /**
     * @param l
     */
    public void setNumRequests(long l)
    {
            numRequests = l;
    }

public void setSiteNumRequests(long l)
    {
            SitenumRequests = l;
    }

public void setSiteNumHits(long l)
{
      SitenumHits = l;
}
```

```
    /**
     * @param l
     */
    public void setSize(long l)
    {
            size = l;
    }

    /**
     * @return
     */
    public long getHits()
    {
            return hits;
    }

public double getExpirationTime()
    {
            return ExpirationTime;
    }

    /**
     * @param l
     */
    public void setHits(long l)
    {
            hits = l;
    }

    public String toString()
    {
            String ret=new String("----Cache Entry----\n");
            ret+=("site:"+(this.getUrl()).getSite() +", item:"+ (this.getUrl()).getItem() + "\n");
      ret+=("size:"+this.getSize() + "\n");
            ret+=("numRequests:"+this.getNumRequests() + "\n");
            ret+=("SitenumRequests:"+this.getSiteNumRequests() + "\n");
            ret+=("SitenumHits:"+this.getSiteNumHits() + "\n");
            ret+=("hits:"+this.getHits() + "\n");
            ret+=("created:"+this.getCreated() + "\n");
            ret+=("lastAccess:"+this.getLastAccess() + "\n");
            ret+=("\tFunction U List:"+this.getListFunctionU() + "\n");
            ret+="\tRequest list ---\n";
            Vector r=this.getHistory();

            for (int i=0;i<r.size();i++)
            {
                    RequestData rd=(RequestData)r.elementAt(i);
                    ret+= "\t\t User:" + rd.getUserId() + " date:" + rd.getTime()+"\n";
```

```
                }
                ret+="\t----End of request list\n";
                ret+=("-------End Of Cache Entry------\n");
                return ret;
        }

        /**
         * @return
         */
        public Vector getHistory()
        {
                return history;
        }

        /**
         * @param vector
         */
        public void setHistory(Vector vector)
        {
                history = vector;
        }

        public void addRequest(RequestData r)
        {
                this.history.add(r);
        }

        /**
         * @return
         */
        public boolean isDeleted() {
                return deleted;
        }

    public boolean inCache() {
                return inCache;
        }

        /**
         * @param b
         */
        public void setDeleted(boolean b) {
                deleted = b; }

    public void setinCache(boolean b) {
                inCache = b; }

}
```

Κώδικας **Main.java** και **CacheHTable.java**
για προσομοίωση ATTL

## Main.java

```
import java.util.*;
import java.io.*;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Dimitris
 */
public class Main {
static String input = "";


    public static void main(String[] args) {

        try {

            FileReader infile = null;
            String thisLine = "";
            // Requests to caching server
            input = "caching_server_traces.txt";
            infile = new FileReader(input);
            BufferedReader b = new BufferedReader(infile);
            CacheHTable a = new CacheHTable();

            thisLine = b.readLine();
            int count=0;
            while ( (thisLine = b.readLine()) != null) {
                String[] tmp = thisLine.split(" ");
                entryUrl Url = new entryUrl(tmp[0],tmp[2]);
                //size is 1 for all items
                cachentry alpha = new cachentry(Url, 1 , Double.parseDouble(tmp[3]));
                // Request served by caching server
                a.handleEntry(alpha);
                count++;
            }

System.out.println("\n"+"Number of total requests: "+a.getTotalNumRequests()+"\n"
        +"Current Size of caching server: "+a.getCurrentSize()+"\n"
        +"Maximum Size of caching server: "+a.getMaxSize()+"\n");
System.out.println("Changed in source while TTL in effect:"+a.getChangedInSource()+"\n"
        +"Not changed in source while TTL in effect:"+ a.getNotChangedInSource()+ "\n"
        +"Total Number of checks: "+(a.getChangedInSource()+a.getNotChangedInSource()));
```

```
a.clear();

      } catch (IOException ex) {
         Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
      }

   }

}
```

## CacheHTable.java

```
import java.util.*;
import java.io.*;
import java.util.logging.Level;
import java.util.logging.Logger;

/* A cache Hash table */

/**
        @author Manos Spanoudakis
        @version 1.0

 *      Changed by Dimitris LOrentzos

        CacheHtable is a Hash table of cachentries
*/

public class CacheHTable{

private int size=400;
private long MaxSize;
private long totalSize;
private long currentSize;
private long totalNumRequests;

int hits=0,stale=0, changed_in_source=0, not_changed_in_source=0 ;


   Hashtable cachetable;

        public CacheHTable()
        {
           cachetable = new Hashtable(size);
             MaxSize = size;
             totalNumRequests = 0;
        }
```

```
    public CacheHTable(int s)
    {  /* Ean dwsei kapoio sugkekrimmeno mege8os */
       size=s;
         MaxSize = size;
       cachetable = new Hashtable();
         totalNumRequests = 0;
    }


    public Hashtable getTable(){ return cachetable;}

  //--------------
  public void handleEntry(cachentry ce)
    {
       System.out.println("----------\nMethod 'handleEntry' invoked");
       entryUrl url=ce.getUrl();
       ce.advanceNumRequests();

         advanceTotalSize(ce.getSize());


         if (containsUrl(url))
       {
                cachentry c=(cachentry)cachetable.get(url);
              if (isValid(c,c.getSize()) && (!c.isDeleted()))
                // Hit !
                {
                        System.out.println("Cache hit");
if(c.getExpires() != -1) hits++;
                        c.advanceNumRequests();
                    c.advanceSiteNumRequests();
                        c.advanceHits();
                    c.advanceSiteNumHits();
                    c.setinCache(true);
                        double requestDate;
                        requestDate = ce.getLastAccess();
                    c.setLastAccess(requestDate);
                    RequestData r=new RequestData(requestDate);
                        c.addRequest(r);
UpdateATTL( c, "source_traces.txt");
CheckIfStale( (cachentry)cachetable.get(ce.getUrl()),"source_traces.txt" );

                }
                if (c.isDeleted())
                { // Re-enter entry in cache...
                        c.setDeleted(false);
                        c.advanceNumRequests();
                        double requestDate;
                        requestDate = ce.getLastAccess();
```

```
                    c.setLastAccess(requestDate);
                RequestData r=new RequestData(requestDate);
                    c.addRequest(r);
                }
        }
        else
        {//create a new cache entry
                System.out.println("Creating new Cache Entry");
                double requestDate;
               requestDate = ce.getLastAccess();

                cachentry c=new cachentry(ce.getUrl(),ce.getSize(),requestDate);
                c.setLastAccess(requestDate);
                RequestData r=new RequestData(requestDate);
CalcATTL(c,"source_traces.txt");
                c.addRequest(r);
                cachetable.put(url,c);
                advanceSize(ce.getSize());
          }
          advanceTotalNumRequests();
        cleanup(this);

System.out.println("Site:"+ce.getUrl().getSite()+", Item:"+ce.getUrl().getItem()+"\n"
      +"Last Modified at: "+( (cachentry)cachetable.get(ce.getUrl()) ).getLastModified()
      +"\n"+"Created at: "+( (cachentry)cachetable.get(ce.getUrl()) ).getCreated()
      + "\n"+"Last Accessed at: "+( (cachentry)cachetable.get(ce.getUrl()) ).getLastAccess()
      + "\n"+"Expires at: "+( (cachentry)cachetable.get(ce.getUrl()) ).getExpires() );

System.out.println("\n"+"hits so far: "+hits+"\nstale so far: "+stale);
System.out.println("Method 'handleEntry' terminated");
       }

   public entryUrl getLRUEntryKey()
   {
        double min = 14000000.000;
          entryUrl keyvalue=null;
        entryUrl currentkey=null;

        Enumeration e=cachetable.keys();

        while (e.hasMoreElements())
        {
                currentkey=(entryUrl)e.nextElement();
                if (min >(((cachentry)cachetable.get(currentkey)).getLastAccess()))
                {
                        keyvalue=currentkey;
                        min=((cachentry)cachetable.get(currentkey)).getLastAccess();
                }
                //System.out.println("key="+currentkey);
```

```
            //System.out.println("Last access:" +
((cachentry)cachetable.get(currentkey)).getLastAccess().getTime());

            }
            return ((cachentry)cachetable.get(keyvalue)).getUrl();
        }


        /**
            Searches for a url in the cachtable

            @param s The url we are searching for
            @return true if it is found, or false if it is not found

         */
        public boolean findUrl(String s)
        {
            return cachetable.containsKey(s);
        }

       public cachentry getEntry(entryUrl urlname)
        {
            return (cachentry)cachetable.get(urlname);
        }

        public synchronized void clear()
        {
            cachetable.clear();
        }

        public int getSize()
        {
            return cachetable.size();
        }

       public void print ()
        {
            Enumeration e=cachetable.keys();
            entryUrl currentkey;

            while (e.hasMoreElements())
            {
                        currentkey=(entryUrl)e.nextElement();
                        System.out.println(cachetable.get(currentkey));
                        System.out.println("--------------------");

            }
        }
```

```
    public long getTotalNumRequests(){return totalNumRequests;}

    /**
     @return The max size in bytes of the Cache Directory
     */
     public long getMaxSize(){return MaxSize;}
    /**
     @return The current Cache Directory size in bytes of the Cache Directory
     */
     public long getCurrentSize(){return currentSize;}
/**
     Advances the current Cache Directory size by 'size' bytes. This happens when
     a new entry is added in cache
     @param size The amount of bytes to add in the current Cache Directory size
     */
     public synchronized void advCacheCurrentSize(long size){currentSize+=size;}

     /**
     Reduces the current Cache Directory size by 'size' bytes. This happens when
     an entry is removed from cache
     */
     public synchronized void redCacheCurrentSize(long size){currentSize-=size;}

    public synchronized void addentry(String urlname,cachentry h)
     {
        cachetable.put(urlname,h);
        // else throw MaxCacheException
           advCacheCurrentSize(h.getSize());
     }

          public synchronized void removeEntry(entryUrl urlname)
     {
        cachetable.remove(urlname);

     }

 public synchronized void UpdateATTL(cachentry c, String input)
     {
   if (c.getLastAccess()<c.getExpires()) {return;}
 else if ( (c.getLastAccess()>c.getExpires()) && (c.getExpires()!=-1) ){ try {

    System.out.println("Updating ATTL...");
         //check source file
         FileReader infile = null;
         String line = "";
         infile = new FileReader(input);
         BufferedReader b = new BufferedReader(infile);
         line = b.readLine();
```

```
          while ( (line = b.readLine()) != null) {
             String[] tmp = line.split(" ");
             if ( (c.getUrl()).getSite().equals(tmp[0]) && (c.getUrl()).getItem().equals(tmp[2]) ){
                if (Double.parseDouble(tmp[3])>c.getLastModified() &&
                    Double.parseDouble(tmp[3])<c.getLastAccess()) {
              c.setLastModified(Double.parseDouble(tmp[3]));
              double s= c.getLastAccess()-c.getLastModified();
              c.setExpires( c.getLastAccess() + 0.2*s );
              changed_in_source++;
                }
             } else if (Double.parseDouble(tmp[3])>c.getLastAccess()) {
                    double s= c.getLastAccess()-c.getLastModified();
                    c.setExpires( c.getLastAccess() + 0.2*s );
                    not_changed_in_source++;break;}

          }
System.out.println("Last Modified at: "+c.getLastModified()+", "
       +"Created at: "+c.getCreated()+", "+"Last Accessed at: "+c.getLastAccess()
       +", "+"New Expires at: "+c.getExpires());
System.out.println("Finished Updating ATTL...");
    }
        catch (IOException ex) {
          Logger.getLogger(CacheHTable.class.getName()).log(Level.SEVERE, null, ex);
     }
   }
 }
     public synchronized void CheckIfStale(cachentry c, String input)
        {
        if ( (c.getExpires()!=-1) ){
            try {
          System.out.println("Checking if stale hit...");
          //check source file
          FileReader infile = null;
          String line = "";
          infile = new FileReader(input);
          BufferedReader b = new BufferedReader(infile);
          line = b.readLine();
double temp_last_mod = 0;
          while ( (line = b.readLine()) != null) {
             String[] tmp = line.split(" ");
             if ( (c.getUrl()).getSite().equals(tmp[0]) && (c.getUrl()).getItem().equals(tmp[2]) ){
                if (Double.parseDouble(tmp[3])>c.getLastModified() &&
                    Double.parseDouble(tmp[3])<c.getLastAccess()) {
              stale++;
              if (temp_last_mod !=0) {stale--;}
              temp_last_mod = Double.parseDouble(tmp[3]);
                 System.out.println("Stale! Modified at "+tmp[3]);
             } else if (Double.parseDouble(tmp[3])>c.getLastAccess()) {return;}
             }
```

```
        }
     }
        catch (IOException ex) {
         Logger.getLogger(CacheHTable.class.getName()).log(Level.SEVERE, null, ex);
    } }
    }

    public synchronized void CalcATTL(cachentry c, String input)
       {
       double g = 0;
       int NumStale = 0;
       int NumGood = 0;
System.out.println("Calculating ATTL...");
       try {
         //check source file
         FileReader infile = null;
         String line = "";
         infile = new FileReader(input);
         BufferedReader b = new BufferedReader(infile);
         line = b.readLine();

         while ( (line = b.readLine()) != null) {
            String[] tmp = line.split(" ");
            if ( (c.getUrl()).getSite().equals(tmp[0]) && (c.getUrl()).getItem().equals(tmp[2]) ) {
               if ( Double.parseDouble(tmp[3])>c.getCreated() )
               {
                  break;
               }
               else
               {
                  c.setLastModified(Double.parseDouble(tmp[3]));
                  double ATTL = c.getCreated()-c.getLastModified();
                  c.setExpires( c.getCreated() + 0.2*ATTL);
               }
            }
         }
if (c.getExpires() == -1){c.setLastModified(0);double ATTL = c.getCreated()-c.getLastModified();
                  c.setExpires( c.getCreated() + 0.2*ATTL);
                  }
System.out.println("Last Modified at: "+c.getLastModified()+", "
      +"Created at: "+c.getCreated()+", "+"Last Access at: "+c.getLastAccess()+", "
      +"Expires at: "+c.getExpires());
System.out.println("Finished Calculating ATTL...");
       }
       catch (IOException ex) {
         Logger.getLogger(CacheHTable.class.getName()).log(Level.SEVERE, null, ex);
    }
   }
```

```
    public synchronized void cleanup(CacheHTable cachet)
      {
          System.out.println("    Method 'cleanUp' invoked ");
        System.out.println("       Current size is:" + this.currentSize);
        System.out.println("       Maximum size is:" + this.MaxSize);

          if (getCurrentSize()<(0.8)*getMaxSize()) {
                System.out.println("    Method 'cleanUp' terminated ");
              return;}

        System.out.println("Cache reached 80% of MaximumSize. Cleaning up ...");
        while (getCurrentSize()>((0.6)*getMaxSize()))
        {
                entryUrl victimKey=cachet.getLRUEntryKey();

                long filesize=( (cachet.getEntry(victimKey) ).getSize() );

                        redCacheCurrentSize(filesize);
                        cachet.removeEntry(victimKey);
System.out.println("Removed: " + "Site " + victimKey.getSite() + ", Item " + victimKey.getItem());
        }
          System.out.println("Method 'cleanUp' terminated");

      }

public void advanceSize(long s)
      {
        currentSize+=s;
      }

public void advanceTotalSize(long num)
      {
        totalSize+=num;
      }

public void advanceTotalNumRequests()
      {
        totalNumRequests+=1;
      }

public boolean containsUrl(entryUrl url)
      {
        return cachetable.containsKey(url);
      }

public boolean isValid(cachentry c,long size)
      {
        return (size==c.getSize());
      }
```

```
public long getChangedInSource()
	{
	return changed_in_source;
	}

public long getNotChangedInSource()
	{
	return not_changed_in_source;
	}

}
```

Κώδικας **Main.java** και **CacheHTable.java**
για προσομοίωση Odds algorithm with sequential estimation

## Main.java

```
import java.util.*;
import java.io.*;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Dimitris
 */
public class Main {
static String input = "";

public static void main(String[] args) {

      try {

         FileReader infile = null;
         String thisLine = "";
         // Requests to caching server
         input = "caching_server_traces.txt";
         infile = new FileReader(input);
         BufferedReader b = new BufferedReader(infile);
         CacheHTable a = new CacheHTable();

         thisLine = b.readLine();
         int count=0;
         while ( (thisLine = b.readLine()) != null) {
            String[] tmp = thisLine.split(" ");
            entryUrl Url = new entryUrl(tmp[0],tmp[2]);

            //size is 1 for all items
            cachentry alpha = new cachentry(Url, 1 , Double.parseDouble(tmp[3]));
            // Request served by caching server
            a.handleEntry(alpha);
            count++;
         }

System.out.println("\n"+"Number of total requests: "+a.getTotalNumRequests()+"\n"
               +"Current Size of caching server: "+a.getCurrentSize()+"\n"
               +"Maximum Size of caching server: "+a.getMaxSize()+"\n");

System.out.println("Changed in source when checked: "+a.getAllUpdated()+" "
      +"\nNot changed in source when checked: "+ a.getAllSame()
```

```
        +"\nTotal number of checks: "+(a.getAllUpdated()+a.getAllSame())
        +"\nItems tested in Odds Algorithm :"+a.getNumItemsTested()
        +"\nNumber of Stale hits: "+a.getStale()
        +"\ntimes U is calculated: "+a.getNumItemsCalcU() );

a.clear();

        } catch (IOException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }

    }

}
```

## CacheHTable.java

```
import java.util.*;
import java.io.*;
import java.util.logging.Level;
import java.util.logging.Logger;

/* A cache Hash table */

/**
            @author Manos Spanoudakis
            @version 1.0
 *
 *      Changed by Dimitris Lorentzos

            CacheHtable is a Hash table of cachentries
*/

public class CacheHTable{
private int size=400;
private long MaxSize;
private long totalSize;
private long currentSize;
private long totalNumRequests;
private double calcfunctionU;
private long allUpdated, allSame, Num_Items_tested=0;
int totalrequests=0, hits=0, stale=0, num_items_calc_U=0;


    Hashtable cachetable;

            public CacheHTable()
            {
                    cachetable = new Hashtable(size);
```

```
        MaxSize = size;
        totalNumRequests = 0;
        calcfunctionU = 1201000.000;
        allUpdated = 0;
        allSame = 0;
      }

      public CacheHTable(int s)
      {  /* Ean dwsei kapoio sugkekrimmeno mege8os */
            size=s;
        MaxSize = size;
            cachetable = new Hashtable();
        totalNumRequests = 0;
        calcfunctionU = 1000;
        allUpdated = 0;
        allSame = 0;
      }


      public Hashtable getTable(){ return cachetable;}

    //--------------
    public void handleEntry(cachentry ce)
      {
            System.out.println("----------\nMethod 'handleEntry' invoked");
            entryUrl url=ce.getUrl();
            ce.advanceNumRequests();

        advanceTotalSize(ce.getSize());

        if (containsUrl(url))
            {
                  cachentry c=(cachentry)cachetable.get(url);
              if (isValid(c,c.getSize()) && (!c.isDeleted()))
                  // Hit !
                  {
                        System.out.println("Cache hit");
if(c.getExpires() != 0) hits++;

                        c.advanceNumRequests();
                  c.advanceSiteNumRequests();
                        c.advanceHits();
                  c.advanceSiteNumHits();
                  c.setinCache(true);

                  double requestDate;

                  requestDate = ce.getLastAccess();
```

```
                        c.setLastAccess(requestDate);
                        RequestData r=new RequestData(requestDate);
                                c.addRequest(r);
CheckIfStale( (cachentry)cachetable.get(ce.getUrl()),"source_traces_odds.txt" );
                        }
                        if (c.isDeleted())
                        { // Re-enter entry in cache...
                                c.setDeleted(false);
                                c.advanceNumRequests();

                        double requestDate;
                                requestDate = ce.getLastAccess();
                        c.setLastAccess(requestDate);
                        RequestData r=new RequestData(requestDate);
                                c.addRequest(r);
                        }
                 }
                 else
                 {//create a new cache entry
                        System.out.println("Creating new Cache Entry");
                double requestDate;
                        requestDate = ce.getLastAccess();

                        cachentry c=new cachentry(ce.getUrl(),ce.getSize(),requestDate);
                        c.setLastAccess(requestDate);
                        RequestData r=new RequestData(requestDate);
CalcExpires(c,"source_traces_odds.txt");
                c.addRequest(r);
                        cachetable.put(url,c);
                        advanceSize(ce.getSize());
            }
            countSiteNumRequests(url);
            countSiteNumHits(url);
            advanceTotalNumRequests();
                 cleanup(this);

System.out.println("Site:"+ce.getUrl().getSite()+", Item:"+ce.getUrl().getItem()+"\n"
      +"Last Modified at: "+( (cachentry)cachetable.get(ce.getUrl()) ).getLastModified()
      +"\n"+"Created at: "+( (cachentry)cachetable.get(ce.getUrl()) ).getCreated()
      + "\n"+"Last Accessed at: "+( (cachentry)cachetable.get(ce.getUrl()) ).getLastAccess()
      + "\n"+"Expires at: "+( (cachentry)cachetable.get(ce.getUrl()) ).getExpires() );

System.out.println("hits so far: "+hits+"\nstale so far: "+stale);

if ( ( ( (cachentry)cachetable.get(ce.getUrl()) ).getExpires() )!=0) {totalrequests++;}
System.out.println("Total requests so far: "+totalrequests);

System.out.println("\nMethod 'handleEntry' terminated");
```

```
CalcFunctionU(this,ce.getUrl());
        }

    //--------------
  public entryUrl getLRUEntryKey()
  {
                double min = 14000000.000;
          entryUrl keyvalue=null;
                entryUrl currentkey=null;

                Enumeration e=cachetable.keys();

                while (e.hasMoreElements())
                {
                        currentkey=(entryUrl)e.nextElement();
                if (min >(((cachentry)cachetable.get(currentkey)).getLastAccess()))
                        {
                                keyvalue=currentkey;
                                min=((cachentry)cachetable.get(currentkey)).getLastAccess();
                        }
                        //System.out.println("key="+currentkey);
                        //System.out.println("Last                    access:"                    +
((cachentry)cachetable.get(currentkey)).getLastAccess().getTime());
                }
                return ((cachentry)cachetable.get(keyvalue)).getUrl();
        }

        /**
                Searches for a url in the cachtable
                @param s The url we are searching for
                @return true if it is found, or false if it is not found
         */
        public boolean findUrl(String s)
        {
                return cachetable.containsKey(s);
        }

        public cachentry getEntry(entryUrl urlname)
        {
                return (cachentry)cachetable.get(urlname);
        }

        public synchronized void clear()
        {
                cachetable.clear();
        }

        public int getSize()
        {
```

```
                    return cachetable.size();
                }

            public void countSiteNumHits(entryUrl url)
            {
                Enumeration keys = cachetable.keys();
                while( keys.hasMoreElements() ) {
                entryUrl key = (entryUrl) keys.nextElement();
                  if ( ( key.getSite() ).equals(url.getSite()) && !( ( key.getItem() ).equals(url.getItem()) ) ){
if ( ((cachentry) cachetable.get(key)).inCache() && !( ((cachentry) cachetable.get(url)).inCache() ) ) {
((cachentry) cachetable.get(url)).setSiteNumHits(((cachentry) cachetable.get(key)).getSiteNumHits());
                    //System.out.println(key.getSite()+"."+key.getItem()+"                    "+url.getSite()+"
    "+url.getItem()+"\n");
                    //long k = ((cachentry) cachetable.get(url)).getSiteNumHits();
                    //long l = ((cachentry) cachetable.get(key)).getSiteNumHits();
                    //System.out.println(k+" "+l);
                    }
                    else if ( ((cachentry) cachetable.get(url)).inCache() && !( ((cachentry)
    cachetable.get(key)).inCache() ) ) {
    ((cachentry)                                    cachetable.get(key)).setSiteNumHits(((cachentry)
    cachetable.get(url)).getSiteNumHits());
                    //System.out.println(key.getSite()+"."+key.getItem()+"                    "+url.getSite()+"
    "+url.getItem()+"\n");
                    //long k = ((cachentry) cachetable.get(url)).getSiteNumHits();
                    //long l = ((cachentry) cachetable.get(key)).getSiteNumHits();
                    //System.out.println(k+" "+l);
                    }
                    else if ( ((cachentry) cachetable.get(key)).inCache() && ((cachentry)
    cachetable.get(url)).inCache() ) {
    ((cachentry)                                    cachetable.get(key)).setSiteNumHits(((cachentry)
    cachetable.get(url)).getSiteNumHits());
                    //System.out.println(key.getSite()+"."+key.getItem()+"                    "+url.getSite()+"
    "+url.getItem()+"\n");
                    //long k = ((cachentry) cachetable.get(url)).getSiteNumHits();
                    //long l = ((cachentry) cachetable.get(key)).getSiteNumHits();
                    //System.out.println(k+" "+l);
                    }
                  }
                }
            }

            public void countSiteNumRequests(entryUrl url)
            {
                Enumeration keys = cachetable.keys();
                while( keys.hasMoreElements() ) {
                entryUrl key = (entryUrl) keys.nextElement();
                  if ( ( key.getSite() ).equals(url.getSite()) && !( ( key.getItem() ).equals(url.getItem()) ) ){
                    ((cachentry) cachetable.get(key)).advanceSiteNumRequests();
```

```
((cachentry) cachetable.get(url)).setSiteNumRequests(((cachentry) cachetable.get(key)).getSiteNumRequests());
                //  System.out.println(key.getSite()+"."+key.getItem()+"  "+url.getSite()+" "+url.getItem()+"\n");
                //  long k = ((cachentry) cachetable.get(url)).getSiteNumRequests();
                //  long l = ((cachentry) cachetable.get(key)).getSiteNumRequests();
                //  System.out.println(k+" "+l);
              }
            }
        }

        public synchronized void CalcExpires(cachentry c, String input)
            {
          double g = 0;
          System.out.println(" Calculating Expires Time...");
          try {
            //check source file
            FileReader infile = null;
            String line = "";
            infile = new FileReader(input);
            BufferedReader b = new BufferedReader(infile);
            line = b.readLine();

            while ( (line = b.readLine()) != null) {
               String[] tmp = line.split(" ");
               if ( (c.getUrl()).getSite().equals(tmp[0]) && (c.getUrl()).getItem().equals(tmp[2]) ) {
                  if ( Double.parseDouble(tmp[3])>c.getCreated() )
                  {
                     break;
                  }
                  else
                  {
                  c.setLastModified(Double.parseDouble(tmp[3]));
                  c.setExpires( Double.parseDouble(tmp[4]));
                  c.setExpirationTime(Double.parseDouble(tmp[5]));
                  }
               }
            }
                  if (c.getExpires() == -1){
                     c.setLastModified(0);
                     c.setExpirationTime(0);
                     }
          System.out.println("  Expiration Time is: "+c.getExpirationTime()+"\n  Expires at: "+c.getExpires()
                        +"\n Finished Calculating Expire Time...");
          }
          catch (IOException ex) {
            Logger.getLogger(CacheHTable.class.getName()).log(Level.SEVERE, null, ex);
      }
```

```
}

    public double CalcAvgExpTime(entryUrl url)
    {
       double g=0,avg =0;
       int count=0;

       Enumeration keys = cachetable.keys();
       while( keys.hasMoreElements() ) {
       entryUrl key = (entryUrl) keys.nextElement();
       if ( ( key.getSite() ).equals(url.getSite()) )
       {
          if( ((cachentry) cachetable.get(key)).getExpires() != -1.0) {
             g += ( (cachentry) cachetable.get(key)).getExpirationTime();
             count++;}
       }

       avg = g/count;}
       return avg;
    }

        public void print ()
        {
                Enumeration e=cachetable.keys();
                entryUrl currentkey;

                while (e.hasMoreElements())
                {
                        currentkey=(entryUrl)e.nextElement();
                        System.out.println(cachetable.get(currentkey));
                        System.out.println("--------------------");
                }
        }

    public long getTotalNumRequests(){return totalNumRequests;}

    public double getCalcFunctionU(){return calcfunctionU;}

    /**
       @return The max size in bytes of the Cache Directory
       */
       public long getMaxSize(){return MaxSize;}
    /**
       @return The current Cache Directory size in bytes of the Cache Directory
       */
       public long getCurrentSize(){return currentSize;}
/**
       Advances the current Cache Directory size by 'size' bytes. This happens when
       a new entry is added in cache
```

```
    @param size The amount of bytes to add in the current Cache Directory size
    */
    public synchronized void advCacheCurrentSize(long size){currentSize+=size;}

    /**
    Reduces the current Cache Directory size by 'size' bytes. This happens when
    an entry is removed from cache
    */
    public synchronized void redCacheCurrentSize(long size){currentSize-=size;}

      public synchronized void addentry(String urlname,cachentry h)
    {
            cachetable.put(urlname,h);
            // else throw MaxCacheException
      advCacheCurrentSize(h.getSize());
    }

      public synchronized void removeEntry(entryUrl urlname)
    {
            cachetable.remove(urlname);
    }


public synchronized void CheckIfStale(cachentry c, String input)
    {
   //if ( (c.getExpires()!=-1) ){
   if ( (c.getExpires()!=0) ){
     try {
   System.out.println("Checking if stale hit...");
   //check source file
   FileReader infile = null;
   String line = "";
   infile = new FileReader(input);
   BufferedReader b = new BufferedReader(infile);
   line = b.readLine();
   double temp_last_mod = 0;
   while ( (line = b.readLine()) != null) {
      String[] tmp = line.split(" ");
      if ( (c.getUrl()).getSite().equals(tmp[0]) && (c.getUrl()).getItem().equals(tmp[2]) ){
         if (Double.parseDouble(tmp[3])>c.getLastModified() &&
             Double.parseDouble(tmp[3])<c.getLastAccess()) {
       stale++;
       if (temp_last_mod !=0) {stale--;}
       temp_last_mod = Double.parseDouble(tmp[3]);
       System.out.println("Stale! Modified at "+tmp[3]);
      } else if (Double.parseDouble(tmp[3])>c.getLastAccess()) {return;}
      }
   }
}
```

```
        catch (IOException ex) {
          Logger.getLogger(CacheHTable.class.getName()).log(Level.SEVERE, null, ex);
    } }
    }

    private static boolean CheckSourceFile (cachentry c, String input, double time1, double
time2) {
      boolean changed_in_source=false;

      try {
         //check source file
         System.out.println("Checking if item has changed...");
         FileReader infile = null;
         String line = "";
         infile = new FileReader(input);
         BufferedReader b = new BufferedReader(infile);
         line = b.readLine();

         while ( (line = b.readLine()) != null) {
            String[] tmp = line.split(" ");
            if ( Double.parseDouble(tmp[3])<time1 && Double.parseDouble(tmp[3])>time2
                 && (tmp[0].equals(c.getUrl().getSite()) && tmp[2].equals(c.getUrl().getItem())) )
            {
            System.out.println("Item has already changed in the source...:"+tmp[3]);
            changed_in_source = true;
            c.setLastModified(Double.parseDouble(tmp[3]));
            c.setExpires(Double.parseDouble(tmp[4]));
            c.setExpirationTime(Double.parseDouble(tmp[5]));
            }
            else if (Double.parseDouble(tmp[3])>time1 && (changed_in_source == false) )
            {
            System.out.println("Item has not changed in the source");
            break;
            }
         }
     } catch (IOException ex) {
         Logger.getLogger(CacheHTable.class.getName()).log(Level.SEVERE, null, ex);
     }
   return changed_in_source;
 }

public synchronized void CalcFunctionU(CacheHTable cachet, entryUrl url)
   {
   int num_updated_in_source = 0, num_same_in_source = 0;
   double g = 0;
   double runtime = 0;

      if (cachet.getEntry(url).inCache()==true)
         {
```

```
        runtime = cachet.getEntry(url).getLastAccess();
        }
        else if (cachet.getEntry(url).inCache()==false)
          {
          runtime = cachet.getEntry(url).getCreated();
          }

      if ( runtime > cachet.getCalcFunctionU()) {

                        System.out.println("-------    Method 'CalcFunctionU' invoked ");
                System.out.println("Runtime at: "+runtime+"\n");

                Enumeration e=cachetable.keys();

                while (e.hasMoreElements())
                {
                //count times U is calculated
                num_items_calc_U++;

                entryUrl currentkey1 =(entryUrl)e.nextElement();
 double U;
 double ex=0;
                double k = (double) ( ((cachentry)cachetable.get(currentkey1)
).getSiteNumRequests()) / ( cachet.getTotalNumRequests() );
                double h = (double) ( ((cachentry)cachetable.get(currentkey1) ).getSiteNumHits())
/ ( ((cachentry)cachetable.get(currentkey1) ).getSiteNumRequests()) ;
                double avgexptime = CalcAvgExpTime(currentkey1);
                double r = ( runtime - ((cachentry)cachetable.get(currentkey1) ).getCreated() ) /
avgexptime;
/*              //If item has already expired at the time function U is calculated,
              //set r equal to 1 (the maximum contribution of r). When the item
              //has expired the neccesity to update is 1.
*/                  if (r> 1){r=1.0;}

                if(((cachentry)cachetable.get(currentkey1) ).getExpires() != -1){
                ex = 1 - ( ((cachentry)cachetable.get(currentkey1) ).getExpires() - runtime)
    /(              ((cachentry)cachetable.get(currentkey1)              ).getExpires()              -
((cachentry)cachetable.get(currentkey1) ).getCreated() );
/*              //If item has already expired at the time function U is calculated,
              //set ex equal to 1 (the maximum contribution of ex). When the item
              //has expired the nessecity to update is 1.
                if ( ((cachentry)cachetable.get(currentkey1) ).getExpires() < runtime ) ex=1.0;
*/
                if (runtime> ((cachentry)cachetable.get(currentkey1) ).getExpires() ) {ex=1;}
                U = 0.25*(h+k+r+ex);
                  }
                else U = 0.33*(h+k+r);

                U = U * 10000;
```

```
                U = Math.round(U);
                U = U / 10000;
                ((cachentry)cachetable.get(currentkey1) ).getListFunctionU().add( U );

                System.out.print("-//----//----//----//-\n"+"Site: "+currentkey1.getSite()+", Item: "
                      +currentkey1.getItem()+"\n"+"Site requests so far: "
                      +((cachentry)cachetable.get(currentkey1) ).getSiteNumRequests() +"\n"
             +"Site hits so far: "+((cachentry)cachetable.get(currentkey1) ).getSiteNumHits() +"\n"
             +"Total requests so far: "+cachet.getTotalNumRequests()+"\n"+"k="+ k + ", h="+ h+",
r="+r+", 3="+ex);

   System.out.println("\nAvg Expire Time="+avgexptime+", Expires
at="+((cachentry)cachetable.get(currentkey1)).getExpires()
             +", Runtime="+runtime+"\nItem Hits="+((cachentry)cachetable.get(currentkey1)
).getHits()
             +"\nList of U function: "+ ((cachentry)cachetable.get(currentkey1)
).getListFunctionU() );

              if (((cachentry)cachetable.get(currentkey1) ).getListFunctionU().size() > 6)
              {
               //count times odds alorithm runs
               Num_Items_tested++;
               double sum =0;
     sum = OddsAlgorithm ( 1,((cachentry)cachetable.get(currentkey1)).getListFunctionU(),0.5 );
                  if (sum == 0.0)
                  {
                  System.out.println("Need more records...");
                  ((cachentry)cachetable.get(currentkey1) ).getListFunctionU().clear();
                  }
                  else if (sum < 1.0)
                  {
          boolean updated_in_source1 = CheckSourceFile
((cachentry)cachetable.get(currentkey1), "source_traces_odds.txt",
                     runtime, ((cachentry)cachetable.get(currentkey1)).getLastModified());
          if ( updated_in_source1 == true )
                     num_updated_in_source++;//count changed items in source
          else num_same_in_source++;//count items not changed in source

                  ((cachentry)cachetable.get(currentkey1) ).getListFunctionU().clear();
                  }
              }
          }

          System.out.println("-------  Method 'CalcFunctionU' terminated ");

          cachet.advanceCalcFunctionU();
          allUpdated += num_updated_in_source;
          allSame += num_same_in_source;
       }
```

```
  }

public synchronized void cleanup(CacheHTable cachet)
        {
          System.out.println("    Method 'cleanUp' invoked ");
                System.out.println("       Current size is:" + this.currentSize);
                System.out.println("       Maximum size is:" + this.MaxSize);

          if (getCurrentSize()<(0.8)*getMaxSize()) {
                        System.out.println("    Method 'cleanUp' terminated ");
              return;}

                System.out.println("Cache reached 80% of MaximumSize. Cleaning up ...");
                while (getCurrentSize()>((0.6)*getMaxSize()))
                {
                        entryUrl victimKey=cachet.getLRUEntryKey();
                        long filesize=( (cachet.getEntry(victimKey) ).getSize() );
                                redCacheCurrentSize(filesize);
                                cachet.removeEntry(victimKey);
                                System.out.println("Removed: " + "Site " + victimKey.getSite() +
", Item " + victimKey.getItem());
                }
          System.out.println("Method 'cleanUp' terminated");

        }

public void advanceSize(long s)
        {
                currentSize+=s;
        }

public void advanceTotalSize(long num)
        {
                totalSize+=num;
        }

public void advanceTotalNumRequests()
        {
                totalNumRequests+=1;
        }

public void advanceCalcFunctionU()
        {
          calcfunctionU = calcfunctionU + 1000.000;
      }

   public boolean containsUrl(entryUrl url)
        {
                return cachetable.containsKey(url);
```

```
        }

    public boolean isValid(cachentry c,long size)
        {
                return (size==c.getSize());
        }

private static double OddsAlgorithm (int sd, ArrayList<Double> list, double lthresh) {
  //Static Parameters from Odd Algorithm project
  int s;
  double val, sum=0;
  String lkfile;
  //-----------------------------------
    int success, k;
    double c, N;
    float p = 0;
    s = sd;
    boolean cont = true;

  double fk [] = new double [list.size()];
  double [] lk = new double [ list.size() ];

  for (int j=0; j < list.size(); j++) {
        lk[j] = list.get(j);
  }

  //--------- Odds Algorithm with sequential estimation ----------------
  for (int v = 0; v < fk.length; v++){
        fk [v] = (double) 1.00/(double) (v+1);
  }

  while (cont) {
        success = 0 ;
        c = 0;
        double temp = lk[0];
        for (int i = 0; i < s; i++) {
           if ( (lk[ i ])  > lthresh ) success ++;
           c += fk[i];
        }

        p = (float) success/ (float) c;
        sum=0;
        k = s+1;
        double [] pk = new double [list.size()-k+1];
        double [] qk = new double [list.size()-k+1];

        for (int u = 0; u < list.size()-k+1 ; u++) {
           pk [u] = p*fk[u+s];
           qk [u] = 1 - pk[u];
```

```
            sum += pk [u] / qk[u];
          }

          System.out.println("step " + s + " - sum = " + sum);

        if (sum<1 && sum!=0)
        {
           cont = false;
           if (Ik[s-1]>Ithresh) System.out.println("Stop at "+s+" step.");
           else {
              for (int w=s;w<list.size();w++)
                {
              if (Ik[w]>Ithresh) {System.out.println("Stop at "+(w+1)+" step.");break;}
              else if (w==list.size()-1) System.out.println("Stop at "+(w+1)+" step.");
                }
           }
        }
        else if (sum>=1 || sum==0){
           s++;
           if (s == list.size() ) cont = false;
        }

        if ( sd == list.size() ) {cont = false;s--;}

         val =  ( Ik[s-1] );

       } // end while

 return sum;

 } // end function OddsAlgorithm

 public long getAllUpdated()
   {return allUpdated;}

 public long getAllSame()
   {return allSame;    }

  public long getNumItemsTested()
   {return Num_Items_tested;    }

   public int getStale()
   {return stale;    }

public long getNumItemsCalcU()
   {return num_items_calc_U;}

}
```