ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αρχιτεκτονικές Caching για δυναμικό περιεχόμενο στον Παγκόσμιο Ιστό

Αντώνης Ι. Παπαδημητρίου

Επιβλέπων: Ευστάθιος Χατζηευθυμιάδης, Επίκουρος Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ
ΔΕΚΕΜΒΡΙΟΣ 2005

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αρχιτεκτονικές Caching για δυναμικό περιεχόμενο στον Παγκόσμιο Ιστό

Αντώνης Ι. Παπαδημητρίου

Α.Μ.:018200100119

ΕΠΙΒΛΕΠΩΝ:
Ευστάθιος Χατζηευθυμιάδης, Επίκουρος Καθηγητής ΕΚΠΑ

ΠΕΡΙΛΗΨΗ

Σε αυτή την πτυχιακή εργασία εξετάζονται τα υπάρχοντα δεδομένα σχετικά με το web caching δυναμικού περιεχομένου. Κατά την μελέτη των τεχνικών που έχουν εμφανιστεί ως τώρα γίνεται διαχωρισμός σε δύο κατηγορίες, των τεχνικών που ανήκουν στη δυναμική συναρμολόγηση σελίδων και εκείνων που κάνουν χρήση κάποιας μορφής κωδικοποίησης διαφορών (Delta Encoding). Σκοπός της εργασίας είναι μετά την παρουσίαση των αρχιτεκτονικών που έχουν προταθεί μέχρι σήμερα να γίνει η περιγραφή μιας νέας αρχιτεκτονικής δυναμικού caching που βασίζεται στην κωδικοποίηση διαφορών. Η ιδέα είναι ο συνδιασμός της κωδικοποίησης διαφορών με την ενημέρωση των caches από τον εξυπηρέτη διαδικτύου, σε ένα σχήμα προωθητικού caching (push caching). Στα πλαίσια της νέας πρότασης διερευνόνται θεωρητικά αλλά και πειραματικά τα ωφέλη, τα μειονεκτήματα αλλά και τυχόντες επιλογές για την μελλοντική μετεξέλιξη του σχήματος.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: εφαρμογές διαδικτύου

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: web caching, κωδικοποίηση διαφορών, προωθητικό caching, δυναμικές ιστοσελίδες, κλιμάκωση εφαρμογών διαδικτύου

ΠΕΡΙΕΧΟΜΕΝΑ

ΜΕΡΟΣ Ι : Web Caching

1.1 - Εισαγωγή

Αντικείμενο αυτής της εργασίας είναι οι αρχιτεκτονικές caching για δυναμικό περιεχόμενο στον παγκόσμιο ιστό. Για να γίνουν όμως αντιληπτές οι έννοιες που πρόκειται να περιγραφούν, είναι απαραίτητο να γίνει μια στοιχειώδης εισαγωγή στο θέμα του web caching, που αποτελεί τη γενίκευση και την αφετηρία για επεκτάσεις που αφορούν το δυναμικό περιεχόμενο. Στο πρώτο μέρος, λοιπόν, της εργασίας αυτής θα γίνει μια πολύ συνοπτική παρουσίαση του ζητήματος του caching στον παγκόσμιο ιστό, ώστε να δωθεί στον αναγνώστη το αναγκαίο υπόβαθρο για την συνέχεια.

Το web caching είναι μια ορολογία που υπάρχει αρκετά χρόνια στον παγκόσμιο ιστό (World Wide Web) και μάλιστα έχει συνεισφέρει τα μέγιστα προς την πορεία της τόσο ραγδαίας ανάπτυξής του.

Ένας απλός χρήστης του διαδικτύου (client), όταν περιηγείται στο δίκτυο έχει την εντύπωση πως απλά ζητάει και βλέπει κάποιες σελίδες από ένα απομακρυσμένο σύστημα (server), το σύστημα που φιλοξενεί την εφαρμογή. Αυτό είναι η κοινή διαίσθηση που έχει οποιοσδήποτε για το WWW (υπηρεσίες που τρέχουν πάνω από το πρωτόκολλο HTTP). Η αλήθεια όμως είναι διαφορετική, καθώς πίσω από αυτή την απλοϊκή διαδικασία λαμβάνουν χώρα μια σειρά από πολύπλοκες λειτουργίες, με τη συμμετοχή πολλών διαφορετικών, απομακρυσμένων και εταιρογενών υπολογιστικών συστημάτων.

Οι HTTP αιτήσεις που αποστέλλει ο πελάτης στον εξυπηρέτη δεν ταξιδεύουν κατ'ευθείαν από το ένα άκρο επικοινωνίας στο άλλο, αλλά περνούν μέσα από ενδιάμεσους υπολογιστές, οι οποίοι έχουν την υποχρέωση να δρομολογήσουν τις αιτήσεις προς την κατάλληλη κατεύθυνση. Επιπλέον, επειδή το WWW εξελίχθηκε γρήγορα σε ένα δίκτυο τεραστίων διαστάσεων, πολλοί από τους ενδιάμεσους αυτούς κόμβους κλήθηκαν να βοηθήσουν στην κλιμακωσιμότητα του παγκοσμίου ιστού, δηλαδή στη δυνατότητά του να εξυπηρετήσει πολύ μεγάλο αριθμό χρηστών. Για να γίνει αυτό, οι κόμβοι αυτοί μετατράπηκαν σε caching proxies, δηλαδή υπολογιστικά συστήματα που εφαρμόζουν τεχνικές caching για να εξυπηρετούν τις αιτήσεις των χρηστών, χωρίς αυτές να χρειαστεί να προωθηθούν έως τους εξυπηρέτες που προσφέρουν τις διαδικτυακές πληροφορίες.

Οι τεχνικές caching είναι αρκετά παλιές στην επιστήμη των υπολογιστών και υπονοούν την προσορινή αποθήκευση δεδομένων σε σημείο που είναι πιο γρήγορα προσβάσιμες προκειμένου να βελτιωθεί η απόδοση των διαφόρων συστημάτων. Χαρακτηριστικό παράδειγμα είναι το caching που εφαρμόζεται στην επικοινωνία μεταξύ του επεξεργαστή και της κύριας μνήμης. Επειδή η μνήμη είναι πολύ πιο αργή σε σχέση με την ταχύτητα επεξεργασίας της CPU, αποτελεί ένα στενωπό στη λειτουργία του όλου συστήματος ενός υπολογιστή. Για την καταπολέμηση αυτού, μερικά τμήματα της μνήμης που κρίνεται ότι χρησιμοποιούνται πιο συχνά από άλλα αποθηκεύονται στην μνήμη της cache που είναι πιο γρήγορη από την κύρια μνήμη, και με την οποία επικοινωνεί ο

επεξεργαστής, με αποτέλεσμα, όταν τα δεδομένα που ζητά η CPU βρίσκονται στην cache, να υπάρχει βελτίωση καθώς δεν καθυστερείται η λειτουργία από την αργή κύρια μνήμη.

Το ίδιο γίνεται και στον παγκόσμιο ιστό: οι απαντήσεις των αιτήσεων HTTP (συνήθως αρχεία HTML ή εικόνων) αποθηκεύονται σε ενδιάμεσους υπολογιστές που βρίσκονται πιο κοντά στους χρήστες του διαδικτύου, ώστε να μη χρειάζεται τα δεδομένα να ξαναταξιδέψουν από το server την επόμενη φορά που θα ζητηθούν από τον ίδιο ή διαφορετικό χρήστη.

Η βελτίωση της απόδοσης από αυτή την αρχιτεκτονική είναι θεαματική καθώς οι πόροι του διαδικτύου εμφανίζουν το χαρακτηριστικό να μην αλλάζουν πολύ συχνά, με αποτέλεσμα πολλές διαδοχικές αιτήσεις να εξυπηρετούνται από proxies που έχουν αποθηκεύσει τα εν λόγω δεδομένα. Μάλιστα, το web caching είναι αυτό που δίνει τη δυνατότητα να χρησιμοποιεί το WWW τόσο μεγάλος αριθμός χρηστών, αφού αν δεν υπήρχαν οι proxies το δίκτυο θα είχε πολύ πιο περιορισμένη χωρητικότητα.

1.2 – Η υποστήριξη του web caching από το HTTP

Το web caching υπήρξε από τις πρώτες μέρες ένα τόσο σημαντικό τμήμα του Παγκόσμιου Ιστού, ώστε από την πρώτη ακόμα έκδοση του πρωτοκόλλου HTTP/1.0 [Berners-Lee et al. 1996] ενσωματώθηκαν κάποιες επικεφαλίδες που λάμβαναν πρόνοια για τους μηχανισμούς προσωρινής αποθήκευσης στο διαδίκτυο. Πολύ περισσότερο, στην τρέχουσα έκδοση του πρωτοκόλλου [HTTP/1.1] το web caching υποστηρίζεται εγγενώς από το πρωτόκολλο και η παρουσία του είναι διάσπαρτη σε πολλά σημεία της λειτουργίας του πρωτοκόλλου.

Στη συνέχεια παρατίθενται συνοπτικά τα σημεία αυτά του πρωτοκόλλου που δείχνουν τον τρόπο που το HTTP πρωτόκολλο υποστηρίζει το web caching.

1.2.1 – Αιτήσεις υπό συνθήκη (Conditional Requests)

Στον πίνακα που ακολουθεί φαίνονται οι πιο σημαντικές επικεφαλίδες διακλάδωσης (conditional headers) που χρησιμοποιούνται στα πλαίσια του caching. Η if-modified-since περιέχει την ημερομηνία (και ώρα) που τροποποιήθηκε τελευταία (last-modified date) το αντικείμενο που αποθηκεύεται στην cache. Όταν ένας εξυπηρέτης δέχεται μια αίτηση με αυτή την επικεφαλίδα, επιστρέφει το αντικείμενο μόνο στην περίπτωση που η ημερομηνία last-modified date του αντικειμένου είναι διαφορετική από εκείνη που αναγράφεται στην επικεφαλίδα if-modified-since της αίτησης. Σε αντίθετη περίπτωση, επιστρέφει μια απάντηση με τον κωδικό (status code) “304 Not Modified”.

Για μεγαλύτερη ακρίβεια στον διαχωρισμό των αντικειμένων, στο HTTP/1.1 χρησιμοποιείται και η επικεφαλίδα if-none-match που περιέχει αντί για την last-modified date, την entity tag του αντικειμένου, μια τιμή κατακερματισμού που είναι μοναδική για κάθε αντικείμενο. Όμοια με παραπάνω, ο εξυπηρέτης

απαντά με το αντικείμενο μόνο αν αυτό έχει διαφορετική entity tag από εκείνη που περιέχει η επικεφαλίδα if-none-match.

Header Keyword	Value	Meaning
`if-modified-since`	last-modified date	execute request if requested object updated since last-modified date
`if-none-match`	`ETag`	execute request if `ETag` of requested object is different

1.2.2 – Γήρανση και Λήξη (Age and Expiration)

Μια άλλη πολύ βασική έννοια στο web caching και σε άλλους τομείς της πληροφορικής είναι η ένοια του χρόνου ζωής ενός πόρου TTL (Time To Live). Το TTL δηλώνει το χρόνο για τον οποίο ένα αντικείμενο αναμένεται να είναι έγκυρο και να μην έχει αλλάξει. Με βάση αυτή την εκτίμηση, που συνήθως παρέχουν οι servers που εξυπηρετούν τις σελίδες μιας εφαρμογής, μια cache μπορεί να επιστρέφει το αποθηκευμένο αντικείμενο σε κάθε client που αιτεί τον συγκεκριμένο πόρο μέσα σε αυτό το χρονικό διάστημα (TTL), με μικρή πιθανότητα παράδωσης ξεπερασμένου περιεχομένου.

Την υποστήριξη του TTL στο πρωτόκολλο HTTP παρέχουν οι επικεφαλίδες expires, max-age και age, των οποίων η χρήση περιγράφεται στον ακόλουθο πίνακα:

Header Keyword	Value	Meaning
`expires`	date	date until which the object is valid
`max-age`	seconds	maximum age the object may reach before validation
`age`	seconds	estimated time since the object was served or last validated by the origin server

1.2.3 – Cache-Control Header

Το αποκορύφωμα της υποστήριξης του web caching στο HTTP είναι η επικεφαλίδα cache-control που στην ουσία είναι η επικεφαλίδα στην οποία μπορεί να μπεί κάθε πληροφορία που μπορεί να χρησιμεύσει στο caching. Ανάλογα με το αν η επικεφαλίδα αυτή βρίσκεται σε αίτηση ή απάντηση, οι οδηγίες-κατευθύνσεις (directives) που περιέχει στηην τιμή της έχουν διαφορετικό νόημα, όπως φαίνεται από τους δύο πίνακες που ακολουθούν:

Για HTTP αιτήσεις:

Directive	Value	Meaning
`no-cache`	none	Cached objects cannot be used to satisfy the request.

Directive	Value	Meaning
`no-store`	none	The response to this request cannot be stored in a cache.
`max-age`	seconds	Only younger cached objects can be used to satisfy the request.
`min-fresh`	seconds	Only cached objects that will not expire for a specified time can be used.
`max-stale`	seconds	Cached objects that expired up to the specified time ago can be used.
`no-transform`	none	Only the precise response as provided by the origin server can be used.
`only-if-cached`	none	A proxy should not forward the request on a cache miss.

Για HTTP απαντήσεις:

Directive Keyword	Value	Meaning
`no-cache`	none	The response cannot be cached.
`no-store`	none	The response cannot be stored in any client (proxy or browser).
`private`	none	The response can be reused only for the client that originally requested it.
`public`	none	The response may be cached and shared among different clients.
`must-revalidate`	none	A cache must always validate the cached object before using it.
`proxy-revalidation`	none	Same as `must-revalidate` but applies to proxy caches only.
`max-age`	seconds	A cache must validate this object before serving it to any client once the object age reaches the specified value.
`s-maxage`	none	Same as `max-age` but applies only to proxies.
`no-transform`	none	Only the precise response as provided by the origin server can be used.

Η πλήρης επεξήγηση όλων των παραπάνω οδηγιών δεν είναι μέσα στους σκοπούς αυτής της εργασίας, ωστόσο, είναι ξεκάθαρο ότι το web caching διαδραματίζει τόσο μεγάλο ρόλο στον Παγκόσμιο Ιστό, που υποστηρίζεται σε μεγάλο βαθμό εγγενώς μέσα από το πρωτόκολλο HTTP.

1.3 – Η εγκατάσταση των Proxies

Ένα σημαντικό μέρος της κατανόησης της λειτουργίας του web caching και των proxies είναι η κατανόηση του που ακριβώς τοποθετούνται στο δίκτυο και ιδιαίτερα πως παρεμβαίνουν στην επικοινωνία μεταξύ του πελάτη και του εξυπηρέτη.

Μια cache ενός proxy server ανακόπτει τις HTTP αιτήσεις των clients και αν βρει το αιτούμενο αντικείμενο μέσα στην cache, επιστρέφει το αντικείμενο στο χρήστη. Εάν το αντικείμενο δεν βρεθεί , η cache πηγαίνει στον server ο οποίος έχει το αντικείμενο (origin server) και εκ μέρους του χρήστη, παίρνει το αντικείμενο πιθανώς το αποθηκεύει και στην δική της cache, και τέλος επιστρέφει το αντικείμενο στο χρήστη. Οι proxy caches συνήθως αναπτύσσονται στα άκρα ενός δικτύου ώστε να μπορούν να εξυπηρετήσουν έναν μεγάλο αριθμό χρηστών. Η χρήση των proxy caches τυπικά έχει ως αποτέλεσμα εξοικονόμηση εύρους ζώνης, βελτιωμένους χρόνους απαντήσεων και αυξημένη διαθεσιμότητα των στατικών αντικειμένων και δεδομένων του web. Στο σχήμα που ακολουθεί βλέπουμε διάφορες διαμορφώσεις από τις οποίες η πιο απλή είναι η (a), η οποία και ονομάζεται διαμόρφωση standalone proxy. Η διαμόρφωση αυτή είναι και η λιγότερο δημοφιλής, διότι η cache αντιπροσωπεύει ένα μοναδικό σημείο αποτυχίας δικτύου και έτσι όταν η cache δεν είναι διαθέσιμη εμφανίζεται και το δίκτυο ως μη διαθέσιμο προς τον χρήστη. Επίσης η προσέγγιση αυτή , απαιτεί ότι όλοι οι web browsers πρέπει χειρωνακτικά να διαμορφωθούν για να χρησιμοποιούν την κατάλληλη proxy cache. Έτσι, όταν ο server δεν είναι διαθέσιμος όλοι οι χρήστες πρέπει να επαναδιαμορφώσουν τους browsers τους, ώστε να χρησιμοποιούν μια άλλη cache. Η λύση σε αυτό το πρόβλημα δίνεται με την αυτοδιαμόρφωση του browser. Για τον εντοπισμό των κοντινών proxy caches έχει προταθεί το web proxy auto discovery protocol (WPAD) [Gauthier et al. 1998]. Ένα τελευταίο θέμα που έχει να κάνει με την standalone προσέγγιση, είναι αυτό της κλιμάκωσης. Όσο αυξάνει η ζήτηση, η μια cache πρέπει να συνεχίσει να χειρίζεται όλες τις αιτήσεις. Δεν υπάρχει τρόπος για να προστεθούν δυναμικά περισσότερες caches όταν είναι αναγκαίες, όπως είναι δυνατόν με το transparent proxy caching.

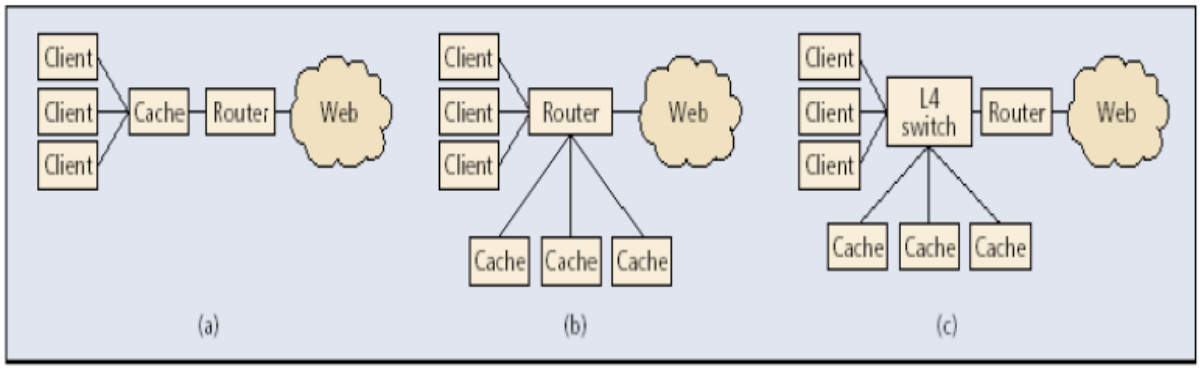
1.3.1 - Reverse Proxy Caching

Μια ενδιαφέρουσα προσέγγιση της αρχιτεκτονικής μιας proxy cache είναι η έννοια του reverse proxy caching, στην οποία οι caches αναπτύσσονται κοντά στον τόπο προορισμού του περιεχομένου σε αντίθεση με την πλευρά του client. Αυτή είναι μια ελκυστική λύση για servers ή domains που περιμένουν έναν μεγάλο αριθμό αιτήσεων από το internet ενώ ταυτόχρονα θέλουν να εξασφαλίσουν υψηλό επίπεδο ποιότητας υπηρεσιών. Το Reverse proxy caching είναι επίσης ένας χρήσιμος μηχανισμός για την υποστήριξη των web hosting farms που είναι μια συνεχώς αυξανόμενη υπηρεσία για πολλούς παροχείς υπηρεσιών του internet (ISP). Θα πρέπει να σημειωθεί ότι η ανάπτυξη του reverse proxy caching είναι εντελώς ανεξάρτητη από το proxy caching στην πλευρά του client. Στην πραγματικότητα μπορούν να συνυπάρχουν και συλλογικά να βελτιώνουν την απόδοση του web τόσο από την προοπτική του χρήστη, όσο και από αυτήν του δικτύου και του server.

1.3.2 - Transparent Caching

Το transparent proxy caching ελαχιστοποιεί ένα από τα μεγαλύτερα μειονεκτήματα στην προσέγγιση του proxy server: Την απαίτηση για διαμόρφωση των web browsers. Οι transparent caches ανακόπτουν τις HTTP αιτήσεις και τις ανακατευθύνουν σε web cache servers ή σε ένα σύμπλεγμα από caches. Αυτό το είδος του caching δημιουργεί ένα σημείο στο οποίο είναι πιθανά διαφορετικά είδη διοικητικού ελέγχου, για παράδειγμα η απόφαση πώς να κατανεμηθεί το ισοζύγιο φόρτου των αιτήσεων κατά μήκος πολλαπλών caches.
Η δύναμη του transparent cache είναι επίσης και η αδυναμία του: Παραβιάζει την end-to-end συμφωνία, μη διατηρώντας σταθερά τα τελικά σημεία της σύνδεσης. Αυτό είναι ένα πρόβλημα όταν μια εφαρμογή απαιτεί να διατηρείται το καθεστώς μέσω συνεχόμενων αιτήσεων ή κατά την διάρκεια που μια λογική αίτηση εμπλέκει πολλαπλά αντικείμενα.

Υπάρχουν δύο τρόποι για να αναπτυχθεί το transparent proxy caching : Στο επίπεδο διακόπτη (switch level) και στο επίπεδο δρομολογητή (router level). Στην δεύτερη περίπτωση χρησιμοποιείται μια πολιτική βασισμένη στην δρομολόγηση (σχήμα c) για να κατευθύνει τις αιτήσεις στην κατάλληλη cache(s). Για παράδειγμα, αιτήσεις από συγκεκριμένους clients μπορούν να συσχετιστούν με κάποια συγκεκριμένη cache.



Στην πρώτη περίπτωση (σχήμα b) ο διακόπτης ενεργεί ως ένας αποκλειστικός εξισορροπητής φόρτου. Οι διακόπτες γενικώς, είναι λιγότερο ακριβοί από τους δρομολογητές. Επίσης η λύση είναι περισσότερη ελκυστική διότι δεν υπάρχει επιπλέον φόρτος όπως αυτός που εισάγεται στην πολιτική που είναι βασισμένη στην δρομολόγηση.

1.4 – Συνεργατικό Proxy Caching

Βέβαια, μια και μόνο cache δεν έχει τους απαραίτητους υπολογιστικούς πόρους (μνήμη, υπολογιστική ισχυ) για να μπορέσει να βελτιώσει θεαματικά την απόδοση του αχανούς Παγκοσμίου Ιστού. Για το λόγο αυτό, οι caches οργανώνονται με τέτοιο τρόπο, ώστε να μπορούν με την μεταξύ τους συνεργασία να εξυπηρετήσουν όσο το δυνατόν μεγαλύτερο αριθμό χρηστών. Ακολουθεί μια περιγραφή των διαφόρων πρωτοκόλλων που χρησιμοποιούνται για την επικοινωνία και το συντονισμό των caches προς την εξυπηρέτηση του κοιινού καλού.

Ανάλογα με το πως οργανώνονται οι caches, μπορούν να δέχονται και να λαμβάνουν πληροφορίες για τους πόρους για τους οποίους ενδιαφέρονται. Αυτή η επικοινωνία είναι εξωτερική, ανεξάρτητη από τα request / response μηνύματα που ρέουν μεταξύ clients και origin servers. Η inter-cache επικοινωνία μπορεί να βασίζεται στο HTTP αλλά συνήθως χρησιμοποιεί εξειδικευμένα, light-weight πρωτόκολλα. Εάν ένα σύνολο από caches οργανώνεται σε ιεραρχία, μία cache μπορεί να επικοινωνήσει με τις υπόλοιπες caches στο ίδιο επίπεδο να διαπιστώσει εάν ο ζητούμενος πόρος είναι διαθέσιμος σε αυτές. Ένα ερώτημα για κάποιο πόρο μπορεί να απαντηθεί από μία ή περισσότερες caches που τυχαίνει να έχουν τον πόρο.

Συχνά, η ανάκτηση ενός πόρου από μία τοπική cache είναι προτιμότερη από την ανάκτηση από τον origin server. Η αναμονή για την απάντηση από όλες τις caches στην ιεραρχία μπορεί να αυξήσει σημαντικά την υστέρηση στον χρήστη. Για την υποβοήθηση της επικοινωνίας μεταξύ των caches εξετάζονται τέσσερα πρωτόκολλα: τα Internet Cache Protocol (ICP), Cache Array Resolution Protocol (CARP), Cache Digest Protocol (CDP) και Web Cache Coordination Protocol (WCCP).

1.4.1 - Internet Cache Protocol (ICP)

Μία cache που δεν διαθέτει τον ζητούμενο πόρο μπορεί να θέλει να ελέγξει την διαθεσιμότητα του πόρου σε μία άλλη γειτονική cache. Αυτή η επικοινωνία είναι διαφορετική από την παραδοσιακή αίτηση για ένα πόρο από τον origin server. Σε αυτή την περίπτωση οι caches αποτελούν την πηγή καθώς και τον προορισμό των μηνυμάτων που ανταλλάσονται. Ένα διαφορετικό πρωτόκολλο απαιτείται για την επικοινωνία μεταξύ των caches. Ένα από τα 13 πρώτα πρωτόκολλα που καθιερώθηκαν σε αυτήν την επικοινωνία είναι το Internet Cache Protocol (ICP). Το ICP είναι ένα πρωτόκολλο ερωταποκρίσεων. Το μήνυμα που στέλνεται από μία client cache είναι μία ερώτηση για το αν ο ομότιμος κόμβος έχει ένα αντίγραφο από τον πόρο που χρειάζεται η συγκεκριμένη cache. Ο δημοφιλής χαρακτήρας του ICP οφείλεται στο γεγονός ότι χρησιμοποιείται από το Squid.

Το ICP χρησιμοποιείται σε ιεραρχίες από caches, σύνολα caches που συνδέονται μεταξύ τους και κάτω από ένα κοινό γονέα. Η διαδικασία αυτή επαναλαμβάνεται και η μετακίνηση προς τα ανώτερα επίπεδα της ιεραρχίας σημαίνει μετακίνηση προς μία περισσότερο κεντρική cache. Οι κεντρικές caches μπορεί να έχουν μια περιφερειακή (regional) cache ως άμεσο πρόγονο ενώ οι περιφερειακές cache μπορεί να έχουν μία εθνική cache ως πρόγονο. Αν υποτεθεί ότι η cache, Original Cache δεν έχει κάποιο ζητούμενο πόρο, θα σταλούν ICP αιτήσεις σε όλους τους ομότιμους κόμβους ταυτόχρονα. Εάν κάποιος από τους ομότιμους κόμβους διαθέτει τον αιτούμενο πόρο, η Original Cache θα ενημερωθεί σχετικά και θα ζητήσει την ανάκτηση του χρησιμοποιώντας HTTP. Εάν κανείς από τους ομότιμους κόμβους δεν διαθέτει τον πόρο, η Original Cache θα προωθήσει την αίτηση στον γονέα της. Ο γονέας της Original Cache επαναλαμβάνει την διαδικασία. Εάν καμία από τις caches δεν διαθέτει τον πόρο, η Original Cache θα πρέπει να προωθήσει την αίτηση στον origin server. Η φιλοσοφία βάσει της οποίας λειτουργεί το ICP είναι ότι η αποστολή των ICP queries, ακόμη και αν αυτή επαναληφθεί πολλές

φορές σε διάφορα επίπεδα της ιεραρχίας, είναι σημαντικά γρηγορότερη την επικοινωνία με τον Origin Server.

1.4.2 - Cache Array Resolution Protocol (CARP)

Το CARP καθορίζει ένα μηχανισμό μέσω του οποίου ένα σύνολο από caching proxies μπορούν να λειτουργήσουν ως μία λογικά ενιαία cache. Ο μηχανισμός χειρίζεται το σύνολο των απαντήσεων που γίνονται cached συλλογικά μεταξύ της ομάδας (array) των proxies ως μία μεγάλη cache. Μία συνάρτηση κατακερματισμού του κλειδιού (hash function) χρησιμοποιείται για να διαιρεθεί το σύνολο των URL μεταξύ των caches. Ένας client που προσπαθεί να εντοπίσει ένα cached πόρο, μπορεί να κατευθύνει την αίτηση στην κατάλληλη cache, εφαρμόζοντας την hash function. Η hash function χρησιμοποιεί το αιτούμενο URL καθώς και την ταυτότητα των μελών του proxy για να διαμορφώσει ένα resolution path (μονοπάτι επίλυσης). Εάν συγκριθεί με το ICP, το CARP έχει ντετερμινιστικό μονοπάτι επίλυσης της αίτησης, εξαλείφοντας έτσι την ανάγκη για μηνύματα ερωταποκρίσεων (queries). Επίσης, υπάρχουν λιγότερα διπλότυπα αποθηκευμένων πόρων στο CARP από το ICP. Το CARP χρησιμοποιεί το HTTP καθώς και Remote Procedure Calls για την επικοινωνία μεταξύ των proxies. Ο κάθε proxy συσχετίζεται με ένα παράγοντα φορτίου (load factor) που λαμβάνεται υπόψη πριν μία αίτηση οδηγηθεί σε ένα συγκεκριμένο proxy. Το CARP διατίθεται ως προϊόν από την Microsoft.

1.4.3 - Cache Digest Protocol (CDP)

Το Cache Digest Protocol αποτελεί μία επέκταση του ICP. Η βασική ιδέα στο Cache Digest είναι η δυνατότητα ανταλλαγής μίας περίληψης (digest) των περιεχομένων της cache. Η περίληψη αποτελεί μία ένδειξη της συλλογής των αντικειμένων σε μία cache. Όταν μία cache έχει στην διάθεση της μία περίληψη από όλους τους ομότιμους κόμβους μπορεί, πολύ εύκολα να ανατρέξει στην περίληψη και να εξετάσει εάν το αιτούμενο αντικείμενο είναι διαθέσιμο σε μία από τις caches. Εάν η διερεύνηση αυτή επιτύχει, ο συγκεκριμένος ομότιμος κόμβος είναι υποψήφιος να δεχθεί μία αίτηση ανάκτησης του πόρου. Εάν ο έλεγχος στις περιλήψεις αποτύχει, οι αντίστοιχες caches δεν ερωτούνται, με προφανές αποτέλεσμα την δραστική μείωση των ερωτημάτων που απευθύνονται στο σύνολο των ομότιμων κόμβων.

Ένα πρόβλημα του μηχανισμού Cache Digest είναι η «παλαιότητα» των περιλήψεων (δεν έχουν ενημερωθεί) και τα λανθασμένα ερωτήματα τα οποία οφείλονται σε αυτήν. Για παράδειγμα, ένα αντικείμενο μπορεί να αφαιρεθεί από μία cache μετά την διαμόρφωση της σχετικής περίληψης. Ένα ακόμη πρόβλημα είναι το μέγεθος των περιλήψεων και η ανταλλαγή τους μεταξύ των ομότιμων κόμβων. Οι περιλήψεις ανταλλάσσονται μέσα σε HTTP μηνύματα, πάνω από το TCP, για λόγους αξιοπιστίας. Μία περίληψη μπορεί να θεωρηθεί σαν ένας κανονικός πόρος και οι τεχνικές ελέγχου της ορθότητας του HTTP (resource revalidation) μπορούν να χρησιμοποιηθούν για την διερεύνηση του επίκαιρου της περίληψης.

1.4.4 - Web Cache Coordination Protocol (WCCP)

Το WCCP είναι ένας μηχανισμός συντονισμού, στενά δεμένος με το επίπεδο δικτύου. Ο σκοπός του WCCP είναι η παρεμπόδιση (intercept) της HTTP αίτησης και η ανακατεύθυνση της στην μηχανή cache. Επειδή η αίτηση θα αποτύχει εάν η cache δεν είναι για κάποιο λόγο, διαθέσιμη, ένας μηχανισμός συντονισμού απαιτείται. Το αντικείμενο του μηχανισμού συντονισμού, είναι να εξισορροπεί τον φόρτο μεταξύ διαφορετικών caches, έχοντας πλήρη γνώση της διαθεσιμότητας τους. Ελέγχοντας περιοδικά τη διαθεσιμότητα μιας cache, ο μηχανισμός εξασφαλίζει ότι δεν πρόκειται να προωθηθούν πακέτα σε μία cache που δεν ανταποκρίνεται σε έλεγχο διαθεσιμότητας (hearbeat check). Ένας τέτοιος μηχανισμός αποτελεί την βάση του πρωτοκόλλου WCCP που υλοποιείται σαν τμήμα της Cisco Cache Engine. Η μηχανή cache ρυθμίζεται ώστε να δέχεται WWW αιτήσεις που ανακατευθύνονται σε αυτήν από ένα δρομολογητή. Ο δρομολογητής, που έχει ενεργοποιημένο το WCCP, μπορεί να επεξεργάζεται όλες τις IP επικεφαλίδες. Ένα TCP πακέτο που στοχεύει στην θύρα 80 ανακατευθύνεται στην cache engine. Επιπλέον, οι δρομολογητές που διαθέτουν WCCP, επικοινωνούν περιοδικά με τις μηχανές caching για να εξασφαλίσουν την διαθεσιμότητα τους.

1.5 – Συνέπεια/Συνάφεια της Cache (Cache Consistency)

Οι caches παρέχουν χαμηλότερη καθυστέρηση πρόσβασης αλλά με μια παρενέργεια: κάθε cache μερικές φορές παρέχει στους χρήστες «μπαγιάτικες» σελίδες (stale pages), δηλαδή σελίδες που είναι ξεπερασμένες σε σχέση με τις πρωτότυπες σελίδες που βρίσκονται στους Web servers. Κάθε Web cache πρέπει να ανανεώνει τις ιστοσελίδες που βρίσκονται σε αυτήν, ώστε να δίνει στους χρήστες ιστοσελίδες που είναι όσο το δυνατόν περισσότερο ενημερωμένες πρόσφατα. Το caching και το πρόβλημα της cache συνάφειας στο WWW είναι παρόμοιο με το πρόβλημα του caching στα κατανεμημένα συστήματα αρχείων. Παρόλα αυτά, το Web είναι διαφορετικό από τα συστήματα αυτά, στα πρότυπα πρόσβασης, στο ότι είναι μεγαλύτερης κλίμακας και στο χαρακτηριστικό τους σημείο, της ανανέωσης (αναβάθμισης) των αντικειμένων του Web.

Σε προηγούμενη παράγραφο περιγράφηκε πως υποστηρίζεται το web caching στο πρωτόκολλο HTTP. Με αυτές τις μεθόδους επιτυγχάνεται και η ως ένα βαθμό εξασφάλιση της συνέπειας των cached δεδομένων. Με τη χρήση αυτών των δυνατοτήτων του πρωτοκόλλου μπορούν να υλοποιηθούν από τους proxies διάφοροι μηχανισμοί συνάφειας, όπως περιγράφεται παρακάτω:

Τα υπάρχοντα σχήματα συνάφειας cache παρέχουν δυο τύπους συνάφειας για caches στο WWW :

1.5.1 - Strong cache consistency

Client validation. Αυτή η προσέγγιση καλείται επίσης και polling-every-time. Ο proxy συμπεριφέρεται στους πόρους που έχουν υποστεί cache ως πιθανώς ξεπερασμένους και μη έγκυρους σε κάθε πρόσβαση και στέλνει μια

επικεφαλίδα If-Modified-Since για κάθε πρόσβαση στους πόρους. Αυτή η προσέγγιση μπορεί να οδηγήσει σε πολλές απαντήσεις 304 (ο HTTP κώδικας απάντησης για το «Not Modified») από τον server αν ο πόρος δεν αλλάζει στην πραγματικότητα.

Server invalidation. Κατά την ανίχνευση μιας αλλαγής πόρου, ο server στέλνει μηνύματα ακύρωσης σε όλους τους χρήστες που τον έχουν πρόσφατα επισκεφτεί και πιθανώς τον έχουν κάνει cache. Αυτή η προσέγγιση απαιτεί έναν server για να κρατά μια λίστα από τους πελάτες. Η λίστα αυτή χρησιμοποιείται όταν πρέπει να ακυρωθούν αντίγραφα που έχουν γίνει cache, από πόρους που έχουν αλλάξει. Αυτή η διαδικασία είναι εξαιρετικά δύσχρηστη για έναν server, όταν ο αριθμός των χρηστών είναι πολύ μεγάλος. Επιπροσθέτως, οι λίστες από μόνες τους μπορούν να γίνουν έωλες προκαλώντας έτσι τον server να στέλνει μηνύματα ακύρωσης σε χρήστες οι οποίοι δεν έχουν πια στην cache τους τον πόρο.

1.5.2 - Weak cache consistency

Adaptive TTL (Time To Live). Το προσαρμοστικό TTL χειρίζεται το πρόβλημα, προσαρμόζοντας το Time To Live ενός εγγράφου βασισμένο σε παρατηρήσεις του χρόνου ζωής του. Το προσαρμοστικό TTL εκμεταλλεύεται το γεγονός ότι η διασπορά του χρόνου ζωής του αρχείου τείνει να γίνει δίμορφη. Εάν το αρχείο δεν έχει τροποποιηθεί για μεγάλο χρονικό διάστημα, τείνει να μείνει αμετάβλητο. Έτσι η ιδιότητα του Time To Live ενός εγγράφου έχει οριστεί να προσδιορίζει ένα ποσοστό από την τρέχουσα «ηλικία» του εγγράφου, η οποία είναι ο παρών χρόνος μείον τον χρόνο τελευταίας τροποποίησης του εγγράφου. Μελέτες έχουν δείξει [Gwetzman and Seltzer 1996] ότι το προσαρμοστικό TTL μπορεί να κρατήσει την πιθανότητα εγγράφων να είναι έωλα (stale) μέσα σε λογικά πλαίσια (<5%). Οι περισσότεροι proxy servers χρησιμοποιούν αυτό τον μηχανισμό. Παρόλο αυτά υπάρχουν διάφορα προβλήματα σχετικά με την συνάφεια που βασίζεται στην ημερομηνία λήξης (expiration). Πρώτον, ο χρήστης πρέπει να περιμένει να γίνουν οι έλεγχοι για την ημερομηνία λήξεως ακόμα και αν είναι ανεκτικός στο να περιέχονται στην αιτούμενη σελίδα, έωλα δεδομένα. Δεύτερον, εάν ο χρήστης δεν είναι ικανοποιημένος με την εωλότητα (staleness) του επιστρεφόμενου εγγράφου, δεν έχει άλλη επιλογή από το να χρησιμοποιήσει μια αίτηση Pragma: No-Cache για να φορτώσει ολόκληρο το έγγραφο από το home site του. Τρίτον, ο μηχανισμός δεν παρέχει ισχυρή εγγύηση ως προς την «φρεσκάδα» του εγγράφου. Τέταρτον, οι χρήστες δεν μπορούν να καθορίσουν τον βαθμό της εωλότητας (staleness) που είναι πρόθυμοι να ανεχθούν. Τέλος, όταν ο χρήστης εγκαταλείπει το φόρτωμα ενός εγγράφου, οι caches συχνά εγκαταλείπουν επίσης το φόρτωμα.

Piggyback invalidation. Ο Krishnamurthy και άλλοι, έχουν προτείνει το μηχανισμό του piggyback invalidation, για να βελτιώσουν την αποτελεσματικότητα της συνάφειας cache. Υπάρχουν τρεις κύριοι μηχανισμοί ακύρωσης που κατά καιρούς έχουν προταθεί. Ο piggyback cache validation (PCV) επικεντρώνεται στις αιτήσεις που έχουν σταλεί από την cache του proxy στον server για να βελτιωθεί η συνάφεια. Στην πιο απλή περίπτωση, όποτε μια proxy cache έχει λόγο για να επικοινωνήσει με έναν server, κάνει

piggyback μια λίστα από πόρους, που έχουν υποστεί cache αλλά είναι πιθανώς ξεπερασμένοι, από τον συγκεκριμένο server για αξιολόγηση. Η βασική ιδέα του μηχανισμού piggyback server invalidation (PSI), είναι για τους servers να κάνουν piggyback σε μια απάντηση στον proxy, τη λίστα των πόρων που έχουν τροποποιηθεί από την τελευταία πρόσβαση από αυτόν τον proxy. Ο proxy ακυρώνει καταχωρήσεις που έχουν γίνει cache στην λίστα και μπορεί να επεκτείνει τον χρόνο ζωής καταχωρήσεων που δεν είναι στην λίστα. Έχει επίσης προταθεί μια υβριδική προσέγγιση που συνδυάζει τις τεχνικές PSI και PCV για να επιτευχθεί η βέλτιστη απόδοση. Η επιλογή του μηχανισμού εξαρτάται από τον χρόνο αφότου ο proxy τελευταία ζήτησε ακύρωση του στοιχείου. Εάν ο χρόνος είναι μικρός, τότε χρησιμοποιείται ο μηχανισμός PSI, ενώ για μεγαλύτερα διαστήματα χρησιμοποιείται ο μηχανισμός PCV για καλύτερη αξιολόγηση των περιεχομένων της cache. Το λογικό είναι ότι για μικρά διαστήματα, ο αριθμός των ακυρώσεων που στέλνονται με το PSI είναι σχετικά μικρός, αλλά όσο ο χρόνος αυξάνεται, η επιβράδυνση για να αποσταλεί ακύρωση θα είναι μεγαλύτερη από την επιβράδυνση για ζητούμενες αξιολογήσεις.

ΜΕΡΟΣ ΙΙ : Caching Δυναμικού Περιεχομένου

ΚΕΦΑΛΑΙΟ 1 : ΕΙΣΑΓΩΓΗ

1.1 – Γενική Εισαγωγή

Όπως εξετάστηκε στις προηγούμενες ενότητες, η τεχνική του Web Caching ήταν εκείνη που έδωσε τη δυνατότητα στον Παγκόσμιο Ιστό να πάρει τις διαστάσεις που έχει σήμερα. Από ένα μικρό ερευνητικό πρόγραμμα στο CERN σε ένα τεράστιο αριθμό υπολογιστών που τρέχουν σήμερα το πρωτόκολλο HTTP σε όλο τον κόσμο.

Η ραγδαία αυτή αύξηση είχε ως αποτέλεσμα και την αλλαγή της φύσεως των εφαρμογών για τις οποίες χρησιμοποιείται το Web. Συγκεκριμένα, έδωσε πρόσφορο έδαφος στην εμφάνιση εμπορικών εφαρμογών, καθώς οι εταιρίες διέκριναν ότι το Internet (για μεγαλύτερη ακρίβεια το Web) αποτελούσε πλέον μια αγορά μεγάλης κλίμακας. Βεβαίως, οι εμπορικές εφαρμογές έχουν διαφορετικές απαιτήσεις από εκείνες της απλής παρουσίασης και διασύνδεσης εγγράφων για τις οποίες και δημιουργήθηκε αρχικά το HTTP και η HTML. Το εμπόριο απαιτεί σελίδες πλούσιες σε πληροφορία, εξατομικευμένες για τον εκάστοτε χρήστη και με περιεχόμενο πάντοτε ενημερωμένο, ασχέτως με το πόσο συχνά αυτό μεταβάλλεται.

Για παράδειγμα, μια εφαρμογή ηλεκτρονικού εμπορίου που επιτρέπει την αγορά προϊόντων από ένα πελάτη μέσω της ιστοσελίδας του καταστήματος, προϋποθέτει ότι η ιστοσελίδα αυτή θα πρέπει αρχικά να περιλαμβάνει όλες τις απαραίτητες πληροφορίες για κάθε προϊόν που ο πελάτης θέλει να αγοράσει (π.χ. ποσότητα, τιμή, ημερομηνία λήξης κ.α.). Επίσης, οι πληροφορίες αυτές επιβάλλεται να είναι συνεπείς με τις πραγματικές τιμές που έχει το κατάστημα την δεδομένη στιγμή. Παράλληλα, για τους συχνούς πελάτες είναι επιθυμητό να υπάρχει κάποιο είδος εξατομίκευσης, όπως προσωπικός χαιρετισμός, προσαρμοσμένη διεπαφή χρήστη κ.α.

Όπως, λοιπόν, φάνηκε από τα παραπάνω, υπήρξε η ανάγκη για δυναμική δημιουργία ιστοσελίδων, δηλαδή για τη δημιουργία των εκάστοτε απαντήσεων σε αιτήσεις HTTP με το που αυτές φθάνουν στον Web Server. Με αυτόν τον τρόπο θα δινόταν η δυνατότητα να παρέχονται ακριβώς οι πληροφορίες που επιθυμεί ο χρήστης, όπως είναι διαμορφωμένες εκείνη τη στιγμή στο περιβάλλον του εξυπηρέτη, και μάλιστα με εξατομικευμένο τρόπο που θα εξαρτιόταν από το ποιος πελάτης έστειλε την αίτηση και ποια χρονική στιγμή.

Με τον όρο, λοιπόν, σελίδες δυναμικού περιεχομένου εννοούνται εκείνες οι σελίδες που επιστρέφονται σε απάντηση μιας HTTP αίτησης, οι οποίες όμως δεν προϋπάρχουν και παραδίδονται απλά με την απάντηση, αλλά δημιουργούνται τη στιγμή της αίτησης στον Web Server.

Διάφορες τεχνολογίες υπάρχουν για την παραγωγή δυναμικού περιεχομένου, όπως ο μηχανισμός του CGI (Common Gateway Interface), τα Java Servlets, η JSP (Java Server Pages), η ASP (Active Server Pages) και η PHP. Όλες αυτές οι τεχνολογίες παρέχουν παρόμοιες λειτουργικές δυνατότητες στους

προγραμματιστές δικτυακών εφαρμογών για την παραγωγή δυναμικού περιεχομένου.

1.2 – Το Δυναμικό Περιεχόμενο δεν γίνεται Cache

Το πρόβλημα με τις σελίδες δυναμικού περιεχομένου είναι ότι δεν μπορούν να θεωρηθούν σαν καλοί υποψήφιοι για να φυλαχτούν στην cache ενός proxy. Αυτό οφείλεται στο γεγονός ότι, στη γενική περίπτωση, οι σελίδες δυναμικού περιεχομένου εξαρτώνται πάντα από την συγκεκριμένη αίτηση που προκάλεσε τη δημιουργία τους, ενώ ταυτόχρονα, πολλές από τις αιτήσεις για δυναμικό περιεχόμενο προκαλούν και άλλες συνέπειες πέρα από τη δημιουργία της απάντησης (όπως την ενημέρωση των στοιχείων κάποιας βάσης δεδομένων), με αποτέλεσμα να είναι αναγκαίο το να φθάσει η αίτηση στον Server και να μην εξυπηρετηθεί από την cache ενός ενδιάμεσου υπολογιστή. Έτσι, χωρίς τουλάχιστον την χρήση επιπρόσθετων μηχανισμών, το caching σελίδων δυναμικού περιεχομένου καθίσταται ασύμφορο, αν όχι απαγορευτικό.

Αυτή η έννοια, της ύπαρξης αντικειμένων (web objects) που δεν μπορούν να πάρουν μέρος στη διαδικασία του caching, οδηγεί στο διαχωρισμό του περιεχομένου του Παγκόσμιου Ιστού σε αντικείμενα που επιδέχονται caching (cacheable) και αντικείμενα που δεν επιδέχονται (uncacheable).

Γενικά, με την παρούσα τεχνολογία caching που είναι προσαρμοσμένη στο παραδοσιακό στατικό περιεχόμενο, οι ακόλουθες αιτήσεις και απαντήσεις στο Web κρίνονται ως uncacheable και δεν φυλάσσονται στους proxies:

- Αιτήσεις για σελίδες δυναμικού περιεχομένου και οι απαντήσεις σε αυτές θεωρούνται uncacheable για τους λόγους που αναφέρθηκαν παραπάνω. Η αναγνώριση των αιτήσεων και απαντήσεων που αφορούν δυναμικό περιεχόμενο γίνεται μέσα από το URL του δικτυακού πόρου που ζητείται ή επιστρέφεται. Ένα λοιπόν URL που περιέχει την συμβολοσειρά “cgi-bin” στο path του επιζητούμενου πόρου ή τελειώνει με το επίθεμα “.cgi” ή απλά έχει το σύμβολου του αγγλικού ερωτηματικού “?” θεωρείται uncacheable και δεν αποθηκεύεται στους proxies.
- Αιτήσεις με την επικεφαλίδα cookie και απαντήσεις με την επικεφαλίδα set-cookie θεωρούνται uncacheable γιατί η χρήση των cookies γίνεται συνήθως για λόγους εξατομίκευσης μιας σελίδας με βάση το χρήστη που τη ζητάει. Σε μια τέτοια περίπτωση, η αποθήκευση του αντικειμένου σε κάποια cache δεν θα έχει δυνατότητα επαναχρησιμοποίησης από άλλους χρήστες και επομένως θα παρουσιάζει εξαιρετικά μικρό hit rate.
- Αιτήσεις που χρησιμοποιούν μια μέθοδο άλλη από τις GET και HEAD δεν αποθηκεύονται. Αν και το HTTP 1.1 ορίζει μερικές περιπτώσεις που θα μπορούσαν να αποθηκευτούν, αυτές είναι τόσο σπάνιες που δεν υλοποιούνται στην πράξη.
- Αιτήσεις με την επικεφαλίδα authorization θεωρούνται και αυτές uncacheable καθ’ ότι και αυτές, όπως οι επικεφαλίδες των cookies,

αναφέρονται σε ένα συγκεκριμένο χρήστη, χωρίς δυνατότητα επαναχρησιμοποίησης από μια πλειάδα χρηστών του Web.

1.3 – Πόσο Διαδεδομένο είναι το Δυναμικό Περιεχόμενο

Σύμφωνα με τα παραπάνω, οι σελίδες δυναμικού περιεχομένου δεν μπορούν να αποθηκευτούν στις caches για να βελτιωθεί η απόδοση του Παγκόσμιου Ιστού. Πόσο μεγάλο πρόβλημα μπορεί να δημιουργηθεί στην γενική απόδοση του Web από αυτή την ιδιαιτερότητα του δυναμικού περιεχομένου;

Κάποιος θα μπορούσε να ισχυριστεί ότι αν απλά λειτουργούσαν μηχανισμοί προσωρινής αποθήκευσης για τις σελίδες στατικού περιεχομένου, ενώ για αυτές με το δυναμικό περιεχόμενο γινόταν ανάκτηση από τον Server, η απόδοση του Ιστού θα ήταν αρκετά καλή, ώστε να μην υπάρχει η ανάγκη να επινοηθούν μέθοδοι για σχήματα caching δυναμικού περιεχομένου. Πράγματι, ως και σήμερα, η πλειοψηφία των δικτυακών κόμβων υποστηρίζει τις παραδοσιακές τεχνολογίες caching για στατικά αντικείμενα, χωρίς να υποστηρίζεται αποθήκευση και εξυπηρέτηση δυναμικών σελίδων.

Όμως, το ποσοστό του δυναμικού περιεχομένου σε σχέση με το συνολικό όγκο κίνησης στο Web συνέχεια αυξάνει, με άμεσο αποτέλεσμα να καθίσταται αναγκαία η επέκταση των παρόντων μηχανισμών προκειμένου να γίνει εφικτή η αποθήκευση δυναμικών αντικειμένων στις caches.

Συγκεκριμένες μελέτες [Williams et al.1996a],[Feldmann et al.1999],[Wolman et al.1999a] που έγιναν λίγο πριν το 2000 έδειξαν ότι περίπου το 40% της συνολικής κίνησης του Παγκοσμίου Ιστού ανήκει στο uncacheable υλικό του Web. Ειδικότερα, τα αποτελέσματα φαίνονται στον παρακάτω πίνακα:

Πίνακας 1.1. Συχνότητα εμφάνισης χαρακτηριστικών στα μηνύματα του Web που τα καθιστούν uncacheable

Uncacheable Χαρακτηριστικό	Συχνότητα Εμφάνισης (%)
cache-control/pragma επικεφαλίδες	9–15
cookie/set-cookie επικεφαλίδες	19–30
authentication επικεφαλίδα	1–1.7
Άλλη μέθοδος από τις GET ή HEAD	1.4–2
"cgi-bin" ή "?" στο URL	1–20
uncacheable response code	22.8
Συνολικές uncacheable αιτήσεις	37–43

Μάλιστα, αυτό το ποσοστό αναμένεται να αυξηθεί στο μέλλον καθώς και ο αριθμός των δυναμικών ιστοσελίδων αναμένεται να μεγαλώσει με βάση τις επιταγές των σύγχρονων Web εφαρμογών, και οι αιτήσεις με μέθοδο POST να πολλαπλασιαστούν για τον ίδιο λόγο [Krishnamurthy and Rexford 2001, p377 Future trends].

Εφ' όσον, λοιπόν, το δυναμικό περιεχόμενο καταλαμβάνει ήδη ένα αρκετά μεγάλο ποσοστό του όγκου κίνησης δεδομένων στον Παγκόσμιο Ιστό και όλα δείχνουν πως αυτό θα αυξάνεται συνέχεια, η ανάγκη για χειρισμό της προσωρινής αποθήκευσης δυναμικών ιστοσελίδων είναι επιτακτική.

1.4 – Πόσο Συχνά Αλλάζει το Περιεχόμενο του Web

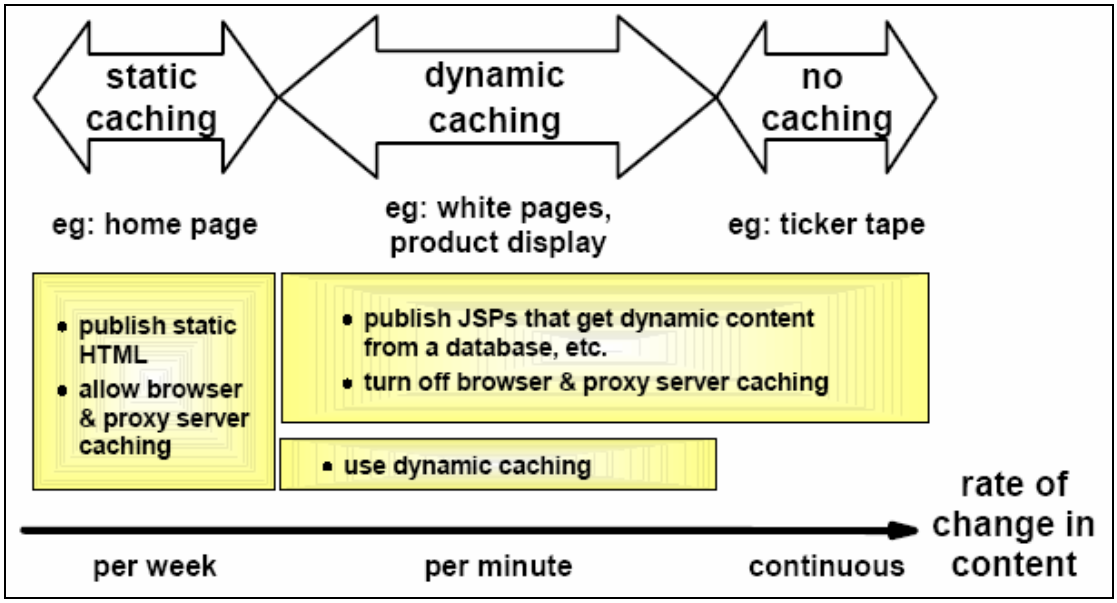
Μέχρι τώρα αναφέρθηκε ο διαχωρισμός των πόρων σε στατικούς και δυναμικούς, με βάση τον ισχυρισμό ότι οι μεν δυναμικοί είναι εκείνοι οι πόροι που αλλάζουν, οι δε στατικοί είναι εκείνοι που παραμένουν αναλλοίωτοι. Αυτό, όμως δεν είναι απόλυτα ακριβές.

Στην πραγματικότητα δεν υπάρχει περιεχόμενο που να μην αλλάζει ποτέ, καθώς πάντα κάποια στιγμή ο πάροχος της πληροφορίας που μια στατική σελίδα περιέχει, αποφασίζει ότι αυτή είναι πλέον ξεπερασμένη και πρέπει να αντικατασταθεί από μια νεότερη έκδοσή της. Δηλαδή, όλες οι σελίδες στον Παγκόσμιο Ιστό αλλάζουν αργά ή γρήγορα. Συνεπώς, δημιουργείται το ερώτημα που αφορά στο πόσο συχνά πρέπει να αλλάζει το περιεχόμενο ενός πόρου στο διαδίκτυο ώστε αυτός να μην θεωρείται πια στατικός αλλά δυναμικός.

Προκειμένου να διερευνηθεί το ερώτημα αυτό, οι [Douglis et al. 1997] μελέτησαν την μετρική της συχνότητας αλλαγής(change ratio) των πόρων στο διαδίκτυο, που ορίζεται ως το ποσοστό των αιτήσεων που επιστρέφουν ένα αντικείμενο που είναι διαφορετικό από την απάντηση στην προηγούμενη αίτηση για αυτό (που μπορεί να υπολογιστεί από την last-modified timestamp που υποστηρίζει το HTTP). Τα ευρήματά τους έδειξαν ότι οι πόροι του διαδικτύου, πέραν εκείνων που είναι δυναμικά παραγόμενοι, παρουσιάζουν ένα change ratio της τάξης του 15%, που σημαίνει ότι παρουσιάζουν αλλαγές μεταξύ δύο διαδοχικών αιτήσεων μόνο στο 15% των περιπτώσεων. Στο υπόλοιπο 85% το περιεχόμενο είναι το ίδιο και μπορεί να εξυπηρετηθεί από μια proxy cache. Το περιεχόμενο αυτό, που μεταβάλλεται μεν, αλλά με μικρό σχετικά change ratio, είναι αυτό που καθιερωμένα αποκαλείται στατικό.

Οι [Krishnamurthy and Rexford 2001] παρατηρούν επιπλέον, ότι από τους στατικούς πόρους οι εικόνες αλλάζουν σε πολύ μικρότερο βαθμό από το κείμενο και την HTML στο Web. Παρ' όλα αυτά, επισημαίνουν ότι οι συχνότητες και τρόποι αλλαγής του περιεχομένου του Παγκοσμίου Ιστού πρόκειται να αλλάξουν στο μέλλον, καθώς όλο και περισσότερες σελίδες θα επικοινωνούν με βάσεις δεδομένων προκειμένου να παράγουν το περιεχόμενό τους. Έτσι, προβλέπεται ότι οι πόροι στο διαδίκτυο θα παρουσιάζουν μια πιο δυναμικά μεταβαλλόμενη μορφή, η οποία όπως έχει συζητηθεί θα επιδρά ανασταλτικά στην εφαρμογή του caching.

Για ανακεφαλαίωση παρατίθεται το ακόλουθο διάγραμμα [Copeland and McClain] που παρουσιάζει το πόσο συχνά μεταβάλλεται το περιεχόμενο σε σχέση με το ποιες τεχνολογίες είναι καταλληλότερες για την υλοποίηση μιας τέτοιας εφαρμογής και το ποιες μέθοδοι caching αρμόζουν ανά περίπτωση:



Όπως λοιπόν διαφαίνεται, όταν κάποιος πόρος μεταβάλλεται σε εβδομαδιαία βάση (και αραιότερα) τότε μπορεί να χαρακτηριστεί στατικός και να υπαχθεί στις συνήθεις διαδικασίες στατικού caching, με αρκετά καλά αποτελέσματα. Παράλληλα, σε περιπτώσεις όπου η μεταβολή είναι αρκετά συχνή, έως και της τάξεως του λεπτού, τότε ο πόρος θεωρείται δυναμικός και για την παρουσίασή του χρησιμοποιούνται τεχνολογίες παραγωγής δυναμικού περιεχομένου, όπως αυτές που έχουν προαναφερθεί. Όσον αφορά το caching, σε περιστάσεις σαν αυτή, μπορούν να εφαρμοστούν κάποιες ειδικές μέθοδοι caching δυναμικού περιεχομένου (στο διάγραμμα αναφέρονται ως dynamic caching), με ενδιαφέροντα αποτελέσματα. Σε όλες τις λοιπές περιπτώσεις, όπου η αλλαγή των πόρων είναι πολύ συχνή (ίσως και διαρκής) κάθε ιδέα για caching είναι άσκοπη, καθώς δεν αναμένεται να έχει καμία θετική συνέπεια.

1.5 – Καθυστερήσεις στην Παραγωγή Δυναμικού Περιεχομένου

Ο κύριος σκοπός του Web caching είναι η επίτευξη κλιμακωσιμότητας (scalability) και η βελτίωση της απόδοσης του Παγκοσμίου Ιστού. Προκειμένου, λοιπόν, να γίνει μια αποτελεσματική εφαρμογή του caching στις δυναμικές σελίδες πρέπει να μελετηθούν οι καθυστερήσεις που υπεισέρχονται στην όλη διαδικασία της παραγωγής του δυναμικού περιεχομένου. Μια περιληπτική αναφορά όλων των στενωπών που συναντώνται στην παραγωγή δυναμικού περιεχομένου βρίσκεται στο [Yagoub et al. 2000], όπου παρατίθεται η παρακάτω λίστα, με τις εμπλεκόμενες καθυστερήσεις σε σειρά που εμφανίζονται στη διαδικασία:

Χρόνος αναμονής κατά την διαδικασία παραγωγής δυναμικού περιεχομένου
(1) Network Communication Time (χρόνος δικτυακής επικοινωνίας)
(2) HTTP Connection Time (εγκαθίδρυση HTTP σύνδεσης)
(3) Web Application Startup Time (έναρξη δικτυακής εφαρμογής)
(4) DBMS Connection Time (σύνδεση με τη βάση δεδομένων)
(5) SQL Execution Time (χρόνος εκτέλεσης SQL επερώτησης)
(6) XML Generation Time (χρόνος παραγωγής XML)
(7) HTML Generation Time (χρόνος παραγωγής HTML)

Η διαδικασία παραγωγής του δυναμικού περιεχομένου έχει ως εξής:

Όταν κάποιος χρήστης του διαδικτύου ζητήσει να δει μια σελίδα δυναμικού περιεχομένου στο πρόγραμμα πλοήγησης (browser) που χρησιμοποιεί, αποστέλλεται μια HTTP αίτηση προς τον εξυπηρέτη που φιλοξενεί την δικτυακή εφαρμογή, η οποία ταξιδεύει μέσα στο δίκτυο σαν πακέτα του πρωτοκόλλου TCP (TCP segments) και υπόκειται στις γνωστές καθυστερήσεις δικτύου. Αυτές συνοπτικά είναι οι καθυστερήσεις μετάδοσης, που έχουν να κάνουν με το εύρος ζώνης που διαθέτει κάθε γραμμή του δικτύου (π.χ. 100Mbps), οι καθυστερήσεις διάδοσης, που εξαρτώνται από τις αποστάσεις που διανύουν τα πακέτα μέσα στο δίκτυο μέχρι να φθάσουν στον τελικό προορισμό τους, την καθυστέρηση αναμονής στους ενδιάμεσους κόμβους του δικτύου, που κάνουν την δρομολόγηση των πακέτων, και εμφανίζεται όταν υπάρχει συμφόρηση στο δίκτυο και τέλος την καθυστέρηση επαναμετάδοσης των TCP πακέτων που είτε παρελήφθησαν λανθασμένα από τον παραλήπτη, είτε δεν έφθασαν καθόλου στον server.

Με το που θα φθάσει αυτή η αίτηση στον Web Server τότε αποδίδεται στην αίτηση αυτή, και σε όσες πρόκειται να ληφθούν από τον ίδιο χρήστη, ένας ειδικός αριθμός ταυτοποίησης που λέγεται session ID και χρησιμοποιείται εσωτερικά από τον server προκειμένου να ξεχωρίζει που πρέπει να στείλει τις απαντήσεις, δεδομένου ότι μπορεί ταυτόχρονα πολλοί χρήστες να υποβάλλουν HTTP αιτήσεις στον εξυπηρέτη. Η διαδικασία αυτή, σε συνδυασμό με την υποκείμενη εγκαθίδρυση της TCP σύνδεσης που διαδραματίζεται στο παρασκήνιο, εισάγουν την καθυστέρηση της εγκαθίδρυσης της HTTP σύνδεσης.

Η διαδικασία μέχρι αυτή την στιγμή είναι η ίδια και για τις αιτήσεις που απευθύνονται σε στατικό περιεχόμενο. Η διαφορά ξεκινά μόλις ο Web server επεξεργαστεί την αίτηση και συμπεράνει ότι πρέπει να ενεργοποιήσει την εκτέλεση κάποιου script για να παραχθεί το δυναμικό περιεχόμενο. Βέβαια, προκειμένου να τρέξει το script πρέπει να προηγηθούν κάποιες άλλες λειτουργίες, ανάλογα με την τεχνολογία που χρησιμοποιείται, όπως για παράδειγμα η εκκίνηση ενός interpreter. Αυτές οι λειτουργίες αποτελούν την φάση εκκίνησης της Web εφαρμογής και προφανώς προσθέτουν στον συνολικό χρόνο που απαιτείται για να προετοιμαστεί η απάντηση.

Οι δυναμικές σελίδες στη μεγάλη τους πλειοψηφία ανακτούν τις πληροφορίες που εμφανίζουν από μία ή και περισσότερες βάσεις δεδομένων που βρίσκονται στο τοπικό δίκτυο της εταιρείας ή οργανισμού που διατηρεί το web site. Έτσι, κάποια στιγμή κατά την εκτέλεση του script που παράγει την απάντηση, επιχειρείται η σύνδεση με τη βάση στην οποία αποθηκεύονται τα δεδομένα, προκειμένου να αποσταλεί σε αυτή μια SQL επερώτηση που θα ανακτήσει τα δεδομένα που πρέπει να συμπεριληφθούν στη συγκεκριμένη σελίδα. Η σύνδεση αυτή είναι ούτως ή άλλως μια αρκετά χρονοβόρα διαδικασία, και αν συλλογιστεί κανείς ότι τις περισσότερες φορές το σύστημα διαχείρισης της βάσης δεδομένων βρίσκεται σε διαφορετικό μηχάνημα από ότι ο Web Server, γίνεται εύκολα αντιληπτό ότι παίζει σημαντικό ρόλο στην καθυστέρηση δημιουργίας του δυναμικού περιεχομένου.

Μετά την σύνδεση με τη βάση δεδομένων, το container μέσα στο οποίο τρέχει η δικτυακή εφαρμογή αποστέλλει κάποιες SQL επερωτήσεις που ανακτούν το κατάλληλο περιεχόμενο μεταξύ των εγγραφών που είναι αποθηκευμένες στη βάση. Αυτές μπορεί να είναι ιδιαίτερα απλές και να εκτελούνται σχετικά γρήγορα, αλλά μπορεί εξίσου να είναι και εξαιρετικά πολύπλοκες και να απαιτούν σημαντικό χρόνο επεξεργασίας από το σύστημα της βάσης δεδομένων. Σε αμφότερες τις περιπτώσεις υπάρχει κάποιος χρόνος καθυστέρησης που υπεισέρχεται στην διαδικασία.

Στα σύγχρονα εμπορικά συστήματα, τα δεδομένα που ανακτούνται από τη βάση επιστρέφονται στα υπόλοιπα στρώματα της εφαρμογής (οι λεγόμενες πολυστρωματικές – multitiered εφαρμογές) κωδικοποιημένα στη γλώσσα περιγραφής δεδομένων XML για διάφορους λόγους, όπως η δυνατότητα διεπαφής μεταξύ ετερογενών συστημάτων. Σε μια τέτοια περίπτωση, στην όλη διαδικασία της παραγωγής δυναμικού περιεχομένου περιλαμβάνεται και η μετατροπή των πρωταρχικών δεδομένων (raw data) της βάσης σε μορφή XML.

Στο τέλος της όλης διαδικασίας είναι το επίπεδο – στρώμα της παρουσίασης, που είναι υπεύθυνο για το πώς θα παρουσιαστούν τα δεδομένα στον χρήστη. Εδώ τοποθετούνται τα δεδομένα στα σωστά σημεία της σελίδας, ενώ ταυτόχρονα γίνεται και η μετατροπή των XML δεδομένων σε κώδικα HTML που καθορίζει πέρα από αυτά καθ' εαυτά τα δεδομένα και τον τρόπο που θα εμφανιστούν στην οθόνη. Μετά την παραγωγή της απάντησης μένει μονάχα να επιστραφεί πίσω στον Web Server, ο οποίος αναλαμβάνει να την αποστείλει στον σωστό χρήστη.

Από τα παραπάνω διαφάνηκε η αυξημένη πολυπλοκότητα της παραγωγής του δυναμικού περιεχομένου, η οποία είναι πολύ σημαντικό να είναι κατανοητή όταν μελετώνται διάφορες τεχνικές caching, διότι όλες οι τεχνικές προσπαθούν να βελτιώσουν την απόδοση του συστήματος στα παραπάνω επτά σημεία καθυστέρησης.

Προτού κλείσει η παρούσα ενότητα, είναι σκόπιμο να αναφερθεί ότι ο χρόνος δικτυακής επικοινωνίας (το 1 στη λίστα) λέγεται και καθυστέρηση δικτύου (network latency), ενώ τα υπόλοιπα (2-7) συνθέτουν την καθυστέρηση εξυπηρέτη (server latency) διότι ακριβώς εντοπίζονται εντός των νοητών ορίων της επιχείρησης που διαχειρίζεται τον δικτυακό τόπο.

1.6 – Γενική Επισκόπηση

Πολλές προσπάθειες έχουν γίνει στα πλαίσια του δυναμικού caching, με απώτερο σκοπό να αναπτυχθούν μέθοδοι που θα καθιστούν το caching δυναμικού περιεχομένου όσο το δυνατόν πιο αποδοτικό. Για την απλοποίηση της μελέτης όλων αυτών των επιστημονικών εργασιών είναι αναγκαίο να γίνει αρχικά μια ταξινόμηση τους. Η ταξινόμηση των μεθόδων caching που αφορούν δυναμικό περιεχόμενο μπορεί να γίνει με βάση του τι απάντηση δίνεται από κάθε μία από αυτές στα παρακάτω ερωτήματα:

Πού τοποθετούνται οι caches;
Τι είναι προτιμότερο να αποθηκεύεται στις caches;
Πως θα πρέπει να ενημερώνονται για τις αλλαγές των πόρων οι caches;

Που τοποθετούνται οι caches;
Όσον αφορά την τοποθεσία της cache, στο πρώτο μέρος είχε αναλυθεί το πώς οι caches τοποθετούνται στο παραδοσιακό caching σαν reverse & forward proxies. Οι ίδιες αρχιτεκτονικές όπως είναι αναμενόμενο μπορούν να χρησιμοποιηθούν και στο δυναμικό caching. Πέρα όμως από αυτές, στην περίπτωση του δυναμικού caching μπορούν να εφαρμοστούν αρχιτεκτονικές που εντοπίζονται στην πλευρά του server (back-end caching).

Στην κατηγορία, λοιπόν, του ***back-end caching*** ανήκουν τα [Challenger et al. 1997] , [Challenger et al. 1999], [TenTimesTen 2000], [Yagoub et al. 2000], [Labrinidis-Roussopoulos 2000], [Luo et al. 2000], [Datta et al. 2001a], [Datta et al. 2001b], [Zhu and Yang 2001] στα οποία προτείνονται κάποιες αρχιτεκτονικές που έχουν σαν σκοπό να βελτιώσουν την απόδοση του συστήματος caching, αντιμετωπίζοντας ένα ή και περισσότερα από τα προβλήματα που εισάγουν καθυστέρηση στον server (server latency) και αναφέρθηκαν προηγουμένως. Για παράδειγμα, στο [Yagoub et al. 2000] παρουσιάζεται ένα σύστημα στο οποίο εφαρμόζονται τεχνικές caching εντός του server, σε διάφορα σημεία από εκείνα στα οποία εμφανίζονται καθυστερήσεις, όπως σε επίπεδο αποτελεσμάτων SQL επερωτήσεων, XML δεδομένων και HTML σελίδων και τμημάτων σελίδων. Το [Labrinidis-Roussopoulos 2000] προτείνει το caching αποθηκευμένων όψεων της βάσης, ενώ τα [Challenger et al. 1997], [Challenger et al. 1999] παρουσιάζουν τον τρόπο με τον οποίο οι συγγραφείς αντιμετώπισαν τις υψηλές απαιτήσεις σε κλιμακωσιμότητα των δικτυακών τόπων των Ολυμπιακών Αγώνων της Ατλάντα το 1996 και των Χειμερινών Ολυμπιακών Αγώνων του 1998, για λογαριασμό μεγάλης εταιρείας πληροφορικής. Μάλιστα, στα δύο αυτά άρθρα αναλύεται ένας ενδιαφέρον αλγόριθμος που ονομάζεται DUP (Data Update Propagation) και επιτρέπει στο σύστημα να υπολογίζει άμεσα τις σελίδες, των οποίων τα δεδομένα αλλάζουν κάποια στιγμή στη βάση.

Γενικά, το προτέρημα των back-end caching αρχιτεκτονικών είναι ότι, επειδή βρίσκονται εντός του server, υπάρχει όλη η πληροφορία που απαιτείται για να επιτευχθεί caching σε πιο λεπτομερές επίπεδο από ότι το επίπεδο ολόκληρης της HTML σελίδας, όπως για παράδειγμα της αποθήκευση στην cache των τμημάτων των σελίδων που παραμένουν σταθερά, παρά τη δυναμική φύση της σελίδας. Επίσης, για τον ίδιο λόγο, τα συστήματα caching αυτής της κατηγορίας μπορούν να παρέχουν στους χρήστες πάντα συνεπείς (consistent) απαντήσεις, αφού γνωρίζουν αν τα δεδομένα που συνθέτουν μια απάντηση έχουν αλλάξει κατά την άφιξη μιας νέας αίτησης για αυτή. Βέβαια, οι προσεγγίσεις αυτές παρουσιάζουν ένα σημαντικό μειονέκτημα, με αποτέλεσμα το μεγαλύτερο κομμάτι της έρευνας να διοχετεύεται προς τα συστήματα proxy caching. Το μειονέκτημα αυτό είναι ότι επιτυγχάνουν βελτιώσεις μονάχα στις καθυστερήσεις του εξυπηρέτη, χωρίς όμως να βοηθούν καθόλου στις καθυστερήσεις δικτύου, ένα σημείο το οποίο είναι ιδιαίτερα κρίσιμο λόγω του αυξημένου κόστους των δικτυακών πόρων.

Για τον λόγο, λοιπόν, αυτόν οι αρχιτεκτονικές ***proxy και reverse-proxy*** απασχολούν περισσότερο τους ερευνητές ανά τον κόσμο, ώστε να προκύψουν νέα αποτελέσματα. Για υπενθύμιση, μια cache λειτουργεί σε reverse-proxy τρόπο λειτουργίας όταν είναι τοποθετημένη ακριβώς μπροστά από τον server που φιλοξενεί την εφαρμογή και τα firewalls που τον προστατεύουν, προκειμένου να φιλτράρει τις εισερχόμενες αιτήσεις, ώστε αυτές να μην επιβαρύνουν άδικα το εταιρικό δίκτυο. Σε αυτή την λειτουργία, οι caches λέγονται και επιταχυντές (web accelerators). Αντίθετα, μια cache αποκαλείται και proxy cache όταν είναι τοποθετημένη σε κάποιο κομβικό σημείο μέσα στο δίκτυο, συνήθως πιο κοντά στον πελάτη (client) και απαντά αντιπροσωπεύοντας τον server προορισμού, σε περίπτωση φυσικά που έχει αποθηκευμένη την απάντηση στην αίτηση που δέχεται.

Στην κατηγορία των reverse-proxies ανήκουν τα [Datta et al. 2002], [Candan et al. 2001] καθώς και πολλά εμπορικά συστήματα σαν τα [InktomiCorp.], [MicrosoftISA], [NetworkAppliance], [CacheFlow] και την [OracleWebCache 2002]. Στα forward-proxies ανήκουν τα [Attar et al. 2002], [Luo et al. 2001], [Cao et al. 1998], [Gadde et al. 1997], [Rabinovich et al. 1999].

Σε γενικές γραμμές, τα συστήματα που ανήκουν σε αυτές τις κατηγορίες προσφέρουν καλύτερα αποτελέσματα από τα back-end σε σχέση με τις οικονομίες σε χρόνο δικτυακής επικοινωνίας. Τα συστήματα forward-proxy, μάλιστα, επιτυγχάνουν και πιο εκτεταμένα αποτελέσματα καθώς εξυπηρετούν πολλούς χρήστες και servers ταυτόχρονα, κάτι που συναντάται και σε διατάξεις reverse-proxy που επιταχύνουν πάνω από ένα sites (σε αυτή την περίπτωση οι caches λέγονται και surrogates). Όπως όμως είναι φυσικό, τα συστήματα αυτά είναι πιο δύσκολα στην υλοποίηση, αφού στερούνται της γνώσης της κατάστασης του server και συνεπώς εμφανίζουν προβλήματα συνέπειας.

Τέλος, έχουν κατά καιρούς προταθεί και συστήματα δυναμικού caching που τοποθετούνται στους web clients, όπως ακριβώς και οι παραδοσιακές caches που διαθέτουν οι περισσότεροι browsers. Παραδείγματα τέτοιων προτάσεων είναι τα [Brabrand et al. 2001], [Rabinovich et al. 2003].

Οι προτάσεις αυτές, όπως γίνεται κατανοητό, δεν επιτυγχάνουν τα ίδια θετικά αποτελέσματα με τις άλλες αρχιτεκτονικές, γιατί εξυπηρετούν μονάχα έναν χρήστη και βελτιώνουν την καθυστέρηση που εκείνος αντιλαμβάνεται, χωρίς όμως να παρουσιάζουν την εκτεταμένη εμβέλεια των άλλων τεχνικών δυναμικου caching.

Τι είναι προτιμότερο να αποθηκεύεται στις caches;
Πέρα από την ταξινόμηση των συστημάτων δυναμικού caching με βάση την τοποθεσία της cache, είναι δυνατόν τα συστήματα αυτά να κατηγοριοποιηθούν σύμφωνα με το τί ακριβώς αποθηκεύεται στις caches. Στο παραδοσιακό caching αποθηκεύονται ολόκληρες οι σελίδες HTML που στέλνονται ως απαντήσεις στις αιτήσεις των χρηστών, καθώς επίσης και διάφορα αντικείμενα, όπως εικόνες (.jpg, .gif κ.α) και έγγραφα (.pdf, .doc κ.α).

Όπως έχει ήδη αναλυθεί, το δυναμικό caching επικεντρώνεται στην αποθήκευση του δυναμικού περιεχομένου, το οποίο είναι συνήθως η δυναμικά δημιουργούμενη HTML, αφού τα υπόλοιπα αντικείμενα, όπως οι εικόνες, είναι συνήθως στατικά (αν και υπάρχουν τρόποι για τη δυναμική δημιουργία και των εικόνων σε μια σελίδα).

Σύμφωνα με τα παραπάνω, λοιπόν, στα δυναμικά συστήματα caching γίνεται αποθήκευση αρχείων που περιέχουν HTML. Εκτός από αυτή την περίπτωση υπάρχουν συστήματα που αποθηκεύουν και ***άλλου τύπου δεδομένα***, όπως XML ή αποτελέσματα επερωτήσεων SQL, αλλά αυτά είναι κυρίως συστήματα back-end caching. Για τα συστήματα proxy-caching έχουν επικρατήσει δύο εναλλακτικές λύσεις: η αποθήκευση ***τμημάτων HTML*** (HTML fragments) και η αποθήκευση ολόκληρων ***HTML σελίδων***.

Η αποθήκευση τμημάτων HTML συνδυάζεται με τεχνικές δυναμικής συναρμολόγησης σελίδας (dynamic page assembly), οι οποίες προβλέπουν τον τρόπο με τον οποίο οι caches μπορούν να συνθέσουν τις απαντήσεις που θα δώσουν στους χρήστες από τα τμήματα σελίδων HTML που έχουν αποθηκευμένα. Αν και η λογική και τα πλεονεκτήματα αυτών των τεχνικών θα παρουσιαστούν αναλυτικά σε επόμενη ενότητα, σε γενικές γραμμές η αποθήκευση τμημάτων μιας HTML σελίδας επιτρέπει την καλύτερη εκμετάλλευση του αποθηκευτικού χώρου της cache (δηλαδή την επίτευξη υψηλότερων hit rates) επειδή μπορεί να αποθηκεύμει μονάχα εκείνα τα τμήματα που παραμένουν σχετικά σταθερά σε μια δυναμική σελίδα, όπως για παράδειγμα το μενού πλοήγησης του site που αλλάζει σπανιότερα από το υπόλοιπο περιεχόμενο.

Αντίθετα με τη δυναμική συναρμολόγηση σελίδας, όταν αποθηκεύονται ολόκληρες HTML σελίδες χρησιμοποιείται μια άλλη τεχνική, η οποία ονομάζεται κωδικοποίηση διαφορών (Delta Encoding). Στο Delta Encoding αξιοποιείται το γεγονός ότι μεταξύ δύο απαντήσεων σε αιτήσεις για την ίδια σελίδα, οι διαφορές είναι μικρές ακόμα και αν πρόκειται για δυναμικά παραγόμενη ιστοσελίδα. Με αυτό το σκεπτικό, αν ο server στέλνει μόνο την διαφορά των δύο απαντήσεων, η cache μπορεί να σχηματίσει τη νέα απάντηση συμβουλευόμενη την παλιά και τη «λίστα» των μεταξύ τους διαφορών. Η τεχνική του Delta Encoding θα εξεταστεί με ιδιαίτερη λεπτομέρεια στη συνέχεια.

Οι παρακάτω εργασίες ανήκουν στην κατηγορία των συστημάτων που αποθηκεύουν αντικείνα διαφορετικά από την HTML των απαντήσεων: [Florescu et al. 1999], [Yagoub et al. 2000], [Labrinidis-Roussopoulos 2000], [Luo et al. 2001], [Luo et al. 2002], [Attar et al. 2002]. Τα [Challenger et al. 1997], [Holmedahl et al. 1998], [Cao et al. 1998], [Candan et al. 2001] ανήκουν στο caching ολόκληρων σελίδων, ενώ τα [Douglis et al. 1997b], [Challenger et al. 2000], [Brabrand et al. 2001], [OracleWebCache 2002], [Datta et al. 2002] στο caching τμημάτων HTML.

Η συγκριτική αξιολόγηση των δύο εναλλακτικών περιπτώσεων του caching τμημάτων HTML και ολόκληρων σελίδων παρουσιάζει εξαιρετικό ενδιαφέρον

και θα παρουσιαστεί σε επόμενη ενότητα, αφού μελετηθούν διεξοδικά οι δύο αρχιτεκτονικές.

Πως θα πρέπει να ενημερώνονται για τις αλλαγές των πόρων οι caches; Τελευταία ταξινόμηση των συστημάτων δυναμικού caching είναι αυτή που γίνεται με βάση του τρόπου με τον οποίο μεταδίδονται οι αλλαγές του περιεχομένου στην cache. Οι εναλλακτικές δυνατότητες είναι δύο, το μοντέλο του ***pull-caching*** και το μοντέλο του ***push-caching***. Η πρώτη περίπτωση είναι η πιο διαδεδομένη, καθώς πρόκειται για την προκαθορισμένη συμπεριφορά των caches στο παραδοσιακό web caching, όπου η ενημέρωση για τις αλλαγές γίνεται με την αίτηση κάποιου χρήστη για τον συγκεκριμένο πόρο. Η δεύτερη περίπτωση πρόκειται για ένα αρκετά ενδιαφέρον μοντέλο, όπου οι αλλαγές του περιεχομένου αντικατοπτρίζονται άμεσα στις caches, με ποικίλες τεχνικές.

Το pull-caching είναι η συμπεριφορά του παραδοσιακού Web Caching, οπότε οι ιδέες που είναι σχετικές με αυτό υπάρχουν διάσπαρτες σε όλες τις εργασίες που αφορούν το Web Caching. Μεγαλύτερο ανδιαφέρον παρουσιάζουν οι εργασίες που είναι σχετικές με το push-caching, που είναι και πιο σπάνιες. Το push caching είναι το μοντέλο που προβλέπει την μετάδοση πληροφορίας από τον server στον proxy με πρωτοβουλία του server. Μια αρχική μορφή τέτοιας συμπεριφοράς (αν και δεν συγκαταλέγεται στο push caching) είναι η μέθοδος του invalidation, η οποία είναι αρκετά παλιά στο web caching με την ιδέα των leases [Gray and Cheriton 1989] και αποτυπώνεται άριστα στην εργασία [Candan et al. 2001] μέσα στα όρια του δυναμικού web caching. Για τη μέθοδο του invalidation στο caching δυναμικού περιεχομένου έχουν αναπτυχθεί διάφορα πρωτόκολλα, όπως το ESI Invalidation Protocol 1.0 [ESI Technical Specs] και το πρωτόκολλο WCIP [Li et al. 2001], για τα οποία και γίνονται κινήσεις για την πρωτοτυποποίησή τους. Για το καθαρό μοντέλο push υπάρχουν λιγότερα επιστημονικά έργα, βασικό εκ των οποίων είναι το [Chen et al. 1999], στο οποίο γίνεται η ταυτοποίηση διαφόρων προβλημάτων σχετικών με την κλιμακωσιμότητα του web και αναγνωρίζεται η ανάγκη για την ύπαρξη ενός τρόπου για την προώθηση των ενημερώσεων ενός site στους proxy που το εξυπηρετούν. Στα πλαίσια αυτά προτείνεται μια νέα μέθοδος PUSH για το πρωτόκολλο HTTP. Τέλος, υπάρχει και μια υλοποίηση ανοικτού λογισμικού ([Squid Push Patch]) για την υποστήριξη ενός τέτοιου push μηχανισμού στην cache, αλλά δεν χρησιμοποιείται η μέθοδος PUSH, αλλά η PUT σε διαφορετικό ρόλο.

Συγκριτικά, τα δύο μοντέλα εμφανίζουν πλεονεκτήματα και μειονεκτήματα. Το μοντέλο του push-caching έχει το πλεονέκτημα να μπορεί να παρέχει πάντοτε ενημερωμένα δεδομένα στον χρήστη κάτι που είναι αδύνατον με το μοντέλο του pull-caching. Βέβαια, η επιπλέον αυτή δυνατότητα του push-caching δεν έρχεται χωρίς κόστος. Συνήθως, τα συστήματα αυτά απαιτούν να καταβληθεί αρκετό υπολογιστικό κόστος για να υποστηριχθεί ο μηχανισμός που θα εντοπίζει τις αλλαγές του περιεχομένου και θα ενημερώνει κατάλληλα τις caches. Συνεπώς, η απόφαση επιλογής μεταξύ των δύο μοντέλων έγκειται στις εκάστοτε ανάγκες που καλείται να εξυπηρετήσει το σύστημα caching,

καθώς και από τη στατιστική που ακολουθούν οι αιτήσεις για τον συγκεκριμένο δυναμικό πόρο που προκειται να γίνει cache.

ΚΕΦΑΛΑΙΟ 2 : ΔΥΝΑΜΙΚΗ ΣΥΝΑΡΜΟΛΟΓΗΣΗ ΣΕΛΙΔΑΣ

2.1 – Εισαγωγή

Ο όρος δυναμική συναρμολόγηση σελίδας (dynamic page assembly) αναφέρεται στη συναρμολόγηση κάθε δυναμικής σελίδας από τμήματα της σελίδας που είναι αποθηκευμένα σε έναν proxy. Αυτό προύποθέτει ότι μια σελίδα είναι χωρισμένη σε τμήματα (fragments), τα οποία αποθηκεύονται ανεξάρτητα στην cache και μάλιστα εμφανίζουν διαφορετικούς χρόνους ζωής που επιτρέπεται να μείνουν στην cache.

Έτσι, αν χωριστεί μια δυναμική σελίδα σε τμήμαα ανάλογα με τον ρυθμό με τον οποίο αλλάζει στα συγκεκριμένα σημεία, π.χ. το μέρος όπου τοποθετείται η ημερομηνία αλλάζει κάθε μέρα και εντάσσονται σε ένα τμήμα, αλλά οι πληροφορίες για τα αποτελέσματα κάποιων αθλητικών γεγονότων αλλάζουν κάθε λεπτό και ενάσσονται σε άλλο τμήμα της σελίδας. Με αυτόν τον τρόπο, ενώ κανονικά η σελίδα θα είχε χρόνο ζωής τον ελάχιστο από όλα τα τμήματα (στο π.χ. 1 λεπτό), με την τεχνική του dynamic page assembly κάθε τμήμα έχει το χρόνο ζωής που του αρμόζει.

Η διαδικασία εξυπηρέτησης μιας σελίδας από μια cache στην οποία είναι αποθηκευμένα τα τμήματά της συνίσταται στο να ανακτηθούν από την cache τα τμήματα που είναι ακόμα έγκυρα εκείνη τη δεδομένη στιγμή, ενώ όσα έχουν ξεπεράσει το χρόνο ζωής τους να ανακτούναι από τον server.

Στην επόμενη ενότητα θα εξεταστεί η πιο οργανωμένη και εμπεριστατομένη προσπάθεια προς την κατεύθυνση της δυναμικής συναρμολόγησης σελίδας, η ESI. Στα πλαίσια αυτής της συζήτησης θα γίνει και η ανάλυση των πλεονεκτημάτων και μειονεκτημάτων που παρουσιάζει η τεχνική αυτή.

2.2 – Edge Side Includes (ESI)

Σε αυτή την ενότητα θα παρουσιαστεί η γλώσσα Edge-Side Includes (ESI) που χρησιμοποιείται για την δυναμική συναρμολόγηση σελίδων από τα διάφορα τμήματά της στην cache.

Η ESI είναι μια mark-up γλώσσα, όπως η HTML, η οποία βασίζεται στη γλώσσα περιγραφής δεδομένων XML. Η ESI αναπτύχθηκε από κοινού από τις εταιρείες Oracle και Akamai προκειμένου να χρησιμοποιηθεί στη συναρμολόγηση πόρων σε HTTP πελάτες (clients). Συνεπώς έχει συγκεκριμένους στόχους που συνοψίζονται στην χρησιμοποίηση από αρχιτεκτονικές caching σε σημεία που βρίσκονται κοντά στον πελάτη (edge), ώστε να επιτευχθεί μείωση των καθυστερήσεων που αντιλαμβάνεται ένας χρήστης που αιτεί έναν πόρο, μείωση στο υπολογιστικό φορτίο του server και αύξηση της διαθεσιμότητας (availability) της εφαρμογής. Τα σημεία που βρίσκονται κοντά στον πελάτη εσκεμένα δεν περιγράφονται επακριβώς, καθώς μπορούν να είναι οποιαδήποτε σημείο στο δίκτυο που μπορεί να τοποθετηθεί μια cache: στον browser ενός χρήστη, ένα forward-proxy ενός CDN (Content Delivery Network) ή ένα reverse-proxy αμέσως μετά τον server που φιλοξενεί την εφαρμογή. Αυτό ισχύει γιατί η ESI δεν είναι μια αρχιτεκτονική caching, αλλά μια γλώσσα περιγραφής μεταπληροφορίας

απαραίτητης για το caching δυναμικού περιεχομένου, η οποία μπορεί να ολοκληρωθεί πλήρως με μια οποιαδήποτε αρχιτεκτονική ή σύστημα caching, προκειμένου αυτό να γίνει ικανό να αποθηκεύει τους πόρους σε επίπεδο τμήματος HTML.

Η ESI είναι μια ανοιχτή γλώσσα, με την έννοια ότι οι δημιουργοί της επιτρέπουν ελεύθερα τη χρήση της από όποιον επιθυμεί να την χρησιμοποιήσει. Μάλιστα, μετά την ανάπτυξη της από τις δύο μεγάλες αυτές εταιρείες, η ESI καταχωρήθηκε και ως προτεινόμενο standard στο Wolrd Wide Web Consortium (W3C) [ESI W3C submission], με σκοπό να γίνει ένα πρότυπο που θα χρησιμοποιείται γενικά και όχι μονάχα στα πλαίσια εμπορικών λύσεων που προσφέρονται από τις εταιρείες αυτές (ή και όσες ασπάζονται αυτή την τεχνική).

Οι λεπτομέρειες της ESI περιγράφονται στο [ESI Technical Specs] και θα παρουσιαστούν συνοπτικά εδώ. Αρχικά παρατίθεται ένα παράδειγμα:

```
1. <HTML>
2. <!--esi
3.   <H3>Stock quote for ${QUERY_STRING}</H3>
4.   <esi:try>
5    <esi:attempt>
6.      <esi:include src=/quote.html
7.                alt=/delayed_quote.html/>
8.      <esi:choose>
9.        <esi:when
10.          test="$(HTTP_COOKIE{Type})==premium">
11.         <esi:include src=/market_news.html/>
12.       </esi:when>
13.       <esi:otherwise>
14.         To subscribe to premium services
15.          <A href=/subscribe.html> click here </A>
16.       </esi:otherwise>
17.     </esi:choose>
18.   </esi:attempt>
19.   <esi:except>
20.     <esi:include src=/sorry.html />
21.   </esi:except>
22.  </esi:try>
23. -->
24. <esi:remove>
25.    Please click on a
26.    <A href=/non_esi.html> non-ESI version </A> of this site
27. </esi:remove>
</HTML>
```

Τα <!—esi και <esi:remove> επιτρέπουν στα templates, που δημιουργούνται για αρχιτεκτονικές που χρησιμοποιούν ESI, να χειρίζονται σωστά από πελάτες και proxies που δεν υποστηρίζουν ESI. Αυτό γίνεται ως εξής: στην HTML ότι περικλείεται σε <!-- και --> θεωρείται σχόλιο και δεν εμφανίζεται στην οθόνη του χρήστη. Έτσι, σε έναν browser που δεν υποστηρίζει ESI οι γραμμές 2-23 θα θεωρηθούν σχόλια και δεν θα εμφανιστούν καθόλου στο παράθυρο του προγράμματος πλοήγησης. Επίσης, πέρα από τα σχόλια, οι άγνωστες ετικέτες <esi:remove> θα αγνοηθούν από τον browser (αυτή είναι η προκαθορισμένη συμπεριφορά των περισσότερων προγραμμάτων πλοήγησης όταν αυτά συναντήσουν κάποια ετικέτα της οποίας την σημασία δεν γνωρίζουν). Με αυτόν τον τρόπο θα εμφανιστεί στην οθόνη μόνο το περιεχόμενο της άγνωστης ετικέτας που είναι έγκυρη HTML που κατευθύνει τον χρήστη σε μια άλλη έκδοση του πόρου για μη-ESI συστήματα. Αντίθετα, ένας ESI επεξεργαστής(ESI processor) θα αφαιρέσει τις ετικέτες <!-- --> και θα θεωρήσει τις ετικέτες <esi:remove> σαν σχόλια.

Όσον αφορά την επεξεργασία των γραμμών 3-21, το attempt μπλοκ (γραμμές 5-18) είναι αυτό που εκτελείται πρώτα. Η γραμμή 6 προσπαθεί να εισάγει ένα τμήμα (fragment) με σχετικό URL ”/quote.html”. Αν για κάποια αιτία το κατέβασμα (downloading) αυτού του τμήματος αποτύχει, το τμήμα “delayed_quote.html” θα δοκιμαστεί στη συνέχεια. Αν και αυτό το download αποτύχει, το except μπλοκ των γραμμών 19-21 εκτελείται.

Οι γραμμές 8-17 δίνουν ένα παράδειγμα υπο συνθήκης εισαγωγής με βάση την τιμή που έχει ένα cookie της αίτησης. Αν, λοιπόν, το cookie της αίτησης περιείχε μια παράμετρο με όνομα “Type” και τιμή “premium”, η γραμμή 11 θα εισήγαγε το “/market_news.html” τμήμα, αλλιώς θα εισαγόταν η πρόσκληση για εγγραφή στην αντίστοιχη υπηρεσία. Τα <esi:remove> tags καθοδηγούν τον ESI επεξεργαστή να αφαιρέσει το περικλειόμενο κείμενο από την τελική σελίδα.

Το παράδειγμα αυτό παρουσιάζει τις δυνατότητες της ESI για εισαγωγή τμημάτων HTML, υπό συνθήκη εκτέλεση και χειρσμό λαθών. Άλλες δυνατότητες είναι οι εμφωλευμένες ESI δομές και η πρόσβαση σε μεταβλητές περιβάλλοντος. Μεταγεννέστερες εκδόσεις της ESI επιτρέπουν επίσης τον ορισμό και την χρήση και άλλων μεταβλητών πέρα από τις μεταβλητές περιβάλλοντος.

Πιο συγκεκριμένα, τα στοιχεία (elements) της ESI είναι τα παρακάτω:

<esi:include src=”URI” alt=”URI” onerror=”continue” /> Η ετικέτα include εισάγει στην απάντηση τον πόρο που περιγράφει το URI που έχει το src attribute της ετικέτας. Αν αυτό δεν βρεθεί για κάποιο λόγο, ο ESI επεξεργαστής προσπαθεί να βρει το αντίστοιχο του alt attribute. Αν και εκείνο δεν βρεθεί, ο ESI επεξεργαστής επιστρέφει έναν HTTP κωδικό λάθους πάνω από το 400, εκτός και αν η ετικέτα έχει το χαρακτηριστικό onerror=”continue”, οπότε και απλά αγνοείται η εισαγωγή και συνεχίζεται η επεξεργασία του υπόλοιπου template.

<esi:inline name="URI" fetchable="{yes|no}">fragment to be stored within an ESI processor**</esi:inline>** Η ετικέτα inline παρέχει έναν τρόπο ώστε να διαχωρίζονται τα fragments μεταξύ τους όταν αυτά περικλείονται στο σώμα μιας HTTP απάντησης. Η χρήση της γίνεται για σκοπούς καλύτερης απόδοσης, ώστε να μην χρειάζονται πολλές διαφορετικές HTTP αιτήσεις για να προσκομηθούν τα διάφορα τμήματα. Ως εκ τούτου, η υλοποίηση αυτής της ετικέτας από τα ESI συστήματα είναι προαιρετική. Τα inline fragments αποθηκεύονται και χρησιμοποιούνται από τον ESI επεξεργαστή με τον ίδιο τρόπο που χρησιμοποιούνται τα include τμήματα.

choose | when | otherwise : Η ESI παρέχει την δυνατότητα για υπό συνθήκη λογική με αυτές τις ετικέτες. Η χρήση τους είναι η εξής:

```
<esi:choose>
   <esi:when test="…">
      …
   </esi:when>
   <esi:when test="…">
      …
   </esi:when>
   <esi:otherwise>
      …
   </esi:otherwise>
</esi:choose>
```

Η ετικέτα choose πρέπει να περιέχει τουλάχιστον ένα στοιχείο when, ενώ προαιρετικά μπορεί να υπάρχει το πολύ ένα στοιχείο otherwise. Η εκτέλεση που ακολουθείται είναι η συνηθισμένη από τις διαδικαστικές γλώσσες προγραμματισμού.

try | attempt | except : Η τριάδα αυτή των ετικετών επιτρέπει τον χειρισμό λαθών κατά την επεξεργασία των ESI templates. Η δομή είναι παρόμοια με εκείνη της υπό συνθήκη λογικής:

```
<esi:try>
   <esi:attempt>
      …
   </esi:attempt>
   <esi:except>
      …
   </esi:except>
</esi:try>
```

Αντίθετα με την ετικέτα choose, η try πρέπει να περιέχει ακριβώς ένα στοιχείο attempt και ένα στοιχείο except. Ο ESI επεξεργαστής εκτελεί αρχικά το μπλοκ που περιέχεται μέσα στην ετικέτα attempt. Αν κάποια <esi:include> δήλωση αποτύχει μέσα στο σώμα της attempt,τότε εκτελείται η except (μόνο οι include δηλώσεις μπορούν να προκαλέσουν την εκτέλεση του except).

<esi:comment text="…" /> Η ετικέτα comment χρησιμοποιείται για την εισαγωγή σχολίων από τους προγραμματιστές των εφαρμογών ESI, χωρίς αυτά να εμφανίζονται στην έξοδο του ESI επεξεργαστή.

<esi:remove> Η remove επιτρέπει τον ορισμό μη-ESI εξόδου, στην περίπτωση που δεν υποστηρίζεται η χρήση των ESI templates, με τον τρόπο που περιγράφηκε στο παράδειγμα.

<esi:vars>…</esi:vars> Η vars χρησιμοποιείται για την χρήση ESI μεταβλητών εκτός από ένα ESI μπλοκ. Οι ESI μεταβλητές που υπάρχουν είναι οι ακόλουθες:

ESI Variables

Variable Name	HTTP Header	Substructure Type	Example
HTTP_ACCEPT_LANGUAGE	Accept-Language	list	da, en-gb, en
HTTP_COOKIE	Cookie	dictionary	id=571; visits=42
HTTP_HOST	Host	-	esi.xyz.com
HTTP_REFERER	Referer	-	http://roberts.xyz.com/
HTTP_USER_AGENT	User-Agent	dictionary (special)	Mozilla; MSIE 5.5
QUERY_STRING	-	dictionary	first=Robin&last=Roberts

2.3 – Βελτίωση Απόδοσης με την ESI

Σε αυτήν την ενότητα θα παρουσιαστεί ένα παράδειγμα από το [OracleWebCache 2002], το οποίο δείχνει πόσο μεγάλη μπορεί να είναι η βελτίωση της απόδοσης ενός συστήματος caching με τη χρήση των Edge Side Includes, ενώ επίσης θα μελετηθούν και οι συνθήκες κάτω από τις οποίες η χρήση μιας ESI αρχιτεκτονικής ωφελεί ένα σύστημα caching και ποιος είναι η παγίδα στη χρήση των ESI που πρέπει να αποφευχθεί.

Θεωρούμε μια απλή δυναμική εφαρμογή που παρέχει στον χρήστη μια εξατομικευμένη σελίδα που αποτελείται από τις τιμές πέντε μετοχών, τριών ειδησεογραφικών τίτλων, τριών αποτελεσμάτων αθλητικών γεγονότων και την πρόβλεψη του καιρού για μια πόλη. Ο αριθμός και η μεταβλητότητα των δεδομένων παρουσιάζονται στον πίνακα:

Content	Number of Objects	Time to Live
Stock Quotes	10,000 Securities	15 minutes
Weather	1,000 Cities	One Hour
Sports	500 Teams	One Hour
News	50 Topics	One Hour

Αυτή η εφαρμογή είναι μια συλλογή από JSPs και EJBs (Enterprise Java Beans). Τα JSPs είναι αυτά που παράγουν την HTML της απάντησης και τα EJBs είναι εκείνα που παρέχουν μια απλή μέθοδο προσπέλασης για τα δεδομένα. Οι μετρήσεις που περικλείονται στο [OracleWebCache 2002] έγιναν σε έναν υπολογιστή 933MHz Intel Pentium με λειτουργικό σύστημα LINUX, ο οποίος έτρεχε και την εφαρμογή και την Web Cache που υποστηρίζει τα ESI. Χωρίς να λειτουργεί η Web Cache, το σύστημα είχε την δυνατότητα να παράγει 50 σελίδες το δευτερόλεπτο. Αντίθετα, με την Web Cache, μετρήθηκε να παράγει ακόμα και 2140 σελίδες το δευτερόλεπτο (κάνονται χρήση 12 ESI τμημάτων στα οποία χωρίστηκαν οι σελίδες της

εφαρμογής). Στον παρακάτω πίνακα παρουσιάζονται οι αιτήσεις και οι χρόνοι απόκρισης για τις δύο περιπτώσεις της εφαρμογής με ή χωρίς την Web Cache:

	Web Cache	Application
Without Web Cache		1,000,000 JSP 1,000,000 Template EJB 5,000,000 Stock EJB 1,000,000 Weather EJB 3,000,000 Sports EJB 3,000,000 News EJB *Requires 20,000 cpu seconds*
With Web Cache	1,000,000 Pages *Requires 468 cpu seconds*	41,550 JSP w/JESI 0 Template EJB 40,000 Stock EJB 1,000 Weather EJB 500 Sports EJB 50 News EJB *Requires <831 cpu seconds*

Με την χρήση μιας τεχνικής δυναμικής συναρμολόγησης σελίδας, όπως εδώ της χρήσης ESI templates, η Web Cache εμφανίζει πολύ μεγαλύτερα ποσοστά επιτυχίας (hit rates) και βελτιώνει σημαντικά την απόδοση της εφαρμογής. Στο συγκεκριμένο παράδειγμα, επιτεύχθηκε πάνω από 15 φορές καλύτεροι χρόνοι απόκρισης, σε σύγκριση με το να μην υπήρχε καθόλου cache (ούτε και κάποια cache στατικού περιεχομένου).

Βέβαια, τα πράγματα δεν τόσο ιδανικά όσο φαίνονται από αυτό το πείραμα. Χρειάζεται πάντα προσοχή στην απόφαση για το αν θα γίνει χρήση κάποιας αρχιτεκτονικής ESI. Αυτό συμβαίνει διότι μπορεί να αυξάνονται τα ποσοστά επιτυχίας της cache, ως συνέπεια του caching στατικών τμημάτων μιας κατά τα άλλα δυναμικής και uncacheable σελίδας, αλλά αυτό δεν γίνεται χωρίς κόστος. Το κόστος είναι ότι μετά την χρήση των templates αυξάνονται οι αιτήσεις που αποστέλλονται στον Web Server για μια σελίδα. Συγκεκριμένα, ενώ πριν με μια αίτηση στον server επιστρεφόταν το περιεχόμενο ολόκληρης της σελίδας, μετά τον διαχωρισμό της σελίδας σε τμήματα χρειάζονται τόσες αιτήσεις, όσα και τα τμήματα στα οποία χωρίστηκε η σελίδα.

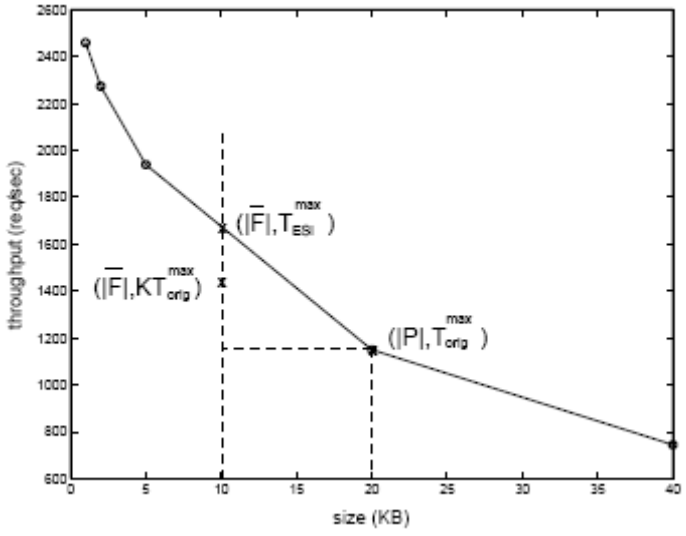
Αυτή την παρατήρηση κάνουν οι συγγραφείς του [Rabinovich et al. 2003], οι οποίοι και μελέτησαν διεξοδικά το ζήτημα και πρότειναν ένα τρόπο απόφασης για το αν τελικά αρμόζει η χρήση των ESI στην εκάστοτε περίπτωση. Αυτά που επισημαίνουν στο άρθρο τους είναι συνοπτικά αυτά:

Η απάντηση για το αν θα πρέπει να χρησιμοποιηθούν τα ESI εξαρτάται από το αν μια δυναμική σελίδα μπορεί να διασπαστεί με τέτοιο τρόπο ώστε τα επιμέρους τμήματα (ή κάποια από αυτά) να έχουν αρκετά μεγαλύτερο TTL (Time To Live) από την αρχική σελίδα, και επίσης από το αν μετά την διάσπαση, ο Web Server θα εμφανίζει αποτελεσματικότερη χωρητικότητα, όπου με τον όρο χωρητικότητα εννοείται ο αριθμός των αιτήσεων που μπορεί να σηκώσει ο server σε συνάρτηση με τη συνολική πληροφορία που μεταδίδει.

Για να γίνει κατανοητό αυτό, ας θεωρηθεί ένα υποθετικό παραδείγμα στο οποίο η χρήση των ESI μπορεί να είναι καταστροφική για την απόδοση ενός server. Έστω, λοιπόν, μια σελίδα μεγέθους 20K που έχει τρία τμήματα μεγέθους 2K το κάθε ένα, τα οποία αλλάζουν κάθε λεπτό. Η υπόλοιπη σελίδα είναι σχετικά στατική και μπορεί να αποθηκευτεί με καλά αποτελέσματα στην cache. Σύμφωνα με την λογική των ESI αρχιτεκτονικών, η συγκεκριμένη σελίδα αποτελεί έναν πολύ καλό υποψήφιο για χρήση ESI. Στην περίπτωση που αποφασιστεί η χρήση δυναμικής συναρμολόγησης σελίδας με ESI, με κάθε αίτηση που στέλνει ένας πελάτης για αυτή την σελίδα, στον server θα φτάνουν από τον ESI proxy 3 ή 4 αιτήσεις(4 για την πρώτη φορά που θα ζητηθεί η σελίδα και δεν έχει φυλαχθεί ακόμα το στατικό τμήμα της) μια για κάθε τμήμα της σελίδας. Παράλληλα βέβαια, θα μειωθεί το ποσό της συνολικής πληροφορίας που θα πρέπει να στείλει πίσω ο server, καθώς στην συνήθη περίπτωση θα στέλνει 3*2K=6K αντί των 20K που είναι η απάντηση όταν δεν χρησιμοποιείται ESI. Συνεπώς, υπάρχει μια τριπλάσια (20/6=3.3) μείωση του εύρους ζώνης του server, ενώ παράλληλα και τριπλάσια αύξηση των αιτήσεων (αφού αντί για μία θα καταφθάνουν 3 αιτήσεις). Κάτω από αυτές τις συνθήκες, ο συμπερασμός για το αν η χρήση των ESI είναι ωφέλιμη δεν είναι πάντα εύκολη υπόθεση και εξαρτάται σε μεγάλο βαθμό και από τον φόρτο που εισάγει στον server η άφιξη μια νέας αίτησης.

Η μεθοδολογία που προτείνεται για την εξακρίβωση του αν είναι ωφέλιμη η χρήση των ESI είναι η ακόλουθη:

1) Stretch-Testing of Server: αρχικά πρέπει να διενεργηθεί μια δοκιμασία της αντοχής του εξυπηρέτη για να βρεθεί πως εξαρτάται ο ρυθμός εξυπηρέτησης αιτήσεων (αριθμός αιτήσεων ανά δευτερόλεπτο) από το μέγεθος των απαντήσεων. Με βάση τα αποτελέσματα αυτού του test μπορεί να κατασκευαστεί το παρακάτω διάγραμμα:



όπου η συμπαγής γραμμή δείχνει τον αριθμό αιτήσεων που μπορεί να εξυπηρετήσει ο server για δεδομμένο μέγεθος των απαντήσεων που αντιστοιχούν σε αυτές. Η γραμμή αυτή θα καλείται καμπύλη χωρητικότητας (capacity curve).

2) Μετά την κατασκευή της καμπύλης, ας θεωρηθεί η σελίδα μεγέθους |P| για την οποία πρέπει να αποφασιστεί η χρήση των ESI. Πρώτα πρέπει να υπολογιστεί το μέσο μέγεθος των ESI τμημάτων |F| και η αύξηση του ρυθμού άφιξης αιτήσεων K. Δηλαδή, αν T_{orig} είναι ο ρυθμός άφιξης για την αρχική σελίδα, τότε το K* T_{orig} θα είναι ο ρυθμός μετά την διάσπαση της σελίδας σε τμήματα.

3) Δεδομένων των μεγεθών |P| και |F|, κατασκευάζονται στο διάγραμμα δύο κατακόρυφες γραμμές στα δύο αντίστοιχα σημεία και υπολογίζονται τα σημεία τομής τους με την καμπύλη χωρητικότητας. Στο σχήμα αυτά είναι (|P|,$Torig_{max}$) και (|F|,$Tesi_{max}$).

4) Ο ρυθμός άφιξης αιτήσεων στον server μετά την εφαρμογή των ESI θα είναι T_{after}=K*$Torig_{max}$, αφού έτσι ορίστηκε το K. Η απόφαση, συνεπώς, για την χρήση των ESI έγκειται στο αν T_{after}< $Torig_{max}$: αν ισχύει η ανισότητα, τότε είναι συμφέρον για την απόδοση της εφαρμογής να χρησιμοποιηθεί δυναμική συναρμολόγηση σελίδας με ESI.

Αυτό που μένει να εξεταστεί είναι ο τρόπος υπολογισμού των |F| και K. Έστω ότι το μέγεθος της σελίδας είναι |P|, και αυτή αποτελείται από n τμήματα (fragments) $F_1,F_2,\ldots,F_i,\ldots,F_n$ με κάθε τμήμα i να έχει χρόνο ζωής (lifetime) t_i και μέγεθος $|F_i|$. Ακόμα, έστω t_m ο μικρότερος χρόνος ζωής από τους n και ότι t_m>0. Τότε, είναι προφανές ότι, ενώ πριν ο ρυθμός άφιξης αιτήσεων ήταν $1/t_m$ ο νέος ρυθμός άφιξης αιτήσεων θα είναι $\sum_{i=1}^{n}(1/t_i)$. Αυτό αιτιολογείται από το γεγονός ότι, χωρίς ESI ο proxy server έστελνε μία αίτηση κάθε φορά που η σελίδα ξεπερνούσε το TTL στην cache, δηλαδή κάθε t_m δευτερόλεπτα. Επίσης, με την χρήσης ESI, ο αριθμός των αιτήσεων αυξάνεται καθώς ο proxy στελνει μια αίτηση για κάθε τμήμα που τελειώνει ο χρόνος ζωής του και με αυτόν τον τρόπο προκύπτει το άθροισμα. Εξ' ορισμού, το K είναι ο λόγος των δύο ρυθμών αφίξεων, δηλαδή:

$$K = \frac{\sum_{i=1}^{n}(1/t_i)}{1/t_m} = [\sum_{i=1}^{n}(1/t_i)] * t_m$$

Όμοια, ο ρυθμός μετάδοσης δεδομένων από τον server στον proxy είναι:

$$\sum_{i=1}^{n}(|F_i|/t_i)$$

Και το μέσο μέγεθος των απαντήσεων:

$$|F| = \frac{\sum_{i=1}^{n} (|F_i| / t_i)}{\sum_{i=1}^{n} (1 / t_i)}$$

Μέσα από όλα αυτά τα παραδείγματα και τις εξισώσεις έγινε κατανοητό πως η χρήση μιας τεχνικής δυναμικής συναρμολόγησης σελίδας μπορεί να βελτιώσει θεαμετικά την απόδοση ενός σχήματος δυναμικού caching, πάντα όμως με τον κίνδυνο η επιλογή αυτή, αντί να βοηθήσει, να δημιουργήσει επιπλέον προβλήματα στη λειτουργία του συστήματος. Στη συνέχεια θα παρουσιαστούν ορισμένα μειονεκτήματα του ESI και πως διάφορες ερευνητικές εργασίες προτείνουν την επίλυσή τους.

2.4 – Αυτοματοποίηση της διάσπασης σε τμήματα

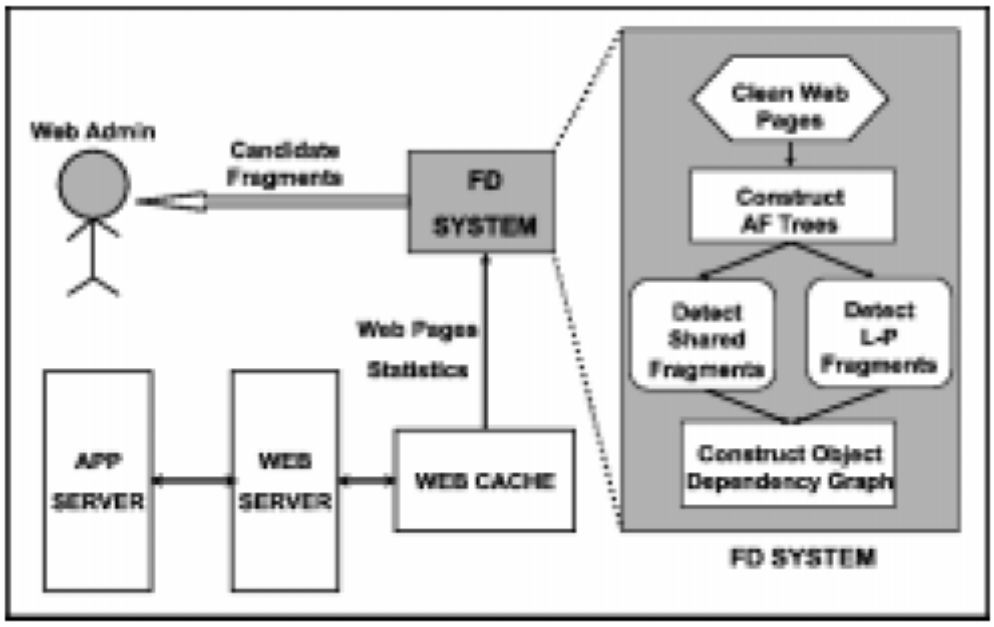
Ένα από τα βασικότερα προβλήματα της προσέγγισης της δυναμικής συναρμολόγησης σελίδας (με κύριο εκπρόσωπο τα ESI) είναι ότι απαιτεί την αλλαγή του κώδικα της εφαρμογής, ώστε κάθε σελίδα να διασπαστεί στα τμήματα που θα γίνουν ξεχωριστά caching. Αυτό σημαίνει ότι είτε ο προγραμματιστής που φτιάχνει την εφαρμογή, είτε ο διαχειριστής του δικτύου στο οποίο φιλοξενείται η εφαρμογή, πρέπει να προσδιορίσουν τα τμήματα με τρόπο χειρονακτικό (manually). Όπως γίνεται εύκολα αντιληπτό, η διαδικασία αυτή είναι εξαιρετικά χρονοβόρα και επιρεπής σε λάθη, ώστε να είναι απαγορευτικό στις περισσότερες εφαρμογές που δεν ήταν αρχικά σχεδιασμένες σε ESI αρχιτεκτονική να μεταβούν σε αυτή.

Μια λύση σε αυτό το μειονέκτημα των ESI είναι η πρόταση των [Ramaswamy et al. 2004]. Σε αυτή την εργασία παρουσιάζεται ένας αυτοματοποιημένος τρόπος διάσπασης μιας σελίδας σε τμήματα τα οποία εμφανίζουν τις ιδιότητες που χρειάζονται ώστε η χρήση μιας ESI caching αρχιτεκτονικής να έχει τα καλύτερα αποτελέσματα. Οι ιδιότητες που πρέπει να πληροί ένα τμήμα είναι είτε να διαμοιράζεται μεταξύ πολλών σελίδων, ώστε η αποθήκευση να γίνεται μία φορά και η χρήση σε πολλές σελίδες, είτε να είναι ένα μικρό τμήμα που εμφανίζει μικρό χρόνο ζωής (lifetime), και το οποίο αν διαχωριστεί θα ελευθερώσει την υπόλοιπη σελίδα από την υποχρέωση να ενημερώνεται κάθε φορά που το τμήμα αυτό ξεπερνά τον χρόνο ζωής του.

Για την αυτοματοποίηση της διαδικασίας, γίνεται η παρουσίαση δύο αλγορίθμων, ένας για την ταυτοποίηση τμημάτων που διαμοιράζονται μεταξύ Μ σελίδων (Shared Fragment Detection Algorithm - SFDA) και ένας για την αναγνώριση τμημάτων που εμφανίζουν διαφορετικά χαρακτηριστικά χρόνου ζωής και εξατομίκευσης, με βάση τα διαφορετικά στιγμιοτύπα της ίδιας σελίδας (Lifetime-Personalization based (L–P) Fragment Detection – L-PFDA). Προκειμένου να γίνει αποδοτικά η σύγκριση μεταξύ των διαφορετικών σελίδων, εισάγεται μια νέα δομή δεδομένων για την αναπαράσταση XML

εγγράφων, η οποία με το να κρατά κάποια βοηθητικά δεδομένα κάνει εφικτή την γρήγορη συγκριση εγγράφων.

Η δομή ενός συστήματος αυτόματης αναγνώρισης τμημάτων (Automatic Fragment Detection System) έχει ως εξής:



FD System: Fragment Detection System Architecture

Ο σκοπός του συστήματος είναι να εντοπίσει και να παρουσιάσει στον διαχειριστή του συστήματος μια λίστα με τα τμήματα που είναι καταλληλότερα για την διάσπαση των σελίδων σε μια αρχιτεκτονική caching δυναμικής συναρμολόγησης σελίδας. Για να γίνει αυτό το Fragment Detection (FD) σύστημα δέχεται σαν είσοδο από την cache και τον web server τις σελίδες της εφαρμογής καθώς επίσης και κάποια στατιστικά που αφορούν το μέγεθος και του ρυθμού ζήτησης σελίδων και τμημάτων, τα οποία βοηθούν τον διαχειριστή να αποφασίσει τελικά για την χρήση ή όχι των ESI. Αυτό καθ' εαυτό το σύστημα FD σε πρώτο βήμα διαβάζει τις δυναμικές σελίδες που δέχεται στην είσοδο και κατασκευάζει ένα Augmented Fragment Tree (AF Tree) για κάθε μια σελίδα, όπου AF Tree είναι η δομή δεδομένων που αναφέρθηκε προηγουμένως. Στη συνέχεια, τρέχει τους δύο αλγορίθμους εντοπισμού τμημάτων (SFDA και L-PFDA) προκειμένου να βρεθούν τα υποψήφια τμήματα στις δεδομένες δικτυακές σελίδες. Τέλος, κατασκευάζεται ένας ODG (Object Dependency Graph) γράφος για κάθε σελίδα, ο οποίος περιέχει την πληροφορία για τις συσχετίσεις μεταξύ των τμημάτων ενός δικτυακού τόπου, ο οποίος σε συνδιασμό με τα στατιστικά δίνει στον διαχειριστή μια ολοκληρωμένη είκόνα για κάθε προτεινόμενο τμήμα, ώστε εκείνος τελικά να αποφασίσει την ενεργοποίηση της τμηματοποίησης που του προτάθηκε από το αυτόματα σύστημα.

Η δομή των Augmented Fragment Trees είναι μια εναλλακτική της δομής DOM (Document Object Model) που είναι η πιο διαδεδομένη δομή αναπαράστασης ενός εγγράφου. Η ανάγκη ανάπτυξης μιας νέας δομής έγκειται στο ότι η δομή DOM δεν είναι αποτελεσματική για τον εντοπισμό

τμημάτων για δύο λόγους. Πρώτον, πολλοί από τους κόμβους της δεντροειδούς δομής DOM αντιστοιχούν σε στοιχεία που δεν συμμετέχουν στην σύγκριση των εγγράφων, όπως για παράδειγμα οι ετικέτες μορφοποίησης κειμένου. Έτσι, μια δομή DOM έχει μεγαλύτερο μέγεθος από μια δομή AF Tree με άμεσο αποτέλεσμα την λιγότερο αποδοτική σύγκριση των εγγράφων. Δεύτερος λόγος που τα AF Trees αρμόζουν καλύτερα για την διαδικασία του εντοπισμού των τμημάτων είναι ότι οι κόμβοι του DOM δεν περιέχουν όλη την απαραίτητη πληροφορία για την αποδοτική σύγκριση εγγράφων.

Οι λόγοι αυτοί οδήγησαν τους συγγραφείς της δημοσίευσης αυτής να αναπτύξουν τα AF Trees. Τα AF Trees είναι μια ιεραρχική δομή αναπαράστασης δικτυακών εγγράφων (HTML ή XML). Πρώτα από όλα, ένα AF Tree είναι ένα συμπυκνωμένο DOM δέντρο από το οποίο έχουν αφαιρεθεί όλες οι μη απαραίτητες ετικέτες (όπως για παράδειγμα οι text-formatting tags). Δεύτερον, από το περιεχόμενο κάθε κόμβου του δέντρου παράγεται μια shingles κωδικοποίηση που χρησιμοποιείται για την συγκριση των τμημάτων, και η οποία κωδικοποίηση αποθηκεύεται στον κόμβο για να μην επιβαρύνονται οι αλγόριθμοι με τον επαναλαμβανόμενο υπολογισμό της. Τέλος, σε κάθε κόμβο υπάρχουν και πεδία με βοηθητική πληροφορία. Συνολικά, τα πεδία που υπάρχουν σε κάθε κόμβο είναι: ένας αριθμός (NodeID), η τιμή του κόμβου (NodeValue), η συνένωση όλων των τιμών του υποδέντρου του οποίου ο συγκεκριμένος κόμβος είναι η ρίζα (SubtreeValue), το μέγεθος του υποδέντρο (Subtree Size) και η shingles κωδικοποίηση του υποδέντρου (SuubtreeShingles).

Για λόγους πληρότητας, να αναφερθεί ότι η κωδικοποίηση shingles, είναι μια παρόμοια μέθοδος με την MD5, η οποία όμως παρουσίαζει την αντιθετη από την MD5 ιδιότητα να αλλάζει λίγο για μικρές αλλαγές της κωδικοποιούμενης ποσότητας. Αυτή η ιδιότητα της shingles κωδικοποίησης την έχουν καταστήσει ως μια συνηθισμένη μέθοδο για τον υπολογισμό της ομοιότητας εγγράφων [Broder 1997].

Στη συνέχεια θα γίνει μια κατά το δυνατό συνοπτική΄παρουσίαση των αλγορίθμων εντοπισμού τμημάτων.

Shared Fragment Detection Algorithm

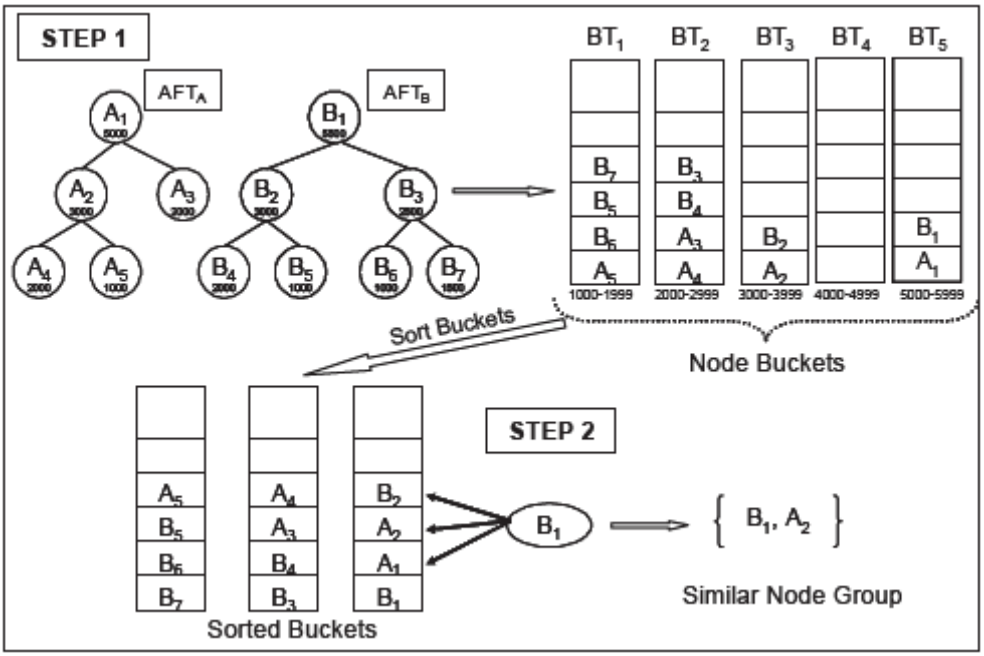
Για την ανάπτυξη ενός αλγορίθμου εντοπισμού τμημάτων που είναι κοινά μεταξύ διαφορετικών σελίδων είναι ανάγκη να αντιμετοπιστούν κατάλληλα δύο θεμελιώδη θέματα: πρώτον, πρέπει να οριστούν οι μετρικές που θα καθορίζουν το αν κάποια τμήματα μπορούν να θεωρηθούν κοινά και δεύτερον, να ακολουθείται μια αποδοτική και κλιμακώσιμη στρατηγική υλοποίησης για την σύγκριση των σελίδων.

Ο συγκεκριμένος αλγόριθμος ορίζει τρεις μετρικές, οι οποίες δίνονται σαν παράμετροι πριν την εκτέλεση του αλγορίθμου, ώστε αυτός να εκτελείται προσαρμοσμένος στην εκάστοτε εφαρμογή. Η ακρίβεια και η απόδοση του αλγορίθμου εξαρτόνται από τις τιμές αυτών των παραμέτρων. Οι παράμετροι είναι:

- **Minimum Fragment Size (MinFragSize):** καθορίζει το ελάχιστο μέγεθος που πρέπει να έχει το εντοπιζόμενο τμήμα. Η παράμετρος αυτή προστατεύει τον αλγοριθμο από το να διασπάσει μια σελιδα σε πολλά και πολύ μικρά τμήματα, κάτι που μπορεί να επιβαρύνει την απόδοση του συστήματος caching για τους λόγους που έχουν αναφερθεί (αυξημένος αριθμός αιτήσεων).
- **Sharing Factor (ShareFactor):** καθορίζει τον ελάχιστο αριθμό σελίδων που πρέπει να εχουν κοινό ένα κομμάτι τους για να χαρακτηριστεί αυτό ως τμήμα των σελίδων αυτών.
- **Minimum Matching Factor (MinMatchFactor):** το ελάχιστο ποσοστό ομοιότητας που πρέπει να έχουν δύο τμήματα (τα πεδία SubtreeShingles τους) για να θεωρηθούν κοινά σε δύο διαφορετικές σελίδες.

Η υλοποίηση του αλγορίθμου είναι αρκετά πολύπλοκη, και περιλαμβάνει δύο βήματα:

Στο πρώτο βήμα, οι κόμβοι των AF Trees που έχουν κατασκευαστεί τοποθετούνται σε μια δεξαμενή (pool) ταξινομημένων κουβάδων (buckets). Η διαδικασία ξεκινά με την δημιουργία N buckets στα οποία τοποθετούνται οι κόμβοι, ανάλογα με το μέγεθός τους (SubtreeSize). Έτσι, κάθε bucket έχει ένα συγκεκριμένο εύρος μεγεθών το οποίο μπορεί να δεχθεί, όπως φαίνεται στο παράδειγμα. Έπειτα, οι κόμβοι ταξινομούνται εντός των buckets με βάση το πεδίο τους SubtreeSize. Επειδή το να είναι όσο το δυνατόν πιο ομοιόμορφη η κατανομή βοηθά στην αποδοτικότερη εκτέλεση του αλγορίθμου, μετά την τοποθέτηση των κόμβων και την ταξινόμησή τους στα buckets γίνεται συγχώνευση μεταξύ γειτονικών buckets που έχουν λίγους κόμβους (βλ. παρακάτω διάγραμμα).



Στο δεύτερο βήμα, γίνεται η αναγνώριση των κοινών τμημάτων με την ομαδοποίηση των όμοιων κόμβων. Αρχίζοντας, λοιπόν, από τον κόμβο με το

μεγαλύτερο μέγεθος, επεξεργάζονται όλοι οι κόμβοι. Στην επεξεργασία κάθε κόμβου περιλαμβάνεται η σύγκρισή του με όλους τους κόμβους του bucket που έχουν μέγεθος μεγαλύτερο από ένα ποσοστό P% του κόμβου αυτού:

$$CSet(A_i) = \{A_j \mid SubtreeSize(A_j) \geq \frac{P \times SubtreeSize(A_i)}{100}\}$$

όπου CSet() είναι το σύνολο των κόμβων με τους οποίους θα γίνει η σύγκριση.

Η επιλογή του P καθορίζει τη σχέση μεταξύ της ακρίβειας των αποτελεσμάτων και της απόδοσης του αλγορίθμου. Αν το P είναι μικρό, αυξάνεται ο αριθμός των συγκρίσεων που γίνονται και επιβαρύνεται η απόδοση του αλγορίθμου. Αντίθετα, αν το P είναι μεγάλο γίνονται λιγότερες συγκρίσεις, αλλά υπάρχει περίπτωση να διαφύγουν κάποια όμοια τμήματα από τον αλγόριθμο.

Αν στην σύγκριση που γίνεται μεταξύ δύο κόμβων αυτοί έχουν όμοια shingles κωδικοποίηση κατά ένα ποσοστό άνω του MinMatchFactor τότε οι κόμβοι κατηγοριοποιούνται στην ίδια ομόδα, όπως διακρίνεται η ομάδα του παραδείγματος. Αν στο τέλος της διαδικασίας μια ομάδα έχει τουλάχιστον ShareFactor κόμβους, τότε αναγνωρίζεται σαν τμήμα για διάσπαση.

Lifetime-Personalization based Fragment Detection Algorithm
Για την ανακάλυψη των L-P τμημάτων συγκρίνονται διαφορετικές εκδόσεις της ίδιας σελίδας και σημειώνονται οι αλλαγές που συμβαίνουν μεταξύ των εκδόσεων. Για να δώσει ποιοτικά αποτελέσματα αυτή η διαδικασία πρέπει να αντιμετοπισθούν δύο θέματα που σχετίζοντια με τον εντοπισμό L-P τμημάτων. Πρώτον, πρέπει να ληφθεί υπόψη ότι δεν είναι αρκετό να συγκριθούν οι κόμβοι των δύο εκδόσεων του εγγράφου σύμφωνα με την απόλυτη θέση τους μέσα σε αυτό. Αυτό συμβαίνει διότι οι αλλαγές που συμβαίνουν σε ένα έγγραφο είναι ποικίλων μορφών, όπως διαγραφή, προσθήκη ή αλλαγή θέσης του τμήματος. Το δεύτερο ζήτημα είναι ότι πρέπει να επιλεχθούν τα υποψήφια τμήματα που είναι τα περισσότερο ωφέλιμα για caching, και η επιλογή αυτή γίνεται με βάση την συχνότητα και την ποσότητα των αλλαγών που έχει το τμήμα σε σχέση και με τις αντίστοιχες τιμές για τα υποτμήματά του που είναι και αυτά υποψήφια τμήματα.

Ο αλγόριθμος LPFDA, προκειμένου να εντοπίσει τα διαφορετικά ήδη των αλλαγών, εισάγει ένα ακόμα πεδίο στη δομή των AF Trees που λέγεται NodeStatus και παίρνει μια από τις τιμές {UnChanged, ValueChanged, PositionChanged}. Ακόμα, για να επιλέξει τα ποιοτικότερα, σε όρους αποτελεσματικότερου caching, L-P τμήματα, εισάγει τις παρακάτω παραμέτρους:

- **Minimum Fragment Size (MinFragSize):** το ελάχιστο μέγεθος για ένα επιλεγόμενο τμήμα,
- **Child Change Threshold (ChildChangeThreshold):** το ελάχιστο κλάσμα των παιδιών ενός κόμβου που πρέπει να αλλάξει η τιμή τους προκειμένου να χαρακτηριστεί ο κόμβος αυτός ότι έχει κατάσταση NodeStatus=ValueChanged.

Ο αλγόριθμος επεξεργάζεται AF Trees που προέρχονται από διαφορετικά στιγμιότυπα της ίδιας σελίδας. Αρχικά, εγκαθιστά την πρώτη έκδοση ως την βασική έκδοση (base version). Έπειτα, συγκρίνει όλες τις ακόλουθες εκδόσεις και ταυτοποιεί τα υποψήφια τμήματα, ενώ, επίσης, αλλάζει την βασική έκδοση κάθε φορά που η σελίδα αλλάζει ριζικά. Κάθε βήμα του αλγορίθμου γίνεται σε δύο φάσεις: στην πρώτη σημειώνεται το NodeStatus κάθε κόμβου και στη δεύτερη υπολογίζονται τα L-P τμήματα τα οποία στη συνέχεια συνδυάζονται σε μια ODG αναπαράσταση.

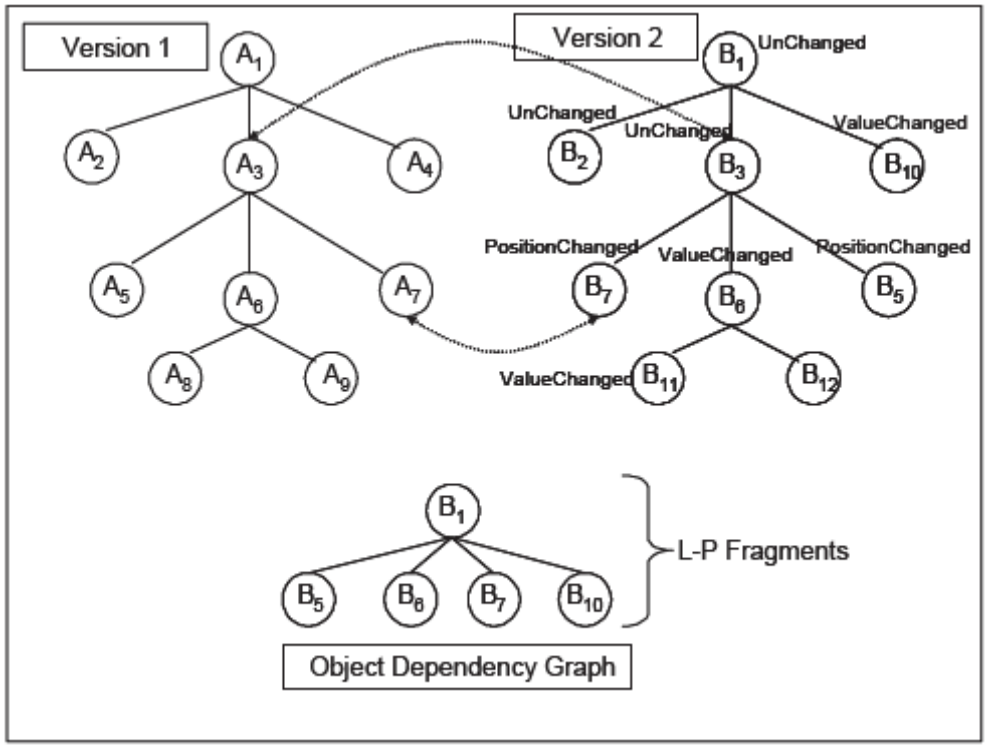
Φάση 1: Σύγκριση των AF Trees και εντοπισμός των αλλαγών.
Για τον εντοπισμό των αλλαγών και του είδους τους γίνεται χρήση μια συνάρτησης της οποίας η τιμή χαρακτηρίζει το ποσοστό ομοιότητας μεταξύ δύο κόμβων. Οι συγγραφείς ορίζουν την συνάρτηση αυτή ως:

$$ShingleSim(A_i, B_j) = \frac{SubtreeShingles(A_i) \cap SubtreeShingles(B_j)}{SubtreeShingles(A_i) \cup SubtreeShingles(B_j)}$$

όπου οι κόμβοι Α είναι από την βασική έκδοση και οι Β από την συγκρινόμενη έκδοση και η συνάρτηση SubtreeShingles() επιστρέφει την κωδικοποίηση του κόμβου, που δέχεται σαν είσοδο, σε δυαδική μορφή.

Τα κριτήρια που δείχνει το NodeStatus ενός κόμβου Β σε σχέση με την βασική έκδοση, είναι το παρακάτω: Αν στην επεξεργασία του Β κόμβου βρεθεί ο Α κόμβος ο οποίος έχει την μέγιστη ShingleSim(A,B) τιμή μεταξύ όλων των κόμβων του Α για τον συγκεκριμένο κόμβο του Β, και η συγκεκριμένη ShingleSim() τιμή είναι κάτω από το όριο OvlpThrshld (μια παράμετρος του συστήματος), τότε σημαίνει ότι κανένας κόμβος του Α δεν είναι αρκετά όμοιος του συγκεκριμένου κόμβου του Β και έτσι αυτός χαρακτηρίζεται με NodeStatus=ValueChanged. Αν βρεθεί ένας τέτοιος κόμβος, τότε συνεχίζεται περεταίρω η σύγκριση των κόμβων. Αν οι δύο κόμβοι έχουν ακριβώς τα ίδια πεδία SubtreeValue και NodeID, τότε το NodeStatus τους είναι UnChanged. Αντίθετα, αν διαφέρουν τα NodeIDs μονάχα, τότε σημειώνεται ότι ο κόμβος είναι PositionChanged. Εάν τα SubtreeValues των κόμβων διαφέρουν, τότε εξετάζεται αν είναι φύλλα του AF Tree. Αν ναι, σημειώνονται ως ValueChanged, αν όχι τότε εξετάζονται με την παραπάνω διαδικασία αναδρομικά τα παιδιά του κόμβου. Αν το κλάσμα των παιδιών του κόμβου που σημειώνονται ως ValueChanged είναι πάνω από το ChildChangeThreshold, τότε ο κόμβος χαρακτηρίζεται και αυτός σαν ValueChanged.

Το παρακάτω διάγραμμα δείχνει και την πρώτη και την δεύτερη φάση:



Στη δεύτερη φάση, ο αλγόριθμος διατρέχει ξανά τους κόμβους του δέντρου και βγάζει στην έξοδο τους κόμβους που είναι σημειωμένοι ως ValueChanged ή PositionChanged. Μια λεπτομέρεια είναι ότι ο αλγόριθμος δεν διατρέχει τα παιδιά ενός κόμβου που είναι σημειωμένος ως ValueChanged προκειμένου να παράγει μόνο τα τμήματα που έχουν το μεγαλύτερο μέγεθος.

Στο παράδειγμα του διαγράμματος φαίνεται πως σημειώθηκαν οι κόμβοι με το κατάλληλο NodeStatus, και ποιοι από αυτούς βγήκαν στην έξοδο. Συγκεκριμένα, φαίνεται πως ο κόμβος B_6 σημειώθηκε με ValueChanged παρότι δεν άλλαξε η τιμή, αλλά άλλαξαν τα δύο παιδιά του. Επίσης, οι κόμβοι B_{11} και B_{12} δεν βγαίνουν στην έξοδο αφού ο αλγόριθμος δεν διατρέχει τους κόμβους που είναι παιδιά ενός ValueChanged κόμβου.

2.5 – CSI: Μεταφέροντας τα ESI στους πελάτες

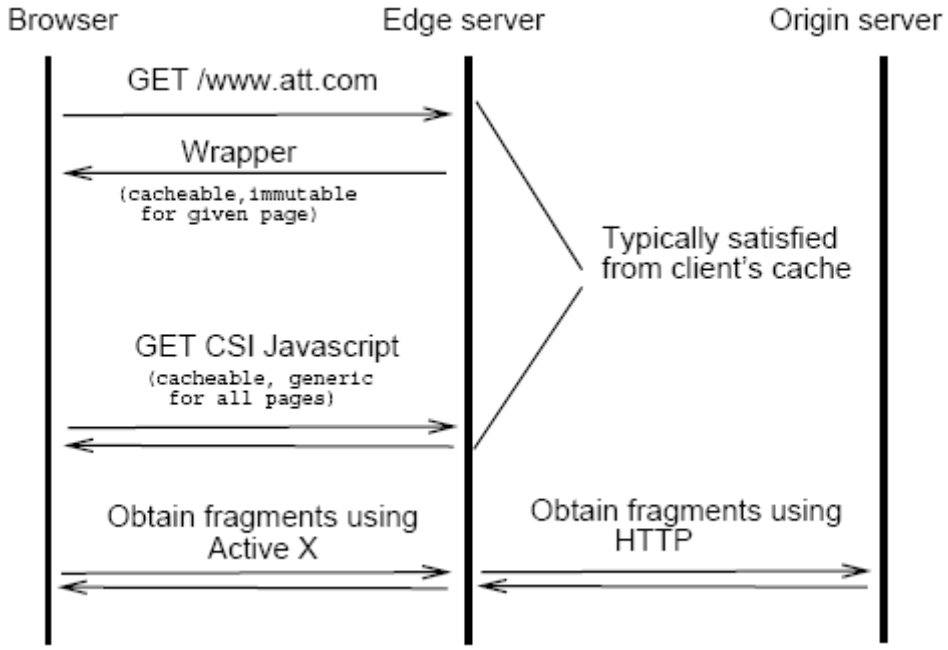
Ένα ακόμα μειονέκτημα των αρχιτεκτονικών ESI είναι ότι η συναρμολόγηση της σελίδας γίνεται στους edge-servers, κάτι το οποίο είναι ιδιαίτερα εξοικονομητικό για την καθυστέρηση που αντιλαμβάνονται οι χρήστες όταν ο στενωπός της επικοινωνίας είναι το φορτίο ή το εύρος ζώνης του server. Αντίθετα, αν ο στενωπός είναι η γραμμή σύνδεσης του τελικού πελάτη στο δίκτυο, τότε τα αποτελέσματα δεν βελτιώνουν σχεδόν καθόλου την καθυστέρηση που βλέπει ο τελικός χρήστης.

Συνήθως, η περίπτωση αυτή ισχύει όταν ο πελάτης είναι συνδεδεμένος στο διαδίκτυο μέσω τηλεφωνικής γραμμής (dial-up connection). Μάλιστα, παρ' ότι οι χρήστες αυτοί μειώνονται λόγω των νέων και γρήγορων τρόπων πρόσβασης στο διαδίκτυο, παραμένουν η μεγάλη πλειοψηφεία των χρηστών. Μελέτες στην Αμερική έδειξαν ότι το Μάρτιο του 2002 το ποσοστό των dial-up

χρηστών ήταν 79% ([RHK 2002]) και δεν αναμενόταν να πέσει κάτω από το 59% στο 2006 ([Jupiter 2001]).

Στο [Rabinovich et al. 2003] δίνεται μια λύση στο πρόβλημα αυτό, καθώς οι συγγραφείς προτείνουν έναν τρόπο για να επεκτεθεί ο μηχανισμός της δυναμικής συναρμολόγησης σελίδας στους τελικούς χρήστες του Web, ή ακριβέστερα στους browsers που αυτοί χρησιμοποιούν για να δούν το περιεχόμενο του Παγκοσμίου Ιστού.

Η αρχιτεκτονική που προτείνουν περιλαμβάνει την χρήση JavaScript για την σύνθεση της σελίδας στο πρόγραμμα πλοήγησης του τελικού χρήστη. Η υλοποίηση ακολουθεί το παρακάτω πλαίσιο λειτουργίας:



Όταν ένα πρόγραμμα πλοήγησης που υποστηρίζει CSI κάνει αίτηση για μια σελίδα, ο server επιστρέφει ένα μικρό wrapper (150 bytes συν την επικεφαλίδα της αίτησης), που φαίνεται παρακάτω:

```
<HTML>
  <BODY>
    <SCRIPT SRC="csi.js"> </SCRIPT>
    <SCRIPT> run("page_template.html"); </SCRIPT>
  </BODY>
</HTML>
```

Ο wrapper ξεκινά το πρόγραμμα σε JavaScript που κάνει την σύνθεση της σελίδας (page assembler – csi.js) και καλεί τη μέθοδο run() του assembler με είσοδο το URL του ESI template που αντιστοιχεί στην αιτούμενη σελίδα. Στη συνέχεια, το πρόγραμμα JavaScript κατεβάζει το template και όλα τα ESI τμήματα και συνθέτει την σελίδα.

Η όλη διαδικασία μοιάζει να εισάγει επιπλεόν πολυπλοκότητα με το κατέβασμα του wrapper και του προγράμματος JavaScript, αλλά αυτό δεν είναι απόλυτα ακριβές. Ο assembler που είναι γραμμένος σε JavaScript είναι ο ίδιος για όλες τις σελίδες και ουσιαστικά το πρόγραμμα πλοήγησης το κατεβάζει μόνο την πρώτη φορά και το αποθηκεύει στην cache, οπότε σε επικείμενες αιτήσεις μπορεί να κληθεί κατ' ευθείαν από το τοπικό αντίγραφο που είναι φυλαγμένο. Είναι σαν να γίνεται εγκατάσταση σε ένα plug-in στον browser, μόνο που σε αυτή την περίπτωση γίνεται με διάφανο ως προς τον χρήστη τρόπο. Ταυτόχρονα, ο wrapper έχει μικρό μέγεθος και παρ' ότι δεν είναι ο ίδιος για όλες τις σελίδες, είναι κοινός για την ίδια σελίδα και μπορεί να γίνει cache και να κληθεί τοπικά για τις επόμενες αιτήσεις στην σελίδα αυτή, ακόμα και αν η σελίδα έχει αλλάξει (αφού ο wrapper παραμένει ο ίδιος).

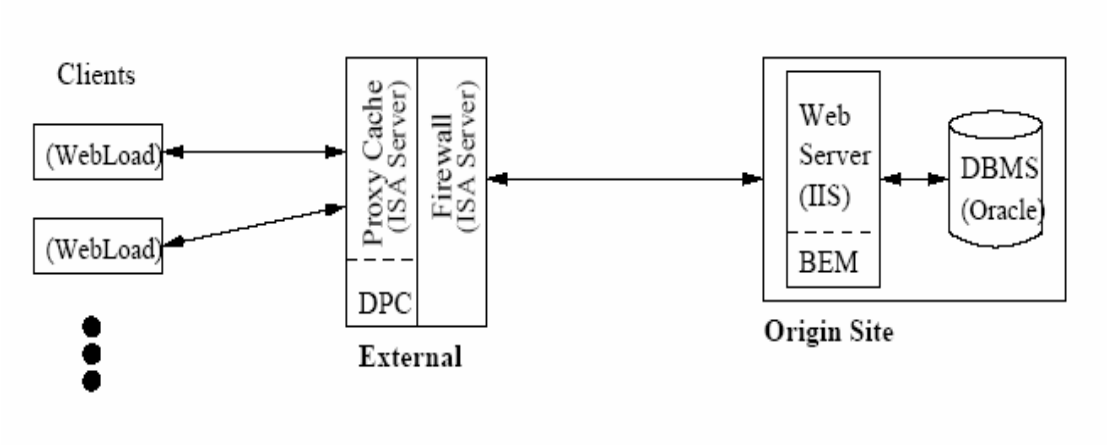
Βέβαια, το πρόγραμμα JavaScript χρησιμοποιεί ActiveX για να κατεβάσει τα ESI τμήματα, κάτι που περιορίζει την υλοποίηση σε προγράμματα πλοήγησης Internet Explorer με ενεργοποιημένο το ActiveX. Για να μην υπάρχει πρόβλημα με τους υπόλοιπους browsers, οι συγγραφείς, πρότειναν τη χρήση ενός διαφορετικού wrapper, ο οποίος στην περίπτωση που υποστηρίζεται το ActiveX λειτουργεί όπως αναφέρθηκε παραπάνω, ενώ αν όχι δίνει την οδηγία να γίνει η σύνθεση της σελίδας στον edge-server, οπότε και έχουμε μια συνηθισμένη ESI αρχιτεκτονική.

2.6 – Υποστηρίζοντας δυναμικές διατάξεις

Τελευταία, οι εφαρμογές του διαδικτύου γίνονται όλο και πιο περίπλοκες, με πλήρως δυναμικές σελίδες, στοιχεία εξατομίκευσης και πλούσια πληροφορία. Στα πλαίσια αυτά, μια νέα έννοια των δυναμικών διατάξεων (dynamic layouts) έχει κάνει την εμφάνισή της. Η διάταξη μιας σελίδας είναι ο «σκελετός» πάνω στον οποίο εμφανίζεται το περιεχόμενο, για παράδειγμα σε μια σελίδα η διάταξη της μπορεί να εισηγείται ότι στο πάνω μέρος της οθόνης θα βρίσκεται ο τίτλος της σελίδας, αριστερά οι επιλογές και στο κέντρο οι πληροφορίες της συγκεκριμένης σελίδας. Οι διατάξεις αφορούν μονάχα τον τρόπο παρουσίασης του περιεχομένου και δεν περιλαμβάνουν καμμία πληροφορία σχετική με αυτό. Οι διατάξεις είναι συνήθως στατικές, και σε αυτές προσαρτάται το δυναμικό περιεχόμενο. Στην αρχιτεκτονική των ESI, η διάταξη μιας σελίδας είναι το template αυτής της σελίδας, το οποίο προσδιορίζει σε ποια σημεία της οθόνης θα εμφανιστούν τα διάφορα ESI τμήματα που την συνθέτουν. Η εμφάνιση των δυναμικών διατάξεων έχει να κάνει με το ότι ολοένα και περισσότερες εφαρμογές δεν χρησιμοποιούν μια δεδομένη και στατική διάταξη για τις σελίδες τους, αλλά τις παράγουν δυναμικά ακόομα και τη διάταξη κάθε σελίδας. Σε μια τέτοια περίπτωση, είναι φανερό ότι η χρήση του ESI είναι ακόμα πιο δύσκολη, καθώς το template που βρίσκεται στον edge-sever δεν μπορεί να θεωρηθεί ότι είναι αξιόπιστοώστε να χρησιμοποιηθεί για την σύνθεση της σελίδας.

Μια αρχιτεκτονική που αντιμετωπίζει αυτό το πρόβλημα προτείνεται στο [Datta et al. 2002], όπου παρουσιάζεται ένα σύστημα δυναμικού caching που υποστηρίζει δυναμικές διατάξεις. Η γενική ιδέα είναι ότι το σύστημα φυλάσει όλα τα τμήματα στην proxy cache, αλλά σε κάθε αίτηση συμβουλεύεται τον server για να αναλτήσει το σωστό layout.

Η αχιτεκτονική αποτελείται από δύο υποσυστήματα, το Back-End Monitor (BEM) και την Dynamic Proxy Cache (DPC). Η DPC αποθηκεύει τα δυναμικά τμήματα (fragments) και τα συνθέτει προκειμένου να δημιουργήσει την απάντηση στις αιτήσεις των χρηστών. Το BEM είναι εκείνο που παράγει δυναμικά τη διάταξη της σελίδας για κάθε διαφορετική αίτηση. Ακολουθεί μια σχηματική απεικόνιση του συστήματος όπως αυτό υλοποιήθηκε και ελέγθηκε από την ομάδα των συγγραφέων:



Το σύστημα λειτουργεί σε δύο φάσεις. Η πρώτη φάση περιλαμβάνει την αρχικοποίηση του σχήματος caching και συγκεκριμένα την αλλαγή των scripts Που παράγουν τις δυναμικές σελίδες τη εφαρμογής ώστε να σημειωθούν σε αυτές τα τμήματα που μπορούν να γίνουν caching. Είναι η ίδια απαραίτητη προσαρμογή των εφαρμογών που προϋποθέτει και η χρήση των ESI, και είναι ένα από τα μεγαλύτερα μειονεκτήματα των τεχνικών δυναμικές συναρμολόγησης σελίδας. Η δεύτερη φάση είναι η κανονική λειτουργία του συστήματος σε πραγμτικό χρόνο. Ένα χαρακτηριστικό της λειτουργίας του συστήματος είναι ότι όλες οι αιτήσεις των τελικών χρηστών πρέπει να μεταβούν μέχρι τον κεντρικό server προκειμένου να δημιουργηθεί το layout.

Η ακριβής διαδικασία είναι η ακόλουθη: με κάθε αίτηση που φθάνει στον server, αρχίζει να εκτελείται κανονική η εφαρμογή μέχρι να συναντηθεί κάποιο μπλοκ κώδικα στο script που να είναι σημασμένο ως επιδεχόμενο caching. Σε μια τέτοια περίπτωση, εξετάζεται στο BEM, το οποίο είναι πάντοτε ενημερωμένο για τα περιεχόμενα της cache, αν το συγκεκριμένο τμήμα βρίσκεται στην DPC. Αν δεν βρίσκεται ή έχει ξεπεράσει τον χρόονο ζωής του στην cache, τότε μια νέα εγγραφή εισάγεται στη λίστα των cached σελίδων στο BEM, το περιεχόμενο παράγεται δυναμικά και μια εντολή SET (SET instruction) γράφεται στο template που τελικά στέλνεται στην DPC. Η εντολή SET κάνει την DPC να αποθηκεύσει το τμήμα αυτό. Στην αντίθετη περίπτωση, που το τμήμα είναι αποθηκευμένο στην cache, μια εντολή GET (GET instruction) καταγράφεται στο template Που στέλνεται στην DPC προκειμένου αυτή να ανακτήσει το τμήμα από τον τοπικό της χώρο αποθήκευσης.

Προφανώς, η αρχιτεκτονική αυτή επιτυγχάνει την υποστήριξη των δυναμικών διατάξεων, αλλά τα συνολικά αποτελέσματα σε σχέση με την βελτίωση της απόδοσης του συστήματος είναι αμφίβολα, καθώςόλες οι αιτήσεις πρέπει να

περάσουν από τον server και μάλιστα να εκτελέσουν και κάποιο τουλάχιστον τμήμα της λογικής της εφαρμογής. Παράλληλα, δεν καταφέρνει να αντιμετωπίσει το πρόβλημα της ανάγκης της τροποποίησης της εφαρμογής πριν από την υποστήριξη της τεχνικής της δυναμικής συναρμολόγησης.

ΚΕΦΑΛΑΙΟ 3 : ΚΩΔΙΚΟΠΟΙΗΣΗ ΔΙΑΦΟΡΩΝ (DELTA ENCODING – DE)

3.1 – Εισαγωγή

Από το προηγούμενο κεφάλαιο φάνηκε πως οι τεχνικές δυναμικής συναρμολόγησης σελίδας έχουν ένα σημαντικό μειονέκτημα: την ανάγκη τροποποίησης της εφαρμογής προκειμένου να ανασκευαστεί και να κάνει χρήση των ESI ή όποιας άλλης τεχνολογίας δυναμικής συναρμολόγησης. Σε αντίθεση με τα συστήματα caching που αποθηκεύουν τμήματα των HTML σελίδων, υπάρχουν άλλα τα οποία αποθηκεύουν ολόκληρες τις σελίδες, και τα οποία στη γενική περίπτωση είναι λιγότερο απαιτητικά ως προς την τροποποίηση της υπάρχουσας τεχνολογικής υποδομής και των εφαρμογών που ήδη έχουν δημιουργηθεί.

Χαρακτηριστικό παράδειγμα προς αυτή την κατεύθυνση είναι η εργασία των [Candan et al. 2001], όπου γίνεται μια αξιόλογη προσπάθεια ώστε το σύστημα caching που προτείνουν να είναι εντελώς ανεξάρτητο από τις τεχνολογίες που χρησιμοποιούνται, να μην απαιτεί καμμία αλλαγή στις τρέχουσες εφαρμογές και να λειτουργεί σε μια διάταξη που θα επιβαρύνει κατά το ελάχιστο δυνατό τον server. Βσική ιδιαιτερότητα αυτού του συστήματος caching είναι η χρησιμοποίηση μιας συμβατικής cache στατικού περιχομένου για την αποθήκευση δυναμιών ιστοσελίδων. Για το λόγο αυτό, η HTML των απαντήσεων αποθηκεύεται ολόκληρη στην cache, και γίνεται η ακύρωσή της (invalidation) με μηνύματα που στέλνει ο server που παρακολουθεί τις αλλαγές του περιεχομένου. Το μόνο μη καθιερωμένο χαρακτηριστικό που πρέπει να έχει η cache είναι να υποστηρίζει με κάποιο τρόπο την λήψη αυτών των μηνυμάτων ακύρωσης από το server, κάτι που όμως δεν είναι τόσο εξεζητημένο και υποστηρίζεται ήδη από πολλές εμπορικές εκδόσεις συστημάτων caching. Στην εργασία αυτή θα επιστρέψουμε σε επόμενο κεφάλαιο που θα αναλυθεί οτρόπος με τον οποίο γίνεται η μετάδοση των μηνυμάτων στην cache από τον server (push caching model).

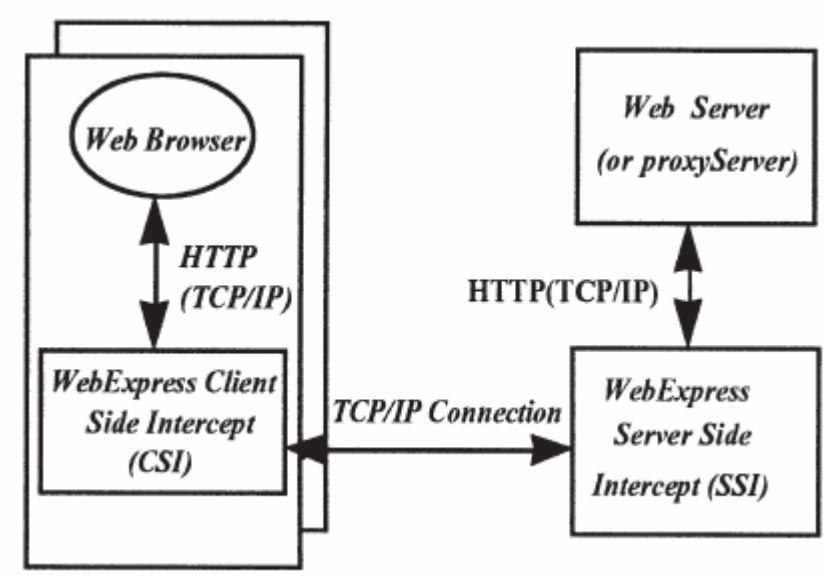
Βέβαια, η αποθήκευση ολόκληρων των σελίδων στις caches δεν εμφανίζει τα εξαιρετικά χαρακτηριστικά που έχει η δυναμική συναρμολόγηση σελίδας, η οποία επιτυγχάνει υψηλότερη επαναχρησιμοποίηση των αποθηκευμένων τμημάτων και την ελάττωση του καταναλισκόμενου εύρους ζώνης στο δίκτυο. Σίγουρα, τα χαρακτηριστικά αυτά κάνουν ένα σύστημα δυναμικής συναρμολόγησης σελίδας πολύ αποδοτικότερο από ένα σύστημα αποθήκευσης ολόκληρων σελίδων, παρ' όλο που μπορεί να είναι πιο πολύπλοκο στην ενσωμάτωσή του σε μια εφαρμογή.

Την επίτευξη των επιθυμητού χαρακτηριστικού της ελάττωσης του καταναλισκόμενου εύρους ζώνης στα συστήματα δυναμικού caching ολόκληρων HTML σελίδων μπορεί να προφέρει η τεχνική του Delta Encoding (DE). Το Delta Encoding βασίζεται σε μια αρκετά παλιά ιδέα που είναι διάσπαρτη σε πολλές επιστήμες, συμπεριλαμβανομένης και της πληροφορικής. Η ιδέα αυτή είναι ότι όταν μελετάται ένα αντικείμενο που αλλάζει, μπορεί να ανακατασκευαστεί μια νεότερη έκδοσή του, αρκεί να φυλάσσεται η αρχική έκδοση και η διαφορά του από αυτή. Στα πλαίσια του caching, το όφελος που μπορεί να δώσει αυτή η τεχνική είναι ότι δεδομένου ότι στον proxy υπάρχει μια παλιότερη έκδοση της σελίδας, μπορεί να του

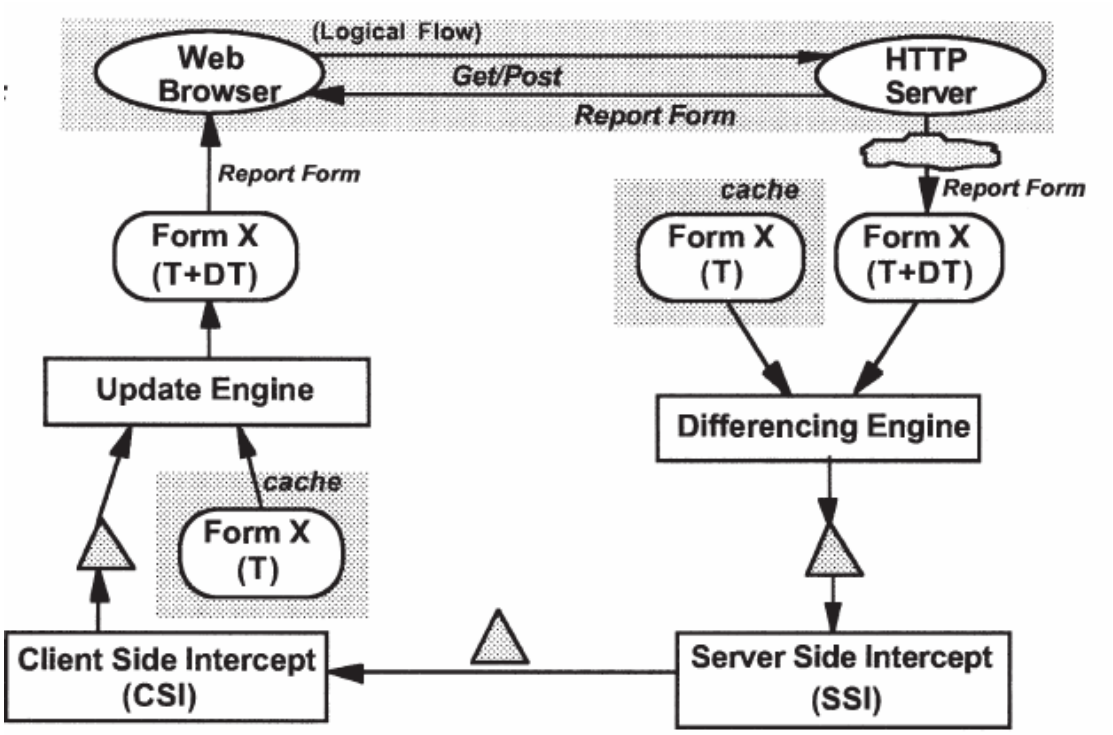
σταλλεί μόνο η διαφορά, αντί ολόκληρης της νέας σελίδας. Τα αποτελέσματα μιας τέτοιας μεθόδου μπορεί να είναι ιδιαίτερα επιτυχή, εξ' αιτίας του ότι συνήθως οι δυναμικές σελίδες αλλάζουν σε μικρό βαθμό σε σχέση με τις κοντινές εκδόσεις τους. Στη συνέχεια του κεφαλαίου θα εξεταστεί σε μεγαλύτερο βάθος η τεχνική του Delta Encoding.

3.2 – Η τεχνική του Delta Encoding

Το Delta Encoding εισάχθηκε για πρώτη φορά στο χώρο του web caching από τους [Banga et al. 1997], και ανεξάρτητα αλλά σε πιο εξειδικευμένη θεώρηση θεώρηση νωρίτερα στο [Housel and Lindquist 1996]. Η εργασία των Housel – Lindquist θεωρείται η πρώτη που χρησιμοποίησε την ιδέα του delta encoding στο web caching, αλλά επειδή είχε σκοπό την ανάπτυξη ενός συστήματος που θα χρησιμοποιόταν αποκλειστικά σε ασύρματες συσκευές, οι χρήστες των οποίων θέλουν να στέλνουν κάποιες δοσοληψίες μέσω του δικτύου, δεν βασίζεται πάνω στο πρωτόκολλο HTTP και δεν μπορεί να χρησιμοποιηθεί γενικά στο WWW. Παρ' όλα αυτά, η ιδέα της κωδικοποίησης διαφορών είναι ξεκάθαρη στη λειτουργία του συστήματος, όπως φαίνεται και από τα σχήματα της αρχιτεκτονικής και λειτουργίας του συστήματος:



Η επικοινωνία μεταξύ του servre και της ασύρματης συσκευής γίνεται σε ένα εξειδικευμένο πρωτόκολλο πάνω από το TCP, αλλά η επικοινωνία από το server στον agent του και από τον agent του client στον browser γίνεται σε HTTP, με αποτέλεσμα να υπάρχει μια λογική επικοινωνία μεταξυ client και server σε πρωτόκολλο HTTP. Ο υπολογισμός της διαφοράς γίνεται στον server και αποστέλλεται στον client μόνο η διαφορά, ώστε να εξοικονομείται εύρος ζώνης στην αργή σύνδεση της ασύρματης συσκευής.



Στο [Banga et al. 1997], γίνεται μια πιο γενική πρόταση του delta encoding.Η τεχνική που προτείνουν οι συγγραφείς είναι ότι όταν κάποιος server είναι ιδιαίτερα βεβαρυμένος, ο τελικός χρήστης θα βλέπει βελτίωση στο χρόνο αναμονής για την απάντηση αν, ενώ περιμένει να απαντήσει στην αίτηση ο server, κατεβάζει από κάποιον proxy μια παλιότερη έκδοση της σελίδας. Μόλις ο server απαντήσει, ο proxy που έστειλε την προηγούμενη σελίδα στον τελικό χρήστη, υπολογίζει την διαφορά και στέλνει μόνο αυτή στον πελάτη. Όπως γίνεται εύκολα αντιληπτό, η αρχιτεκτονική αυτή είναι μια πρώτη προσπάθεια για την χρήση Delta Encoding και δεν αξιοποιεί τα ωφέλη του DE σε όλο το μήκος της επικοινωνίας, ούτε το γεγονός ότι ο τελικός χρήστης μπορεί ήδη να έχει αποθηκευμένο κάποια παλιότερη έκδοση της σελίδας.

Η πιο εμπεριστατομένη δουλειά για την ενσωμάτωση ενός μηχανισμού διαφορών στο HTTP πρωτόκολλο έγινε με τις εργασίες του Mogul, [Mogul et al. 1997] και [Mogul et al. 2001].

Η πρώτη από τις δύο εργασίες αφορούσε την ποσοτικοποίηση, με βάση πειραματικά δεδομένα, των ωφελειών που θα είχε ένας μηχανισμός διαφορών στο HTTP. Για το σκοπό αυτό, οι συγγραφείς διενέργησαν μια αρκετά μεγάλη και σημαντική μελέτη, κατά την οποία συγκεντρώθηκαν και μελετήθηκαν αρχεία καταγραφής κίνησης του δικτύου από δύο διαφορετικές τοποθεσίες στην αμερική. Τα ίχνη από τη μία τοποθεσία καταγράφηκαν σε επίπεδο αιτήσεων και απαντήσεων που πέρασαν από τον proxy, ενώ της δεύτερης σε επίπεδο πακέτων του πρωτοκόλλου TCP.

Τα αποτελέσματα αυτής της μελέτης έδειξαν ότι το 38% των πόρων επιδεχόταν την εφαρμογή delta encoding. Μάλιστα, αν ο χρήστης είχε ήδη ένα αντίγραφο στην τοπική του cache, τότε το 83% περίπου των bytes της απάντησης στην αίτηση για αυτό τον πόρο δεν θα είχε μεταφερθεί πάνω από

το δίκτυο. Αντίστοιχη θα ήταν και η βελτίωση στο χρόνο καθυστέρησης που θα αντιλαμβανόταν μέχρι να λάβει την απάντηση. Ακόμα, στην μελέτη αυτή, συγκρίνονται τρεις αλγόριθμοι υπολογισμού διαφορών (“diff –e”, “diff –e | gzip”, “vcdiff”) το τον vcdiff να αποδεικνύεται ο καλύτερος.

Μετά από αυτή τη μελέτη, οι Mogul et al. πρότειναν μια επέκταση του HTTP πρωτοκόλλου που υποστηρίζει το Delta Encoding [Mogul et al. 2001]. Οι επεκτάσεις που πρότειναν κλήθηκαν να χειριστούν τα ακόλουθα ζητήματα που προκύπτουν στη διαδικασία αίτησης ενός πόρου από ΄ναν πελάτη που υποστηρίζει delta encoding:

1) Η έκφραση του πελάτη ότι υποστηρίζει και ενδιαφέρεται να δεχθεί μια απάντηση κωδικοποιημένη με Delta Encoding. Αυτό επιτυγχάνεται με την επικεφαλίδα A-IM (Accept-Instance-Manipulation), ο οποίος καθορίζει το σύνολο των υποστηριζόμενων αλγορίθμων διαφορών. Για παράδειγμα, η αίτηση:

```
GET /chap15.html HTTP/1.1
Host: www.vcdiff.com
If-None-Match: “38432-s8-13”
A-IM: vcdiff, diffe, gzip
```

σημαίνει ότι ο πελάτης προτιμά την κωδικοποίηση vcdiff από την diffe. Η τιμή την επικεφαλίδας If-None-Match δείχνει με βάση ποιο στιγμιότυπο του πόρου θα πρέπει να υπολογιστεί η διαφορά. Αν ο πόρος δεν έχει αλλάξει, ο server μπορεί όπως συνήθως να απαντήσει με «304 – Not Modified», ενώ αν έχει αλλάξει υπολογίζεται η διαφορά της τρέχουσας έκδοσης του πόρου με εκείνη που είχε etag 38432-s8-13 και αποστέλλεται αντί ολόκληρης της απάντησης.

2) Η έκφραση του server για το αν τελικά επέλεξε να στείλει την διαφορά αντί της σελίδας, και αν ναι με ποιο αλγόριθμο αυτή υπολογίστηκε. Για να δωθεί αυτή η δυνατότητα στον server επεκτείνεται το HTTP με ένα νέο κωδικό απάντησης (response code) και μια νέα επικεφαλίδα:

```
HTTP/1.1 226 IM Used
Date: Sun, 2 Jul 2000 23:35:35 GMT
Etag: “192-qpt-899”
IM: vcdiff
...
```

Ο κωδικός απάντησης «226 IM Used» δηλώνει ότι τελικά ο server υπολόγισε την διαφορά, ενώ η επικεφαλίδα δείχνει τον αλγόριθμο που εφάρμοσε για τον σκοπό αυτό.

3) Υποστήριξη ανταλλαγής δεδομένων με ranges. Το HTTP, ως γνωστό υποστηρίζει την αίτηση για ένα μέρος ενός πόρου. Η επέκταση που προτείνται υποστηρίζει την αίτηση για ένα μέρος της διαφοράς:

Η αίτηση
```
GET /chap15.html HTTP/1.1
```

```
Host: www.vcdiff.com
If-None-Match: "38432-s8-13"
A-IM: vcdiff, range
Range: bytes=0-200
```

Και η απάντηση

```
HTTP/1.1 226 IM Used
Date: Sun, 2 Jul 2000 23:35:35 GMT
Etag: "192-qpt-899"
IM: vcdiff, range
…
```

4) Ο πελάτης μπορεί να έχει πάνω από μια εκδόση ενός πόρου, οπότε σε μια τέτοια περίπτωση, μπορεί να ανακατασκευάσει τον πόρο λμβάνοντας την διαφορά από οποιοδήποτε από τα τρεία αυτά αρχεία. Για να δηλώσει ποια αρχεία μπορεί να χρησιμοποιήσει σαν βάση ο πελάτης, μπορεί να τα συμπεριλάβει στην If-None-Match επικεφαλίδα:

```
GET /chap15.html HTTP/1.1
Host: www.vcdiff.com
If-None-Match: "38432-s8-13", "97-ru486-v"
A-IM: vcdiff
```

Αν ο server λάβει μια τέτοια αίτηση, πρέπει να επισημάνει ποια έκδοση χρησιμοποίησε σαν βάση. Για αυτό τον σκοπό χρησιμοποιεί την επικεφαλίδα Delta-Base:

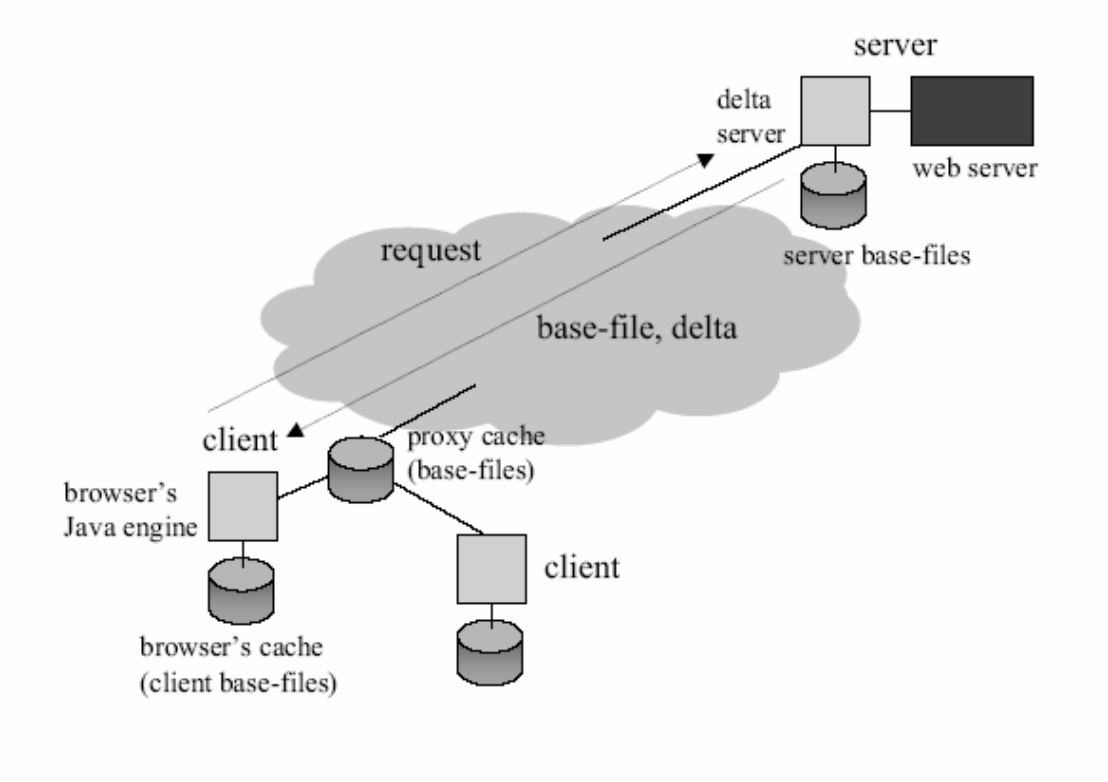
```
HTTP/1.1 226 IM Used
Date: Sun, 2 Jul 2000 23:35:35 GMT
Etag: "282-ela-899"
IM: vcdiff
Delta-Base: "97-ru486-v"
```

3.3 – Delta Encoding βασισμένο σε κλάσεις σελίδων

Το κύριο πρόβλημα του Delta Encoding (DE) είναι ότι έχει μεγάλες απαιτήσεις σε αποθηκευτικό χώρο στον server προκειμένου να φυλλάει αντίγραφα των εκδόσεων που στέλνει στα proxy caches, ώστε να μπορεί στην επόμενη αίτηση να υπολογίσει την διαφορά με βάση την ζητούμενη έκδοση. Πολύ περισσότερο για τις σύγχρονες δυναμικές σελίδες που είναι σε μεγάλο βαθμό εξατομικευμένες στο χρήστη που τις ζητάει, οπότε και θα πρέπει για το ίδιο έγγραφο να φυλάσσονται τόσα διαφορετικά αρχεία βάσεις, όσα και οι διαφορετικοί χρήστες

Η λύση για το πρόβλημα αυτό είναι να αξιοποιηθεί, εκτός από την χρονική συσχέτιση (temporal correlation) μεταξύ διαδοχικών εκδόσεων του ίδιου πόρου, και η χωρική συσχέτιση (spatial correlation) μεταξύ διαφορετικών εγγράφων του ίδιου server, που όμως είναι πιθανό να παρουσιάζουν μεγάλες ομοιότητες. Μια τεχνική που αξιοποιεί και τις δύο συσχετίσεις παρουσιάζεται

από το [Psounis 2002], και σε εμπορική έκδοση το σύστημα δυναμικού caching από την [FineGround]. Η τεχνική που προτείνεται είναι το λεγόμενο Delta Encoding βασισμένο σε κλάσεις (Class Based Delta Encoding - CBDE). Στο CBDE τα δυναμικά έηηραφα ομαδοποιούνται σε κλάσεις και μόνο ένα αρχείο βάση φυλάσσεται σε κάθε μια κλάση. Προκειμένου να επιτευχεί αυτό, γίνεται χρήση δύο αλγορίθμων, ενός για να αυτοματοποιεί τον διαχωρισμό των εγγράφων σε κλάσεις, και ενός για την αποδοτική επιλογή του κατάλληλου αρχειου βάσης κάθε κλάσης, σε πραγματικό χρόνο.



Ο διαχωρισμός των εγγράφων στις κλάσεις:
Ο διαχωρισμός των εγγράφων στις κλάσεις γίνεται με βάση το μέγεθος της διαφοράς του εξεταζόμενου εγγράφου από το αρχείο βάσης κάθε κλάσης. Αν η διαφορά είναι αρκετά μικρή, τότε το εγγραφο κατηγοριοποιείται στην κλάση αυτή. Η όλη λειτουργία του συστήματος βασίζεται σε ένα υποσύστημα που λέγεται delta-server και υλοποιεί το class based delta encoding. Είναι σημαντικό να περνούν όλες οι αιτήσεις από τον delta-server πριν κατευθυνθούν στους web servers. Αρχικά, ο delta-server δεν έχει καμμία κλάση σελίδων. Μόλις φθάσει μια αίτηση στον delta-server, είτε κατατάσσεται σε μια από τις υπάρχουσες κλάσεις, είτε δημιουργείται μια νέα. Η κατάταξη γίνεται μετά την σύγκριση των διαφορών του εγγράφου με τα αρχεία βάσης των κλάσεων. Καθώς μεγαλώνει ο αριθμός των διαφορετικών κλάσεων αυξάνει η πολυπλοκότητα της εξαντλητικής αναζήτησης, οπότε αυτή καθίσταται απαγορευτική. Προκειμένου να γίνεται πιο γρήγορα η αναζήτηση, χρησιμοποιείται σαν ευριστικό κριτήριο η ομοιότητα μεταξύ των URLs των αιτήσεων, καθώς η εμπειρία λέει πως δύο παρόμοια URLs έχουν αυξημένη πιθανότητα να έχουν και παρόμοιο περιεχόμενο.

Αναλυτικότερα, ένα URL χωρίζεται σε τρία τμήματα: το τμήμα του server (server-part), το τμήμα της αναζήτησης (hint-part), και το υπόλοιπο τμήμα του URL. Το τμήμα του server είναι από την αρχή του URL μέχρι την πρώτη κάθετο, ως συνήθως. Το τμήμα που χρησιμοποιείται στην αναζήτηση εξαρτάται από τον ιεραρχικό τρόπο που φυλάσσει τα αρχεία ο web server Και καθορίζεται κάθε φορά από τον διαχειριστή του site. Στον παρακάτω πίνακα φαίνονται διάφορες εναλλακτικές περιπτώσεις για την διάσπαση του URL:

URL	hint-part	rest
www.foo.com/laptops?id=100	laptops	id=100
www.foo.com/?dept=laptops&id=100	dept=laptops	id=100
www.foo.com/laptops/100	laptops	100

Με τη γνώση των παραπάνω μπορεί να σκιαγραφηθεί η λειτουργία της διαδικασίας ομαδοποίησης των εγγράφων σε κλάσεις. Επιτυχή σύγκριση υπάρχει όταν η διαφορά μεταξύ του εν λόγω εγγράφου με το αρχείο βάσης είναι μικρότερη σε μέγεθος από ένα όριο (threshold) που είναι παράμετρος του συστήματος. Επειδή είναι σπάνιο δύο έγγραφα που προέρχονται από διαφορετικούς web servers να εμφανίζουν μεγάλη ομοιότητα, μια νέα κλάση δημιουργείται με την άφιξη μιας αίτησης της οποίας το server-part του URL είναι διάφορο από όλα τα server parts όλων των αρχελιων βάσης όλων κλάσεων. Στην περίπτωση που υπάρχουν κλάσεις που έχουν το ίδιο server part με το έγγραφο, εφαρμόζονται κάποιοι ευριστικοί κανόνες για την απόφαση της κατηγοριοποίησης. Οι ευριστικοί κανόνεςπου ακολουθούνται είναι οι εξής:

- Αν κάποιες κλάσεις έχουν έγγραφα των οποίων τα URLs έχουν hint-parts που είναι τα ίδια με το Hint-part της αίτησης, τότε ο μηχανισμός εξετάζει τις διαφορές του εγγράφου μόνο με τις κλάσεις αυτές.
- Ο αλγόριθμος ποτέ δεν διενεργεί πάνω από Ν συγκρίσεις με υπάρχουσες κλάσεις, για να αποφευχθεί το μεγάλο υπολογιστικό φορτίο στον delta-server. Αν μετά τις Ν συγκρίσεις δεν έχει βρεθεί κάποια κατάλληλη κλάση, τότε το έγγραφο δημιουργεί μια νέα κλάση.
- Πρώτα επιχειρείται η σύγκριση με τις πιο δημοφιλείς κλάσεις, που έχουν περισσότερα έγγραφα, καθώς υπάρχει μεγαλύτερη πιθανότητα να ανήκει εκεί το νέο έγγραφο. Συγκεκριμένα, τις πρώτες α*Ν φορές γίνονται συγκρίσεις με τις πιο δημοφιλείς κλάσεις και τις υπόλοιπες (α-1)*Ν γίνονται συγκρίσεις με τυχαίες επιλογές μεταξύ των υπόλοιπων δυνατών κλάσεων.
- Τέλος, επειδή στην κατηγοριοποίηση αυτή των εγγράφων δεν είναι απαραίτητη η εύρεση της διαφοράς με ακρίβεια, αλλά μόνο η προσεγγιστική εκτίμηση του μεγέθους της διαφοράς, χρησιμοποιείται μια ειδικής ελαφρύτερη έκδοση αλγορίθμου υπολογισμού διαφορών, ώστε να αποφευχθεί κατά το δυνατόν η έντονη υπολογιστική δραστηριότητα.

Η επιλογή του κατάλληλου αρχείου βάσης:
Αφού έχουν δημιουργηθεί οι κλάσεις των σελίδων, πρέπει να επιλεχθεί ή να κατασκευαστεί το αρχείο το οποίο θα χρησιμεύει σαν τη βάση της κλάσης για τον υπολογισμό όλων των απαραίτητων διαφορών. Ο απλούστερος τρόοπος

θα ήταν να επιλέγεται σαν αρχείο βάσης η σελίδα που δημιούργησε την κλάση, όμως αυτός ο τρόπος δεν θα ήταν καθόλου βελτιστοποιημένος. Η βέλτιστη επιλογή θα ήταν να κατασκευάζεται ένα τεχνητό αρχείο που θα συνδύαζε τα περισσότερα κοινά τμήματα μεταξύ των σελίδων της κλάσης. Δυστυχώς, αυτή η επιλογή είναι ιδιαίτερα απαιτητική σε υπολογιστικούς πόρους και δεν μπορεί να εφαρμοστεί με καλά αποτελέσματα. Συνεπώς, πρέπει να γίνει κάποιος συμβιβασμός μεταξύ της απλούστερης και της δυσκολότερης επιλογής. Ο συνβιβασμός που προτείνεται στην εργασία αυτή είναι να γίνεται επιλογή, για αρχείο βάσης, του καλύτερου εγγράφου μεταξύ των εγγράφων της κλάσης.

Η μέτρηση του κατά πόσο είναι «καλή» μια σελίδα για αρχείο βάσης γίνεται με βάση το αν ελαχιστοποιεί την συνολική διαφορά μεταξύ των εγγράφων και του αρχείου βάσης. Βέβαια, σε ένα πραγματικό σύστημα caching δεν είναι γνωστές εκ των προτέρων όλες οι αιτήσεις που θα φθάσουν στον delta-server, οπότε και θα μπορούσε να υπολογιστεί με ακρίβεια η καλύτερη επιλογή με βάση το παραπάνω κριτήριο, αλλά μόνο οι αιτήσεις που έχουν ήδη φθάσει στον delta-server. Επίσης, ούτε η μέθοδος της σύγκρισης με όλες τις προηγούμενες αιτήσεις θα ήταν εφικτός λόγω αυξημένου υπολογιστικού κόστους στην περίπτωση που υπάρχουν πολλές σελίδες. Αντίθετα, στο σύστημα αυτό χρησιμοποιείται ο παρακάτω online αλγόριθμος:

1. Κάθε αίτηση δειγματοληπτείται με πιθανότητα p ώστε να θεωρείται υποψήφιο αρχείο βάσης και αν επιλεχθεί για υποψήφιο αρχείο βάσης αποθηκεύεται στη μνήμη.
2. Χρησιμοποιείται σαν αρχείο βάσης το καλύτερο από εκείνα που είναι αποθηκευμένα στη μνήμη, δηλαδή εκείνο που ελαχιστοποιεί τη συνολική διαφορά από τα υπόλοιπα αποθηκευμένα έγγραφα.
3. Γίνεται αποθήκευση μέχρι K εγγράφων. Αν δειγματοληφθεί κάποιο έγγραφο και βρίσκονται ήδη K στη μνήμη, τότε η πολιτική αντικατάστασης που ακολουθείται είναι η διαγραφή από το σύνολο των υποψηφίων αρχείων βάσης εκείνου του εγγράφου που παρουσιάζει τη μέγιστη συνολική διαφορά.

Προφανώς, η αλλαγή τηυς βάσης μιας κλάσης δεν είναι μια απλή διαδικασία, ακόμα και αν έχει επιλεχθεί η νέα βάση, γιατί πρέπει να ενημερωθούν τα αρχεία βάσεις που βρίσκονται σε όλους τους πελάτες και τις caches. Η διαδικασία της αλλαγής βάσης ονομάζεται rebase στο σύστημα αυτό και προκειμένου να μην γίνεται υπερβολικά πολλές φορές, επιτρέπεται να λάβει χώρα μόνο αν και έχει βρεθεί κάποιο καλύτερο έγγραφο βάση και έχει περάσει ένα χρονικό όριο timeout ()παράμετρος του συστήματος) από την προηγούμενη αλλαγή βάσης.

Οι διαδικασίες αλλαγής βάσης που γίνονται στον αλγόριθμο είναι δύο κατηγοριών, οι group-rebases και οι basic-rebases. Οι group-rebases είναι αυτές που περιγράφηκαν παραπάνω και συμβαίνουν όταν ανακαλυφθεί ένα καλύτερο αρχείο βάσης. Οι basic-rebases προκαλούνται όταν η παραγόμενη διαφορά είναι αρκετά μεγάλη, οπότε και αλλάζει η βάση. Όταν συμβαίνει μια basic-rebase και τα K υποψήφια έγγραφα που είναι αποθηκευμένα στην μνήμη διαγράφονται και η διαδικασία ξεκινά από την αρχή.

Πέρα από τους αλγορίθμους αυτούς που αποτελούν την καρδιά αυτού του συστήματος δυναμικού caching, ο συγγραφέας ασχολείται και με ένα άλλο θέμα που προκείπτει στο class based delta encoding. Προκειται για ζητήματα διατήρησης της ιδιοτικότητας των δεδομένων. Τα προβλήματα αυτά εμφανίζονται διότι αν έχει επιλεχθεί για αρχείο βάσης μια σελίδα που έχει εξατομικευμένα στοιχεία από κάποιο χρήστη, αυτά θα διαδοθούν σε όλους τους πελάτες που κρατούν το αρχείο βάσης για τον υπολογισμό των διαφορών και την ανακατασκευή των απαντήσεων. Ο τρόπος με τον οποίο αντιμετωπίζεται το πρόβλημα αυτό είναι να αφαιρούνται εκείνα τα τμήματα του base file που είναι μοναδικά σε αυτό το έγγραφο και δεν υπάρχουν σε κανένα από τα υπόλοιπα έγγραφα της κλάσης.

3.4– Σύγκριση των τεχνικών του DE και του ESI

Με την επικράτηση των δύο αυτών τεχνικών σαν τις δύο κυριότερες εναλλακτικές λύσεις για την ανάπτυξη συστημάτων δυναμικού caching, άρχισαν να γίνονται και οι πρώτες συγκρίσεις μεταξύ τους. Αν και οι δύο τεχνικές δεν έχουν απόλυτες αντιστοιχίες ώστε να συγκριθούν με συγκεκριμένα κριτήρια για την αξιολόγησή τους, το γεγονός ότι έχουν τον ίδιο στόχο, που είναι η ελάττωση του καταναλισκόμενου εύρους ζώνης, επιτρέπει τη σύγκριση των αποτελεσμάτων των δύο τεχνικών.

Μια τέτοια σύγκριση επιχειρείται στο [Naaman et al. 2003], όπου εφαρμόζονται οι δύο τεχνικές σε δύο είδη δικτυακών πόρων. Οι πόροι που χρησιμοποιούνται είναι ένα υποθετικό ηλεκτρονικό βιβλιοπωλείο με μεγάλα fragments που έχουν και υψηλό TTL και λίγα εξατομικευμένα στοιχεία, και μια προσωπική σελίδα “My Page”, σε μεγάλο βαθμό εξατομικευμένη και με μεγάλη ποικιλομορφία όσον αφοράτο μέγεθος και το TTL των τμημάτων.

Το χαρακτηριστικό της εφαρμογής “My Page” είναι ότι όλες οι σελίδες της μοιαζουν σε μεγάλο βαθμό και συνεπώς στην πλειοψηφία των αιτήσεων ο πελάτης θα παίρνει το ίδιο βασικό αρχείο, οπότε επιτυγχάνεται μείωση της συνολικής πληροφορίας που μεταφέρεται και μεταξύ cache και πελάτη. Τα κέρδη από τις δύο μεθόδους στην εφαρμογή “My Page” φαίνονται παρακάτω:

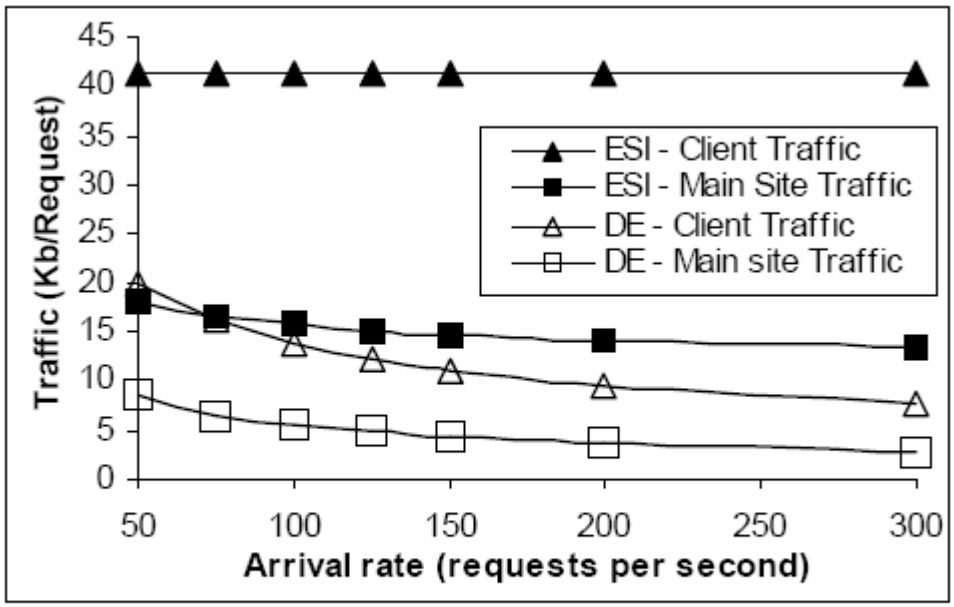
My Page comparison

	Savings client link	Savings server link	Edge cache usage
ESI	0%	62%	1.5Mb
DE	66%	87%	3.2Mb

Προφανώς, η τεχνική του ESI δεν εμφανίζει κάποιο κέρδω στη σύνδεση μεταξύ cache και client, αφού η σελίδα συντίθεται στην cache (edge-server) και στέλνεται ολόκληρη στον πελάτη. Επίσης, είναι φανερές και οι μεγαλύτερες απαιτήσεις του class based delta encoding σε αποθηκευτικό χώρο, ενώ τα κέρδη στη σύνδεση με τον server είναι σαφώς μεγαλύτερα με το

DE γιατί γίνεται πλήρης εκμετάλλευση της χρονικής συσχέτισης μεταξύ των σελίδων και αποστέλλεται μονάχα η διαφορά.

Στο επόμενο διάγραμμα, παρουσιάζεται το ποσό της πληροφορίας που αντιστοιχεί σε κάθε αίτηση, καθώς οι αιτήσεις αυξάνουν σε ρυθμό άφιξης. Και στα δύο συστήματα εμφανίζεται μείωση της κίνησης με την αύξηση των αιτήσεων, λόγω του καλύτερου caching το οποίο επιτυγχάνεται. Βέβαια, υπάρχει η διαφορά ότι στο DE επιτυγχάνεται και μείωση στη σύνδεση του πελάτη (Client Traffic).



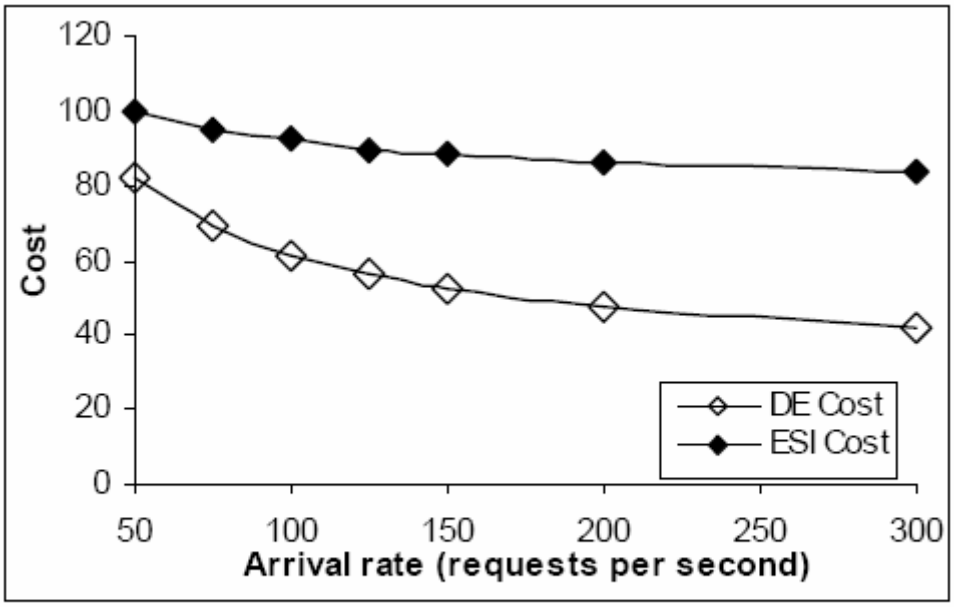
My Page resource, traffic for ESI and class-based DE vs. request arrival rate

Αξίζει να σημειωθεί ότι στο DE οι πελάτες δέχονται δεδομένα και από τις caches (base files) και από το server (deltas). Η καμπύλη του παραπάνω σχήματος δείχνει τον συνολικό όγκο των δεδομένων. Αντίθετα, στην ESI οι πελάτες δέχονται δεδομένα μόνο από τον edge-server. Η λεπτομέρεια είναι ότι η μεταφορά από τον server στον client είναι σίγουρα πιο χρονοβόρα από την μεταφορά μεταξύ cache και πελάτη, κάτι που δεν αντικατοπτρίζεται στο διάγραμμα. Για το λόγο αυτό, μπορεί να χρησιμοποιηθεί μια φόρμουλα κόστους που υπολογίζει με βάρη το συνολικό κόστος που εισάγει στην δικτυακή κίνηση η κάθε τεχνική. Η συνάρτηση, λοιπόν, κόστους είναι η ακόλουθη:

Cost = C1*client-server traffic + C2*cache-server traffic + client-cache traffic

όπου τα C1, C2 είναι οι παράμετροι που καθορίζουν ποια σύνδεση θεωρείται πιο αργή. Για παράδειγμα, για τιμές C1=25 και C2=5, δηλαδή σύνδεση χρήστη-εξυπηρέτη πέντε φορές πιο αργή από αυτήν της cache με τον εξυπηρέτη, 1 byte στη γραμμή client-cache αντιστοιχεί σε 5 bytes στη γραμμή cache-server (προφανώς θεωρούμε ότι η cache βρίσκεται πιο κοντά στον

πελάτη από ότι στον server) και 25 στη γραμμή client-server. Για τα προαναφερθέντα C1 και C2 προκύπτει το ακόλουθο διάγραμμα:

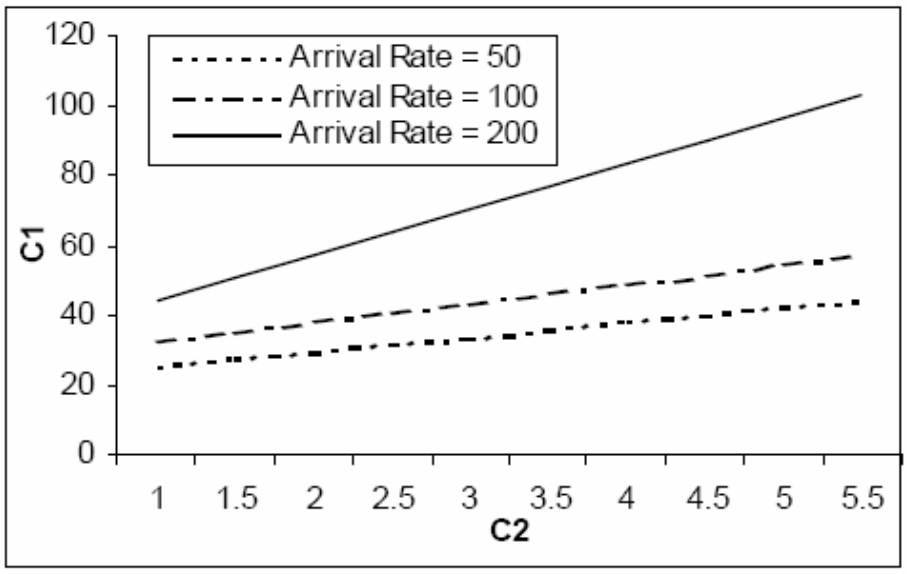


My Page resource, cost of ESI and class-based DE vs. request arrival rate

Είναι ευδιάκριτο ότι ακόμα και με σταθμισμένο κόστος, το DE δίνει καλύτερα αποτελέσματα από ότι τα ESI στην εφαρμογή “My Page”, και μάλιστα είναι ακόμα καλύτερο καθώς αυξάνει ο ρυθμός άφιξης των αιτήσεων.

Στη συνέχεια παρατίθεται ένα σχεδιάγραμμα που δείχνει για διάφορους ρυθμούς αφίξεων ποιες πρέπει να είναι οι τιμές των C1, C2 για ναείναι καλύτερη κάθε μια από τις δύο μεθόδους. Για δεδομένο ρυθμό άφιξης αιτήσεων και σταθερές τιμές των C1 και C2, οι οποίες εξαρτόνται από το φυσικό επίπεδο των συνδέσεων του δικτύου και μπορούν να υπολογιστούν σχετικά εύκολα, με το παρακάτω διάγραμμα μπορεί να αποφασιστεί ποια τεχνική είναι καλύτερη.

Αν το ζεύγος (C1,C2) πέφτει πάνω από την καμπύλη, τότε το κόστος του ESI είναι μικρότερο του DE και συμφέρει να χρησιμοποιηθεί η τεχνική της δυναμικής συναρμολόγησης. Αντίθετα, αν πέφτει κάτω από την καμπύλη, τότε πιο συμφέρουσα επιλογή είναι το class based delta encoding.



My Page, values for C1, C2 where DE cost is equal to ESI cost

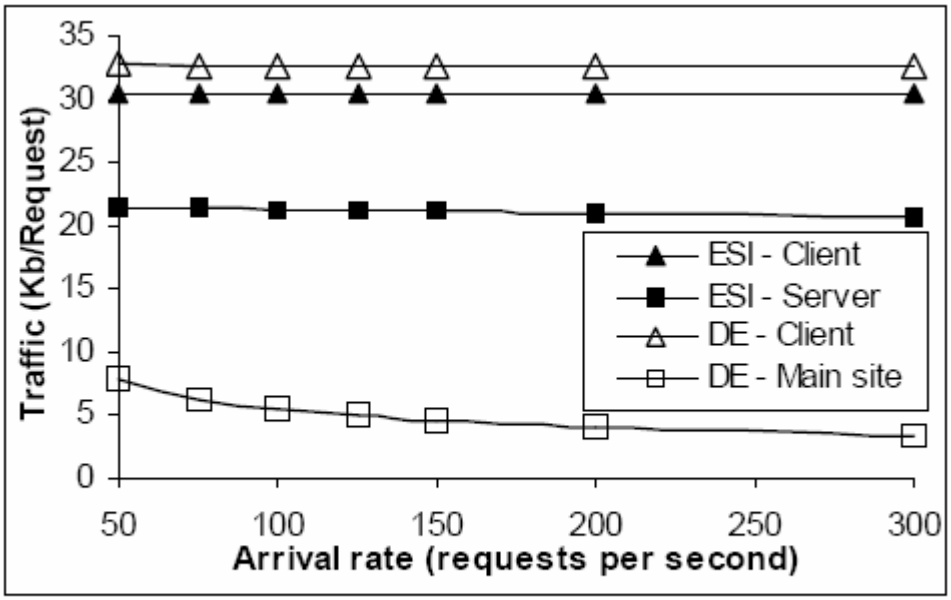
Στη συνέχεια, παρατίθενται τα αντίστοιχα διαγράμματα για το υποθετικό ηλεκτρονικό βιβλιοπωλείο, όπου λόγω της φύσεων των σελίδων, δύο διαδοχικές αιτήσεις έχουν μεγάλη πιθανότητα να είναι σε μεγάλο ποσοστό διαφορετικές, και έτσι ο πελάτης να αναγκάζεται με κάθε αίτηση να κατεβάσει από τον proxy cache server το αρχείο βάσης. Το γεγονός αυτό απεικονίζεται στα αποτελέσματα που φαίνονται παρακάτω, όπου αντί να παρουσιαστεί μείωση του όγκου κίνησης στον πελάτη, παρουσιάζεται μια ελαφρά αύξηση καθώ μεταφέρονται και τα βασικά αρχεία και οι διαφορές.

Online bookstore comparison

	Savings client link	Savings server link	Edge cache usage
ESI	0%	30%	1.2M
DE	-8%	82%	3.3M

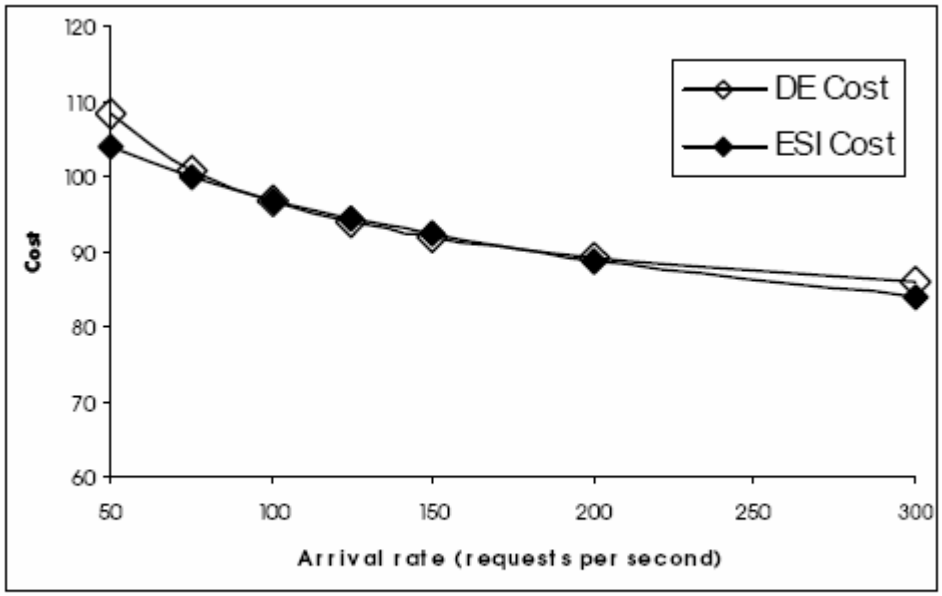
Στην περίπτωση αυτή, η καμπύλη του client traffic θα βρίσκεται ψηλότερα στο DE από ότι στο ESI, ενώ πάντα το DE εμφανίζει τα μεγαλύτερα κέρδη όσον αφορά την εξοικονόμηση εύρους ζώνης στον server.

Ακολουθεί το διάγραμμα του όγκου κινησης σε σχέση με τον ρυθμό άφιξης αιτήσεων:



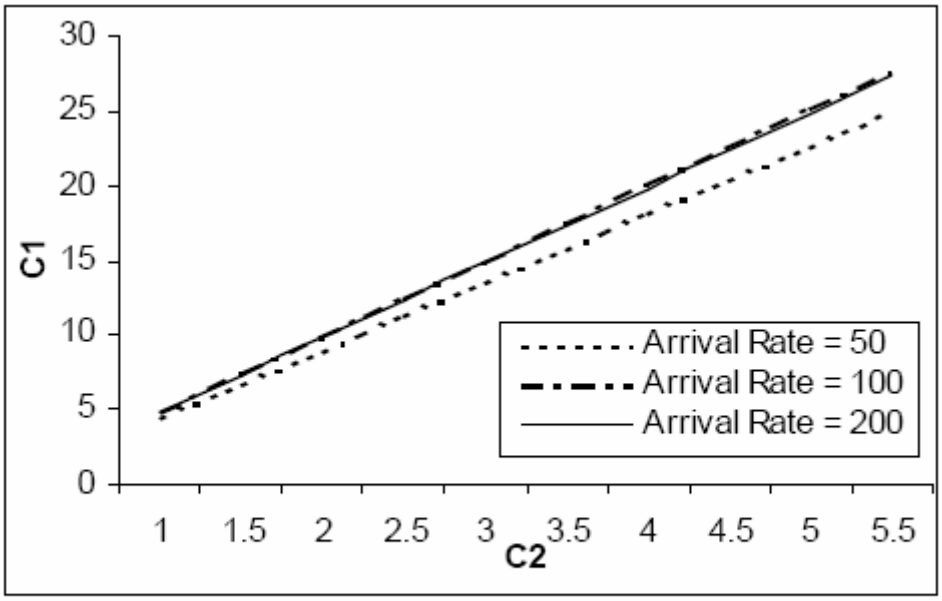
Online Bookstore resource, ESI and class-based DE traffic vs. request arrival rate

Και με σταθμισμένο κόστος όγκου κίνησης (C1=25, C2=5):



Online Bookstore, cost of ESI and class-based DE vs.. request arrival rate

Και το διάγραμμα απόφασης μεταξύ των δύο τεχνικών, ανάλογα με τις τιμές των C1, C2. Να σημειωθεί ότι σε αυτή την περίπτωση, οι λόγοι των C1, C2 για την ισότητα του κόστους των δύο τχνικών είναι αρκετά μικρότεροι, φανερώνοντας ότι σε αυτή την εφαρμογή το DE δεν είναι το ίδιο καλύτερο από ότι το ESI, όσο ήταν και στην εφαρμογή του “My Page”.



Online Bookstore, values for C1, C2 where DE cost is equal to ESI cost

Σαν κατακλείδα, ακολουθεί ένας πίνακας που συνοψίζει τα χαρακτηριστικά των δύο τεχνικών στα οποία παρουσιάζονται διαφορές.

Summary: Excellent *, Good +, Bad -, Sometimes ~

	ESI	DE
Reduces server traffic	+	*
Reduces client traffic	-	~
Reduces load on web server	*	-
Performance dependent on web page structure	Yes	Yes
Performance dependent on characteristics of data	Yes	No
Benefits greater when popularity rises	Yes	Less
Requires main site hardware/software installation	No	Yes
Requires web-page code changes	Yes	No
Requires network infrastructure (CDN services)	Yes	No
Can exploit information available from CDN for page construction	Yes	No

Το class based delta encoding απαιτεί την παραγωγή ολόκληρης της σελίδας με κάθε αίτηση, προκειμένου να δημιουργηθεί η διαφορά. Επίσης, υπάρχει κάποιο μη παραβλέψιμο υπολογιστικό κόστος για την εύρεση ενός καλού base file και τον υπολογισμό της διαφοράς. Επιπλέον κόστος στον πελάτη εισάγει και η διαδικασία της ανακατασκευής της απάντησης από το βασικό αρχείο και την διαφορά. Από την άλλη πλευρά η τεχνική ESI απαιτεί την διαδικασία της σύνθεσης της απάντησης στον edge-server, μια διαδικασία που όομως δεν είναι επιπλέον βάρος καθώς ούτως ή άλλως θα έπρεπε να

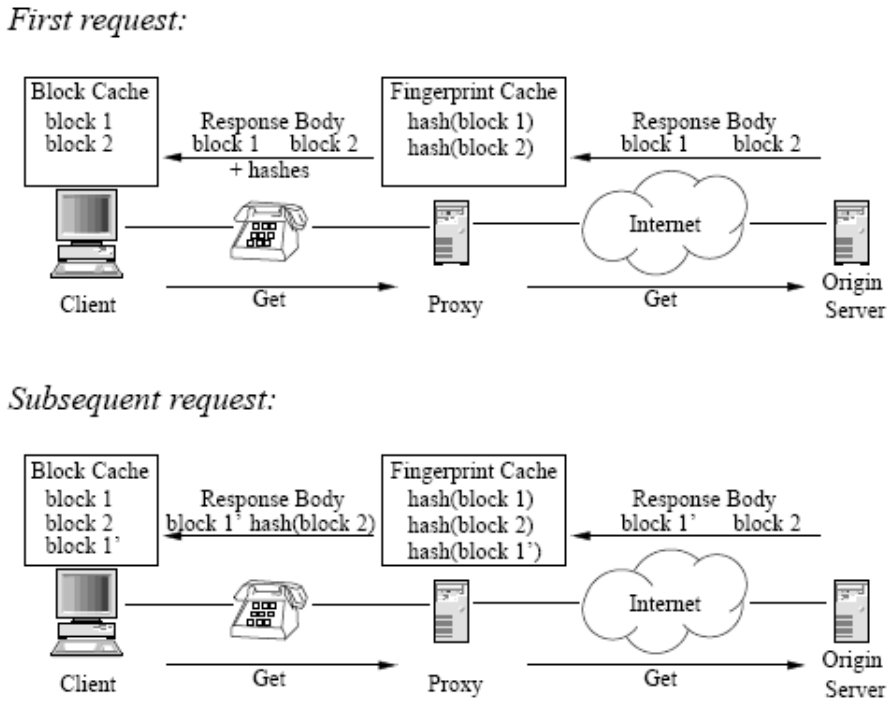
γίνει στον web server. Με την ESI γίνεται απλά μια μετακίνηση του κόστους αυτού από τον server στην cache.

Πέρα από το υπολογιστικό κόστος των δύο μεθόδων, υπάρχει το ζήτημα της διαφανούς εφαρμογής τους. Το DE, παρ' ότι απαιτεί την εγκατάσταση ειδικού υλικού ή λογισμικού στον server, το οποίο υπολογίζει τις διαφορές, κατά τα άλλα δεν χρειάζεται καμμία αλλαγή στον κώδικα της εφαρμογής που καλείται να επιταχύνει, ούτε και στην υπάρχουσα δκτυακή υποδομή από proxies και clients. Αντίθετα, η τεχνική του ESI απαιτεί την αλλαγή της εφαρμογής και του κώδικα των δυναμικών σελίδων, καθώς επίσης και την εκγατάσταση εξειδικευμένων edge-servers, που να μπορούν να συνθέσουν τις απαντήσεις από τα τμήματα τους.

Πάντως, και οι δύο τεχνικές μπορούν να μειώσουν σημαντικά τον συνολικό όγκο κίνησης και να βελτιώσουν το caching των δυναμικών σελίδων. Βέβαια, τα αποτελέσματα εξαρτόνται σε μεγάλο βαθμό από το είδος των πόρων. Για παράδειγμα, τα ESI είναι καλά για σελίδες που αποτελούνται από ένα μικρό αριθμό αντικειμένων, είναι αρκετά δημοφιλείς, και έχουν μεγάλο χρόνο ζωής. Το DE, είναι πάντοτε ωφέλιμο για τη μείωση του όγκου κίνησης στον server, δεν είναι πάντα το ίδιο καλό για τους πελάτες.

3.5 – Value Based Web Caching

Στο σημείο αυτό, και ενώ αναλύθηκαν και συγκρίθηκαν οι τεχνικές των DE και ESI, αξίζει να παρουσιαστεί συνοπτικά και μια άλλη ιδέα που αφορά το να γίνει εφικτό το caching δυναμικών πόρων. Η ιδέα αυτή εξετάζεται στην εργασία [Rhea et al. 2003] και αξιοποιεί το γεγονός ότι οι διαφορετικοί πόροι έχουν πολλές φορές ίδια τμήματα. Η τεχνική που προτείνεται είναι το Value Based Web Caching, και η οποία χωρίζει κάθε πόρο σε blocks και υπολογίζει μια τιμή κατακερματισμού για αυτό. Η λειτουργία φαινεται στο παρακάτω σχήμα:



Καθώς ο proxy λαμβάνει κάθε μπλοκ από τον server, παίρνει την τιμή κατακερματισμού του, αποθηκεύει το αποτέλεσμα στην cache και προωθεί το μπλοκ στον πελάτη. Όταν ένα μπλοκ ξαναστέλνεται σαν τμήμα της απάντησης μιας διαφορετικής αίτησης, μόνο η τιμή κατακερματισμού επαναμεταδίδεται στον πελάτη, ο οποίος από την τιμή αυτή μπορεί να δει ποιο από τα μπλοκ που έχει αποθηκευμένα αντιστοιχεί στο συγκεκριμένο μπλοκ που στέλενεται εκείνη την ώρα.

Η ιδέα αυτή είναι μια ενδιφέρουσα τεχνική για web caching, που στηρίζεται στις ίδιες παρατηρήσεις που στηρίζονται και τα DE και ESI. Παρ' όλα αυτά δεν έχει εξελιχθεί σαν μια τρίτη εναλλακτική προσέγγιση δυναμικού web caching όπως οι άλλες δύο, οπότε δεν θα αναλυθεί στον ίδιο βαθμό με τις άλλες δύο.

ΚΕΦΑΛΑΙΟ 4 : ΠΡΟΩΘΗΤΙΚΟ CACHING (PUSH CACHING)

4.1 – Εισαγωγή

Το παραδοσιακό caching λειτουργεί σε ένα μοντέλο pull, δηλαδή οι αλλαγές στους δικτυακούς πόρους μεταφέρονται στις caches μόνο όταν κάποιος τελικό χρήστης τις ζητήσει. Το μοντέλο αυτό λέγεται pull (σπρώξιμο) γιατί οι servers δεν συμμετέχουν ενεργά στην διαδικασία της ενημέρωσης των caches, αλλά απλά όταν τους ζητηθεί επιστρέφουν ένα ενημερωμένο αντίγραφο του εγγράφου. Το μειονέκτημα αυτού του μοντέλου είναι ότι δεν μπορεί να πετύχει απόλυτη συνέπεια μεταξύ των δεδομένων που βρίσκονται στον εξυπηρέτη και αυτών που δίνονται στον πελάτη όταν αθτός κάνει μια αίτηση για αυτά. Για παράδειγμα, επειδή στο pull μοντέλο caching χρησιμοποιείται κατά κύριο λόγο η έννοια του TTL (time to live) για την ακύρωση των σελίδων που βρίσκονται σε μια cache, μπορεί ένα έγγραφο να έχει δηλωθεί από τον server που το διανέμει ότι έχει χρόνο ζωής 1 ώρα, αλλά λόγω ιδιαίτερων περιστάσεων στο περιβάλλον της εφαρμογής να αλλάξει νωρίτερα, έστω στη μισή ώρα μετά την προηγούμενη δήλωση. Σε αυτή την περίπτωση, οι πελάτες που ζητούν αυτή τη σελίδα θα παίρνουν την πεπαλαιομένη απάντηση για μισή ώρα. Σε μερικές εφαρμογές αυτό μπορεί να είναι ανεκτό, αλλά σε άλλες είναι απαγορευτικό.

Την επίτευξη ισχυρής συνέπειας (consistency) μεταξύ των δεδομένων του server και των απαντήσεων που λαμβάνουν οι χρήστες μπορεί να διασφαλίσει ένα εναλλακτικό μοντέλο, αυτό του push caching. Το push caching προβλέπει την ενεργή συμμετοχή του server στην μετάδοση των αλλαγών του περιεχομένου στις caches. Το μοντέλο αυτό είναι λιγότερο διαδεδομένο από εκείνο του pull, καθώς απαιτεί μεγαλύτερο υπολογιστικό κόστος στην πλευρά του server και είναι σαφώς δυσκολότερο στην υλοποίησή του. Όμως, με την συνεχόμενη αύξηση των δυναμικών εφαρμογών και την ανάγκη για caching αρχιτεκτονικές που θα τις υποστηρίζουν, η ιδέα του push caching φαίνεται άκρως ενδιαφέρουσα. Αυτό γιατί αν ο server ενημερώνει την cache για τις αλλαγές των εγγράφων, τότε δεν θα υπάρχει πρόβλημα συνέπειας, το οποίο πρόβλημα είναι εκείνο το οποίο εμποδίζει την εφαρμογή του παραδοσιακού caching σε δυναμικά έγγραφα.

Πριν συνεχιστεί η συζήτηση για αρχιτεκτονικές push caching, πρέπει σε αυτό το σημείο να προσδιοριστεί ο τρόπος με τον οποίο μπορεί να εξεταστεί αν ένας δυναμικός πόρος έχει αλλάξει. Το ερώτημα αυτό είναι εύλογο, αφού ένας δυναμικός πόρος δεν είναι πουθενά αποθηκευμένος στον δίσκο του server αλλά δημιουργείται την ώρα της αίτησης, με αποτέλεσμα να μην μπορεί να παρακολουθηθεί για τον εντοπισμό των αλλαγών που συμβαίνουν σε αυτόν. Η απάντηση είναι ότι δεν χρειάζεται να παρακολουθείται ο ίδιος ο πόρος, που ούτως ή άλλως είναι στην ουσία μια αφηρημένη έννοια, αλλά τα δεδομένα τα οποία χρησιμοποιεί το script που δημιουργεί την δυναμική σελίδα. Τα δεδομένα αυτά είναι συνήθως αποθηκευμένα σε σχεσιακές βάσεις δεδομένων, οπότε αυτό που στην πραγματικότητα πρέπει να παρακολουθείται είναι οι εγγραφές στους πίνακες της βάσης.

Αφού, λοιπόν, ένας server εντοπίσει ότι ένας πόρος έχει πιθανώς αλλάξει, πρέπει να μεταδόσει με κάποιον τρόπο αυτή την πληροφορία στην cache. Οι διαφορετικές επιλογές που έχει για την μεταφορά αυτής της πληροφορίας

είναι δύο: είτε θα ενημερώσει την cache ότι ο συγκεκριμένος πόρος άλλαξε και θα πρέπει να διαγραφεί από την cache (invalidation), είτε θα ενημερώσει για την αλλαγή και ταυτόχρονα θα στείλει και ένα ενημερωμένο αντίγραφο στην cache προκειμένου να αντικαταστήσει το παλιό (update). Εδώ πρέπει να διευκρινιστεί ότι η δεύτερη συμπεριφορά, δηλαδή η αποστολή του νέου αρχείου, είναι αυτό που ονομάζεται push caching. Αντίθετα, η περίπτωση που γίνεται απλή ενημέρωση για την αλλαγή του πόρου εξακολουθεί να κατατάσσεται στο μοντέλο του pull caching, καθώς η λήψη των νέων αντιγράφων συνεχίζει να γίνεται με την αίτηση κάποιου τελικού χρήστη. Ωστόσο, είναι ένα ενδιάμεσο σχήμα caching που αποτελεί ένα πρώτο βήμα προς το push μοντέλο, και για αυτό θα εξεταστεί και αυτό στη συνέχεια του κεφαλαίου.

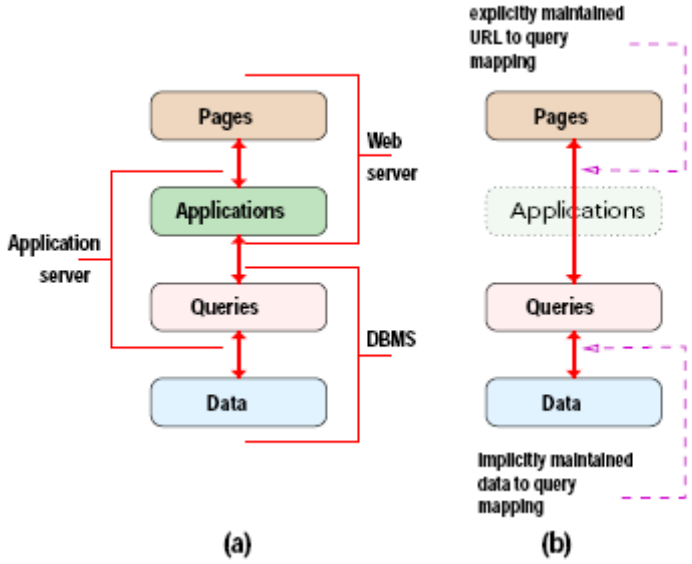
4.2 – Ακυροποίηση (Invalidation)

Η πρώτη από τις δύο διαδικασίες που περιγράφηκε στην προηγούμενη παράγραφο είναι αυτό που λέγεται invalidation. Όταν, λοιπόν, ένας server στέλνει με κάποιο τρόπο ένα μήνυμα στην cache για να την ενημερώσει ότι μια σελίδα έχει αλλάξει και δεν θα πρέπει να θεωρείται πλεον έγκυρη για να αποδοθεί στους χρήστες, τότε αυτή η σελίδα γίνεται invalidated.

Το κέρδος που προκύπτει από την διαδικασία του invalidation, είναι ότι αποφεύγεται η ασυνέπεια των απαντήσεων της cache και ότι ελευθερώνεται ο αποθηκευτικός χώρος της για να τον εκμεταλλετεί καλύτερα με την αποθήκευση κάποιου άλλου έγκυρου πόρου. Η ασυνέπεια αποφεύγεται διότι με την διαγραφή της σελίδας από την cache, μια επόμενη αίτηση για αυτή δεν θα εξυπηρετηθεί από την cache αλλά θα κατευθυνθεί μέχρι τον server, ο οποίος θα απαντήσει με την δυναμικά παραγόμενη σελίδα που θα είναι σίγουρα ενημερωμένη με τα νεότερα δεδομένα.

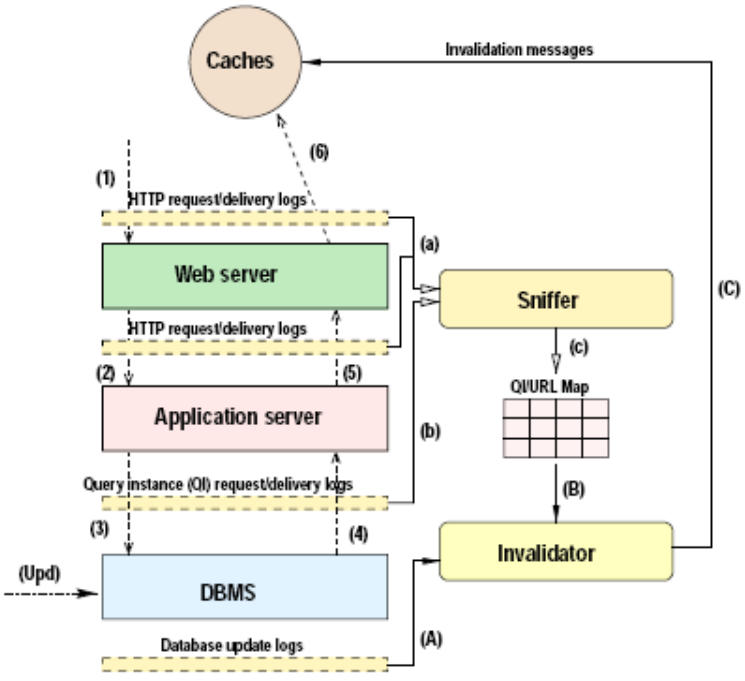
Στα πλαίσια της παραγράφου αυτής θα παρουσιαστεί ένας τρόπος για την αντιστοίχιση των αλλαγών της βάσης με τις δυναμικές σελίδες που επηρεάζουν και της ενημέρωσης της cache, καθώς και ένα πρωτόκολλο που χρησιμοποιείται την περιγραφή και αποστολή μηνυμάτων invalidation.

Η δημοσίευση των [Candan et al. 2001] αντιπροσωπεύει άριστα την μέθοδο του invalidation. Σε αυτή την εργασία, οι συγγραφείς χωρίζουν το πρόβλημα του εντοπισμού των αλλαγών μια δυναμικής σελίδας σε δύο υποπροβλήματα: την αντιστοίχηση μεταξύ των δυναμικών σελίδων και των SQL επερωτήσεων που χρησιμοποιούνται για την παραγωγή τους, και την αντιστοίχηση μεταξύ των αλλαγών στα δεδομένα της βάσης και τις επερωτήσεις των οποίων τα αποτελέσματα αυτές επηρεάζουν. Η ανάγκη για το διαχωρισμό του προβλήματος σε αυτά τα υποπροβλήματα είναι ότι για την απάντηση του χρειάζεται πληροφορία που βρίσκεται σε διαφορετικά υποσυστήματα της αρχιτεκτονικής, όπως φαίνεται στο παρακάτω σχήμα (α):

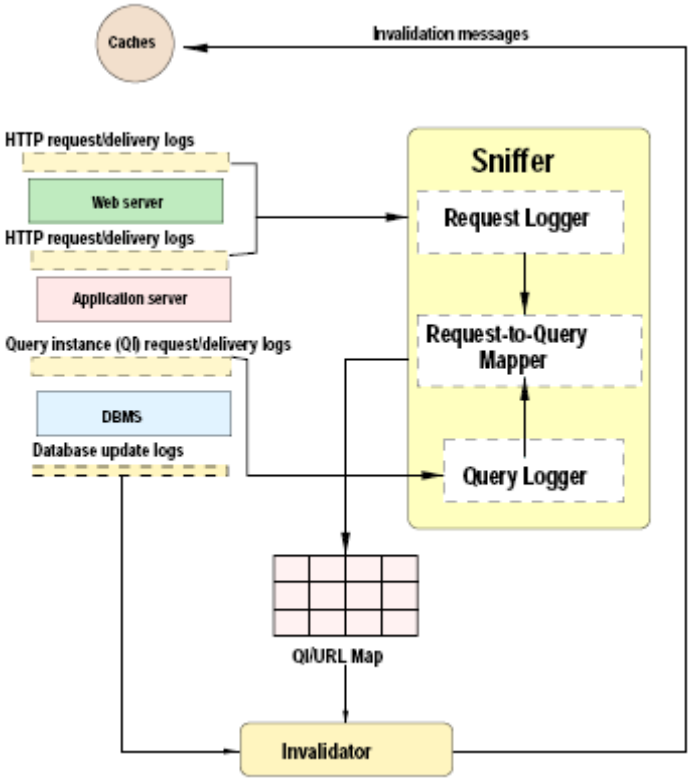


Τα δεδομένα που απαιτούνται για την απάντηση του προβλήματος είναι οι σελίδες, οι επερωτήσεις και τα δεδομένα, τα οποία βρίσκονται σε εντελώς διαφορετικούς servers, τον Web Server, τον Application Server και το Σύστημα Διαχείρισης Βάσεων Δεδομένων.

Με τη διάσπαση του προβλήματος (σχήμα β), η όλη διαδικασία μπορεί να διεκπεραιωθεί με δύο υποσυστήματα, εκ των οποίων το ένα παρακολουθεί τα logs του web server και τα logs του application server με σκοπό να κατασκευάσει την αντιστοιχία μεταξύ των δυναμικών σελίδων (των URL τους) και των επερωτήσεων που υποβάλλονται στη βάση. Το άλλο στέλνει τα invalidation μηνύματα στην cache εξετάζοντας τα logs της βάσης για τις ενημερώσεις που γίνονται σε αυτή. Τα δύο υποσυστήματα αυτά λέγονται αντίστοιχα sniffer και invalidator και τοποθετούνται στο σύστημα όπως φαίνεται στο παρακάτω σχήμα:



Η λειτουργία του Sniffer γίνεται με την παρακολούθηση των logs του web server και toy application server και για αυτό αποτελείται από δύο components που επιτελούν αυτήν την παρακολούθηση, τον request logger και τον query logger. Πέρα από αυτά, ο sniffer έχει και άλλο ένα component, το Request-to-Query Mapper, που είναι αυτό που κάνει την αντιστοίχηση μεταξύ των URLs και των επερωτήσεων SQL.



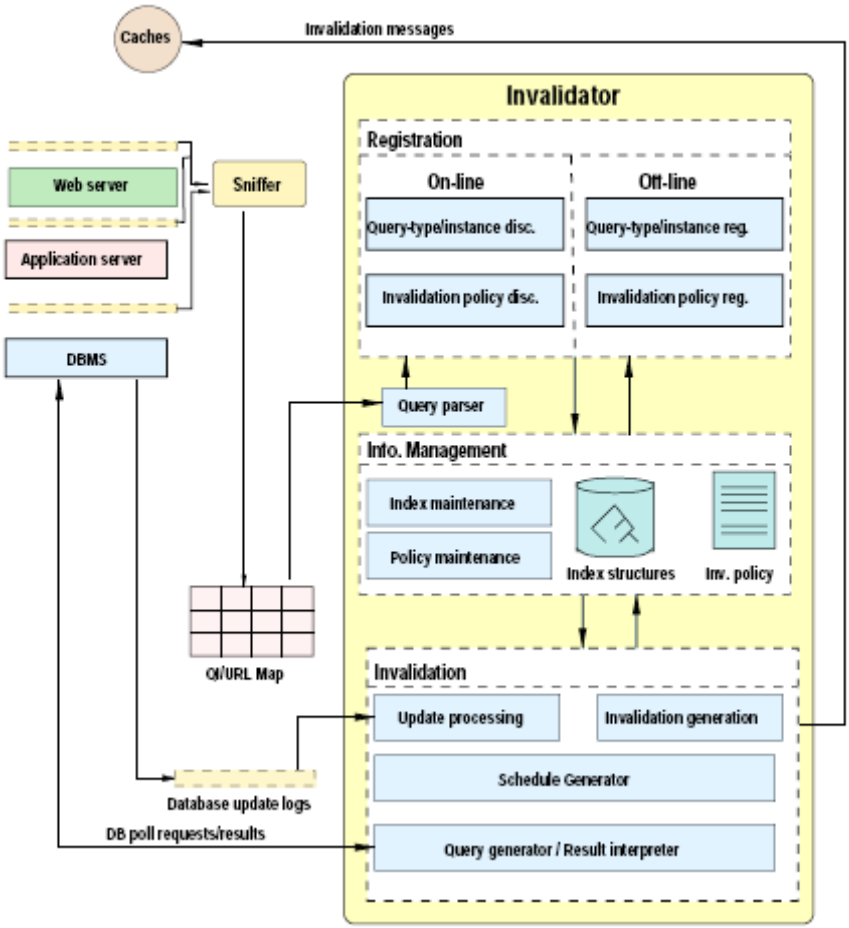
Ο Request Logger αποσπά και αποθηκεύει από τον web server την ακόλουθη πληροφορία: ένα μοναδικό ID για κάθε αίτηση, μια συμβολοσειρά της αίτησης που περιέχει το όνομα της σελίδας και τις GET παραμέτρους, μια συμβολοσειρά με τα Cookies, μια άλλη με τις POST παραμέτρους και δύο χρονοσημάνσεις (timestamps), μια για την άφιξη της αίτησης και μια για της αποστολή της απάντησης σε αυτή.

Ο Query Logger υλοποιείται με έναν wrapper του JDBC driver που χρησιμοποιείται ώστε κάθε επερώτηση που στέλνεται στη βάση να καταγράφεται. Η καταγραφή περιλαμβάνει την ίδια την επερώτηση, καθώς και δύο χρονοσημάνσεις, για τους χρόνους υποβολής επερώτησης και λήψης αποτελεσμάτων.

Ο Request-to-Query Mapper δημιουργεί την αντιστοιχεία μεταξύ Query Instances (QI) και URLs των δυναμικών σελίδων. Τα QI είναι στην ουσία ένα στιγμιότυπο από Query Types, δηλαδή τύποι επερωτήσεων που περιέχουν κάποιες μεταβλητές (κάτι σαν prepared statements). Η αντιστοιχία αυτή γίνεται με την εξέταση των χρονοσημάνσεων. Για κάθε αίτηση για μια σελίδα που έχει καταγραφεί στον web server, εξετάζονται οι χρόνοι άφιξης και αποστολής απάντησης, και έπειτα διατρέχεται το log του Query Logger για να ανιχνευτούν οι επερωτήσεις που υποβλήθηκαν σε αυτό το διάστημα στη

βάση. Ο Request-to-Query Mapper τότε αντιστοιχεί το URL της συγκεκριμένης αίτησης με τις επερωτήσεις που ανίχνευσε. Αυτή η αντιστοιχία είναι κάπως διφορούμενη καθώς επερωτήσεις μπορούν να υποβληθούν στη βάση στο ίδιο χρονικό διάστημα από πολλές εφαρμογές που τρέχουν στον application server ή από άλλες εξωτερικές πηγές. Ωστόσο, οι συγγραφείς προτίμησαν να υϊοθετήσουν αυτό τον τρόπο αντιστοίχησης γιατί η προσπάθειά τους κατευθυνόταν στην ανάπτυξη ενός συστήματος που θα είναι όσο το δυνατόν πιο απλό και ελαφρύ από άποψη υπολογιστικού φορτίου. Έτσι, απέρριψαν τη λύση της χρήσης σκανδάλων (triggers) στη βάση, καθώς θεώρησαν ότι θα επιβάρυνε σε μεγάλο βαθμό το Σύστημα Διαχείρισης Βάσεων Δεδομένων.

Το άλλο component της αρχιτεκτονικής είναι ο invalidator.



Ο invalidator αποτελείται από τρία αρθρωτά τμήματα λογισμικού (modules).

Το Registration Module είναι εκείνο που επιτρέπει στο διαχειριστή του συστήματος, στην offline λειτουργία του, να καθορίσει το τι θα πρέπει να γίνεται Invalidated με την κατάλληλη επιλογή των πολιτικών του invalidation. Επίσης, ο διαχειριστής καθορίζει Query Types που εκ των προτέρων γνωρίζει ότι πρέπει να παρακολουθούνται στη βάση. Αυτός ο κατάλογος των QTs και QIs επαυξάνεται και κατά την online λειτουργία του Registration Module, όπου διατρέχεται ο πίνακας των αντιστοιχιών QIs/URLs, που παράγει ο sniffer, και καταχωρούνται τα νέα QTs και QIs.

Το Invalidation Module είναι υπεύθυνο για την απάντηση στην ερώτηση για το πότε πρέπει να γίνει το Invalidation ενός πόρου. Η λειτουργία του περιλαμβάνει την παρακολούθηση των logs της βάσης για τον εντοπισμό των ενημερώσεων που συμβαίνουν στους πίνακες της, την μεταφορά πληροφοριών στο Information Management Module, την υποβολή βοηθητικών επερωτήσεων (Polling queries) στη βάση και τέλος την αποστολή των μηνυμάτων invalidation. Οι βοηθητικές επερωτήσεις ενεργούν σαν μια βολιδοσκόπηση στις αλλαγές της βάσης για να εξακριβωθεί αν όντως η αλλαγή που έγινε προκάλεσε αλλαγή και στη δυναμική σελίδα που πρόκειται να γίνει invalidated. Για παράδειγμα, αν μια σελίδα δείχνει τα αυτοκίνητα μάρκας Α, και έχει βρεθεί ότι εξαρτάται από μια επερώτηση του τύπου: SELECT * FROM cars WHERE brand='A', τότε μια ενημέρωση του πίνακα με τα αυτοκίνητα δεν θα αλλάξει το περιεχόμενο της σελίδας αν προστέθηκε μια εγγραφή για αυτοκίνητο μιας άλλης μάρκας Β. Με σκοπό να αποφευχθεί το μη αναγκαίο Invalidation, υποβάλλεται μια κατάλληλη polling query στη βάση, της οποίας τα αποτελέσματα κρίνουν το αν έχει επηρεαστεί το περιεχόμενο της σελίδας.

Το Information Management Module δημιουργεί βοηθητικές δομές δεδομένων που χρησιμοποιούνται από το invalidation module για την απόφαση της διενέργειας του invalidation.

Το μήνυμα αυτό καθ' εαυτό στο παραπάνω σύστημα γίνεται με τη χρήση ενός επεκταμένου HTTP πρωτοκόλλου που χρησιμοποιείται για την επικοινωνία invalidator και cache, το οποίο υποστηρίζει ένα cache directive το οποίο ενημερώνει για την διαγραφή μιας σελίδας από την cache. Το directive αυτό είναι το "Cache-Control: eject", και η cache που το υποστηρίζει είναι η [Network Appliance NetCache4.0]. Όμως, σαφώς, η επιλογή αυτή δεν είναι η καλύτερη δυνατή καθώς δεν αποτελεί κομμάτι του προτύπου του HTTP. Δυστυχώς, όμως, δεν υπάρχει κάποιο πρότυπο που να έχει προβλέψει για τις διαδικασίες του invalidation και του push caching.

Ωστόσο, υπάρχουν πολλές προτάσεις για τις διαδικασίες αυτές, που ίσως τα επόμενα χρόνια να γίνουν πρότυπα. Στη συνέχεια του κεφαλαίου θα παρουσιαστούν δύο από αυτά.

Το ένα αφορά μόνο τη διαδικασία του invalidation και είναι το κατάλληλο σημείο για να εξεταστεί. Πρόκειται για το ESI Invalidation Protocol 1.0 [ESI Technical Specs], το οποία αποτελεί τμήμα της όλης πρότασης των Oracle και Akamai για τα Edge Side Include, τα οποία έχουν παρουσιαστεί σε βάθος. Το ESI Invalidation Protocol 1.0 ορίζει το invalidation μηνυμα σαν ένα έγγραφο XML που στέλνεται στην cache με μια POST αίτηση. Το port για invalidation δεν είναι προκαθορισμένο στο HTTP οπότε χρησιμοποιείται αυθαίρετα από πολλές υλοποιήσεις το port 4001. Η αυθεντικοποίηση των αιτήσεων που ζητάνε το Invalidation κάποιας σελίδας στην cache γίνεται με την βασική αυθεντικοποίηση του HTTP (basic authentication).

Το σώμα μια αίτησης για invalidation είναι ένα έγκυρο XML έγγραφο στο οποίο δίνεται ένα ή περισσότερα αντικείμενα invalidation. Ένα invalidation αντικείμενο αποτελείται από ένα ζευγάρι selector-action, το οποίο καθορίζει

ότι σε όλα τα έγγραφα της cache που ταιριάζουν στον επιλογέα selector πρέπει να εφαρμοστεί η δράση action.

Η απάντηση της cache σε ένα invalidation μήνυμα γίνεται και αυτή πάνω από το πρωτόκολλο HTTP και είναι μια απάντηση με κωδικό επιτυχίας 200. Στο σώμα της απάντησης είναι και πάλι ένα XML έγγραφο, στο οποίο υπάρχει μια λίστα από αποτελέσματα invalidation αντικειμένων και αντιστοιχούν στα invalidation αντικείμενα που λήφθηκαν από την cache με την αίτηση για invalidation. Ένα αποτέλεσμα invalidation αντικειμένου αποτελείται από ένα ζευγάρι selector-result, όπου ο selector είναι ο ίδιος με εκείνον στην αίτηση και το result περιέχει το invalidation status για τον συγκεκριμένο πόρο που ζητήθηκε να γίνει invalidate. Παραδείγματα της αίτησης είναι το παρακάτω:

```
1       POST /x-invalidate HTTP/1.0
2       Authorization: Basic aW52YWxpZGF0b3I6aW52YWxpZGF0b3I=
3       Content-Length: 217
4
5       <?xml version="1.0" ?>
6       <!DOCTYPE INVALIDATION SYSTEM "invalidation.dtd">
7       <INVALIDATION VERSION="WCS-1.0">
8       <OBJECT>
9       <BASICSELECTOR URI="/cache.htm" />
10      <ACTION />
11      </OBJECT>
12      </INVALIDATION>
```

Οι selectors μπορεί να είναι basic ή advanced. Ένας basic selector είναι ένα κανονικό URL, ενώ ένας advanced είναι μια έκφραση που επιλέγει ένα σύνολο από σελίδες. Μια action είναι ένα κενό στοιχείο που δέχεται ένα προαιρετικό χαρακτηριστικό, το removalTTL. Σε όλες τις περιπτώσεις το action προκαλεί την ακύρωση της σελίδας στην cache. Η τιμή του removalTTL καθορίζει μετά από πόσο δευτερόλεπτα θα πρέπει να άκυρωθεί η σελίδα στην cache. Έτσι και απουσιάζει το χαρακτηριστικό αυτό, υπονοείται ότι η σελίδα πρέπει να αφεραιθεί αμέσως από την cache. Η απάντηση στην προηγούμενη αίτηση θα είναι η εξής:

```
1       HTTP/1.1 200 OK
2       Date: Sun, 22 Apr 2001 07:54:09 GMT
3       Allow: GET, HEAD
4       Server: Webserver/2.0.0.0.0
5       Content-Type: text/html
6       Content-Length: 284
7
8       <?xml version="1.0"?>
9       <!DOCTYPE INVALIDATIONRESULT SYSTEM "invalidation.dtd">
10      <INVALIDATIONRESULT VERSION="WCS-1.0">
11      <OBJECTRESULT>
12      <BASICSELECTOR URI="/cache.htm" />
13      <RESULT ID="1" STATUS="SUCCESS " NUMINV="1"/>
14      </OBJECTRESULT>
15      </INVALIDATIONRESULT>
```

Το κάθε αποτέλεσμα result έχει τρία πεδία, το id, το status και το numinv. Το id χρησιμοποιείται για να ξεχωρίζει τα αποτελέσματα, το status μπορεί να πάρει διάφορες τιμές, όπως "SUCCESS", "URI NOT FOUND", "URI NOT CACHEABLE" κ.α., ενώ το numinv έχει τον αριθμό των εγγράφων που έγιναν invalidated.

4.3 – Η HTTP Μέθοδος Push

Το επόμενο βήμα στη μετάδοση πληροφορίας από τον web server στην cache μετά την αποστολή του invalidation μηνύματος είναι να αποστέλλεται ταυτόχρονα και ο ανανεωμένος πόρος. Αυτό θα είναι μια αρχιτεκτονική push caching.

Προκειμένου να γίνει αυτό, πρέπει να προτοτυποποιηθεί ο τρόπος επικοινωνίας των εμπλεκόμενων συστημάτων στη διαδικασία. Μια πρόταση για ενσωμάτωση ενός μηχανισμού push στο πρωτόκολλο HTTP γίνεται στο [Chen et al. 1999], όπου προτείνεται μια νέα μέθοδος για το πρωτόκολλο που θα λέγεται PUSH.

Η μέθοδος PUSH είναι παρόμοια με τις μεθόδους POST και PUT που ήδη υπάρχουν στο HTTP/1.1. Μάλιστα είναι σχεδόν ίδια με την μέθοδο PUT, με την μόνη διαφορά ότι μια PUSH αίτηση απλά εισάγει το έγγραφο που βρίσκεται στο σώμα της σε μια cache, με άμεση συνέπεια μια PUSH αίτηση να μην ταξιδεύει στο δίκτυο πέρα από τα όρια του τρέχοντος cache proxy. Στην περίπτωση που μια PUSH αίτηση φτάσει σε μια φάρμα από cache proxies, η μετάδοση της ενημέρωσης σε όλες τις cache γίνεται με άλλους μηχανισμούς όπως το ICP [Wessels 1998].

Ο ορισμός της μεθόδου PUSH είναι ο εξής:

PUSH REQUEST
PUSH URL HTTP/version <crlf> [Request Header] [Entity body]

Όπου το “Entity body” έχει το HTTP αντικείμενο που θα εισαχθεί στην cache, το οποίο αποτελείται από μια επικεφαλίδα απάντησης του HTTP (HTTP response header) ακολουθούμενη από μια HTML σελίδα.

Η σύνταξη της PUSH απάντησης είναι:

PUSH REPLY
HTTP/version Status_Code Explanation <crlf> [Reply Header]

Η απάντηση σε μια PUSH αίτηση δεν έχει σώμα.

Ακολουθούν κάποια παραδείγματα για τις PUSH αιτήσεις και απαντήσεις:

1) PUSH request example:

```
PUSH http://www.foo.com/page.html HTTP/1.0
Content-length: 4090
HTTP/1.0 200 OK
Content-type: text/html
Content-length: 4061
<HTML>
à.
</HTML>
```

2) PUSH response example:

```
HTTP/1.0 200 OK
```

Μια τελευταία λεπτομέρεια είναι ότι μια cache που υποστηρίζει PUSH αιτήσεις θα πρέπει να είναι εξασφαλίσει ότι οι αιτήσεις που δέχεται είναι από τους εξουσιοδοτημένους χρήστες της. Στις παραδοσιακές caches για το λόγο αυτό υπάρχουν οι ACLs (Action Control Lists) που καθορίζουν ποιοι χρήστες μπορούν να εξυπηρετηθούν από την cache. Στην περίπτωση των push caches θα πρέπει να υπάρχουν και άλλες λίστες πρόσβασης, οι PCLs (Push Control Lists), οι οποίες θα καθορίζουν ποιοι web servers θα μπορούν να τις προσπελάσουν για να ανεβάσουν το περιεχόμενό τους στην cache.

4.4 – WCIP: Web Cache Invalidation Protocol

Αφού έγινε αντιληπτή η ανάγκη για invalidation και pushing μεθόδους στο web caching, εμφανίστηκαν προσπάθειες για πιο εμπεριστατωμένο ορισμό πρωτοκόλλων που θα εξυπηρετούσαν τις ανάγκες επικοινωνίας των μηχανισμών αυτών. Μια από αυτές τις προσπάθειες είναι και το πρωτόκολλο WCIP [Li et al. 2001], το οποίο είναι προς το παρόν ένα Internet Draft.

Στο WCIP δίνονται κάποιοι ορισμοί για τις οντότητες που συμμετέχουν στο πρωτόκολλο, οι πιο σημαντικές από τις οποίες φαίνονται παρακάτω:

Object Volume

Ένα σύνολο από συσχετιζόμενα δικτυακά αντικείμενα, μαζί με κάποια στοιχεία απαραίτητα για την συμμετοχή τους στη διαδικασία του invalidation και update.

Volume Synchronization

Η πράξη της ενημέρωσης (update) του object volume στην cache με το νέο έγγραφο του server. Το συγχρονισμό αυτό μπορούν να τον ξεκινήσουν είτε ο server, είτε η cache.

Last Synchronization Time

Ο χρόνος που έγινε ο τελευταίος συγχρονισμός.

Invalidation Channel

Είναι μια αφαιρετική αναφορά στο μέσο στο οποίο μεταφέρονται τα μηνύματα μεταξύ invalidation server και invalidation client (caches) Για το σκοπό του συγχρονισμού.

Invalidation Server

Μια εφαρμογή λογισμικού που παρέχει υπηρεσίες πρωτοκόλλου WCIP στις proxy caches. Ο invalidation server έχει το αρχικό αντίγραφο του object volume και στέλνει τα διάφορα Invalidation και update μηνύματα στις caches. Στη γενική περίπτωση, ο invalidation server διαφέρει από τον origin server, δηλαδή αυτόν που φιλοξενεί την εφαρμογή, αφού ένας invalidation server μπορεί να λειτουργεί για λογαριασμό ενός CDN και να εξυπηρετεί πολλούς διαφορετικούς servers.

Invalidation Client

Μια web cache, συνήθως caching proxy, που εγγράφεται σε ένα κανάλι για invalidation και από εκεί και στη συνέχεια διατηρεί ένα συνεπές αντίγραφο του object volume.

Revalidation Interval

Ο invalidation client ξεκινά ένα γύρο συγχρονισμού με τον invalidation server κάθε φορά που το "last synchronization time" έγινε πρίν από "revalidation interval".

Channel Address

Η πληροφορία που χρειάζεται μια proxy cache προκειμένου να έχει πρόσβαση στο κανάλι του invalidation, για παράδειγμα το όνομα του καναλιού, τη διεύθυνση του Invalidation server κ.α.

Channel Relay

Ένα ενδιάμεσο πρόγραμμα που εγγράφεται σε ένα ή περισσότερα κανάλια invalidation εξυπηρετόντας τους πελάτες του (downstream proxies) και επαναμεταδίδει τα μηνύματα του καναλιού σε αυτούς. Προφανώς, το πρόγραμμα αυτό πρέπει να υλοποιεί τη λειτουργικότητα και του Invalidation client και του invalidation server.

Heartbeat

Είναι ένα περιοδικό μήνυμα που στέλνει ο invalidation server για να αποφευχθεί η μακροχρόνια σιγή του καναλιού. Επιτρέπει στον Invalidation client να ελέγχει τη συνδεσιμότητα του καναλιού και να εξασφαλίζει ότι ο server δεν έχει κλείσει τη σύνδεση χωρίς να τον ενημερώνει για τις αλλαγές που συμβαίνουν.

Heartbeat Interval

Το χρονικό διάστημα που μεσολαβεί μεταξύ των heartbeats που στέλνει ο server όσο δεν έχει να στείλει ενημερώσεις για το object volume.

Η λειτουργία του πρωτοκόλλου είναι η εξής:

1) Με μια κανονική HTTP request-response συναλλαγή, μια caching proxy αποκτά την απαραίτητη πληροφορία για το invalidation channel από την HTTP response επικεφαλίδα "Invalidated-By", την οποία έχει συμπεριλάβει ο server στην απάντηση της HTTP αίτησης που έλαβε.

2) Για να συμμετάσχει στο κανάλι, η proxy cache εγκαθιδρύει μια διαμένουσα (persistent) HTTP σύνδεση με τον invalidation server, υποθέτοντας ότι η υλοποίηση του καναλιού είναι βασισμένη στο HTTP.

3) Αμέσως μετά την εγκαθύδριση της σύνδεσης, ο proxy πρέπει να εκκινήσει έναν γύρο συγχρονισμού (volume synchronization) για να αποκτήσει μια ενημερωμένη άποψη (όχι αναγκαία ενημερωμένα αρχεία, αλλά την πληροφορία για το αν έχουν αλλάξει) του object volume, και έτσι μια ενημερωμένη άποψη όλων των αντικειμένων μέσα σε αυτό.

4) Μετά τον αρχικό γύρο, ο invalidation server μπορεί να εκκινήσει έναν γύρο συγχρονισμού όταν ανιχνεύσει αλλαγές σε κάποια από τα αντικείμενα του object volume ή όταν το κανάλι είναι ανενεργό για χρόνο "heartbeat interval".

5) Κάθε φορά που ο proxy προσέξει ότι το "last synchronization time" έγινε πριν από μεγαλύτερο χρόνο του “revalidation interval", τότε πρέπει να εκκινήσει ένα γύρο συγχρονισμού.

6) Είτε ο proxy, είτε ο invalidation server μπορούν να τερματίσουν την επικοινωνία οποιαδήποτε στιγμή με το να κλείσουν την σύνδεση.

Ακολουθεί ένα παράδειγμα επικοινωνίας μεταξύ του invalidation server και invalidation client με σκοπό να γίνουν ακόμα πιο ξεκάθαρες οι έννοιες που εισάγει το WCIP.

Η λειτουργία του πρωτοκόλλου στην αρχή προβλέπει την εισαγωγή ενός “Invalidated-By” header στην HTTP απάντηση που στέλνει ο server και πρόκειται να περάσει από τον invalidation client. Αφού αντιληφθεί ο Invalidation client το κανάλι για το invalidation στέλνει την πρώτη αίτηση συγχρονισμού:

```
<?xml version="1.0"?>
<!DOCTYPE ObjectVolume SYSTEM "ObjectVolume.dtd">
<ObjectVolume channel="http://cdn.net:88/ch1" version="0">
</ObjectVolume>
```

Ο invalidation server απαντά με ολόκληρο το volume object:

```
<?xml version="1.0"?>
<!DOCTYPE ObjectVolume SYSTEM "ObjectVolume.dtd">
<ObjectVolume channel="http://cdn.net:88/ch1" version="7"
 base="0" date="Fri, 17 Nov 2000 08:22:17 GMT">
  <member op="include">
    <object name="amazon" fresh="120"
      uri="http://www.amazon.com/index.html"
      last-modified="Wed, 15 Nov 2000 04:52:01 GMT"
    />
    <object name="ebay" fresh="240"
      uri="http://www.ebay.com/index.html"
      last-modified="Thur, 16 Nov 2000 03:18:07 GMT"
      etag="yzxzyx"
    />
    <object name="cnn/allpolitics/" fresh="360"
      uri="http://www.cnn.com/allpolitics"
      last-modified="Fri, 17 Nov 2000 08:22:17 GMT"
    />
  </member>
</ObjectVolume>
```

Έστω ότι κάποια στιγμή ο client εκκινεί ένα γύρο συγχρονισμού. Τότε στέλνει ένα μηνυμα στον server, δηλώνοντας ότι έχει το volume object με version ‘7’:

```
 <?xml version="1.0"?>
<!DOCTYPE ObjectVolume SYSTEM "ObjectVolume.dtd">
<ObjectVolume channel="http://cdn.net:88/ch1" version="7"
base="7" date="Fri, 17 Nov 2000 08:22:17 GMT">
</ObjectVolume>
```

Αν δεν έχει αλλάξει το volume object, Ο invalidation server στέλνει την παρακάτω απάντηση:

```
<?xml version="1.0"?>
<!DOCTYPE ObjectVolume SYSTEM "ObjectVolume.dtd">
<ObjectVolume channel="http://cdn.net:88/ch1" version="7"
base="7" date="Fri, 17 Nov 2000 08:42:17 GMT">
</ObjectVolume>
```

Αντίθετα, αν έχει αλλάξει στέλνει την ακόλουθη απάντηση ενημερώνοντας την cache ότι πρέπει να αφαιρέσει ένα έγγραφο από το object volume και να θεωρήσει invalidated ένα άλλο:

```
<?xml version="1.0"?>
<!DOCTYPE ObjectVolume SYSTEM "ObjectVolume.dtd">
<ObjectVolume channel="http://cdn.net:88/ch1" version="9"
 base="7" date="Fri, 17 Nov 2000 08:32:17 GMT">
  <member op="exclude"> ;# exclude object(s) from the volume.
    <object name="amazon" fresh="120"
      uri="http://www.amazon.com/index.html"
      last-modified="Wed, 15 Nov 2000 04:52:01 GMT"
    />
  </member>
  <member state="stale">
    <object name="ebay" fresh="240"
      uri="http://www.ebay.com/index.html"
      last-modified="Thur, 17 Nov 2000 08:20:07 GMT"
      etag="yzkzyx"
    />
</member>
</ObjectVolume>
```

Αν τώρα, κάποια στιγμή ο server ανιχνεύσει αλλαγή στο object volume και θελήσει να ενημερώσει την cache, στέλνει το ακόλουθο μήνυμα:

```
<?xml version="1.0"?>
<!DOCTYPE ObjectVolume SYSTEM "ObjectVolume.dtd">
<ObjectVolume channel="http://cdn.net:88/ch1" version="10"
 base="9" date="Fri, 17 Nov 2000 08:32:17 GMT">
  <member state="stale">
    <object name="ebay" fresh="240"
      uri="http://www.ebay.com/index.html"
      last-modified="Tue, 24 Dec 2000 16:38:20 GMT"
      etag="37bb01a2-7ec-39f5bafc "
    />
  </member>
</ObjectVolume>
```

Αν, από την άλλη πλευρά, ο invaildation server πρέπει να στείλει ένα heartbeat, στέλνει ένα μήνυμα σαν αυτό:

```
<?xml version="1.0"?>
<!DOCTYPE ObjectVolume SYSTEM "ObjectVolume.dtd">
<ObjectVolume channel="http://cdn.net:88/ch1" version="10"
base="10" date="Fri, 17 Nov 2000 08:22:17 GMT">
</ObjectVolume>
```

όπου η έκδοση είναι η ίδια με αυτή που έχει η cache.

Η υποστήριξη του push μοντέλου στο WCIP γίνεται με έμεσο τρόπο, καθώς το πρωτόκολλο ορίζει ένα χαρακτηριστικό update=”yes|no” για τα στοιχεία <object>, με την προκαθορισμένη τιμή να είναι το “no”. Αν τεθεί στα χαρακτηριστικό update η τιμή “yes”, τότε το μόνο που ορίζεται από το πρωτόκολλο είναι ότι θα πρέπει με κάποιο τρόπο να σταλλούν από τον invalidation server στον invalidation client τα ενημερωμένα έγγραφα. Μάλιστα,

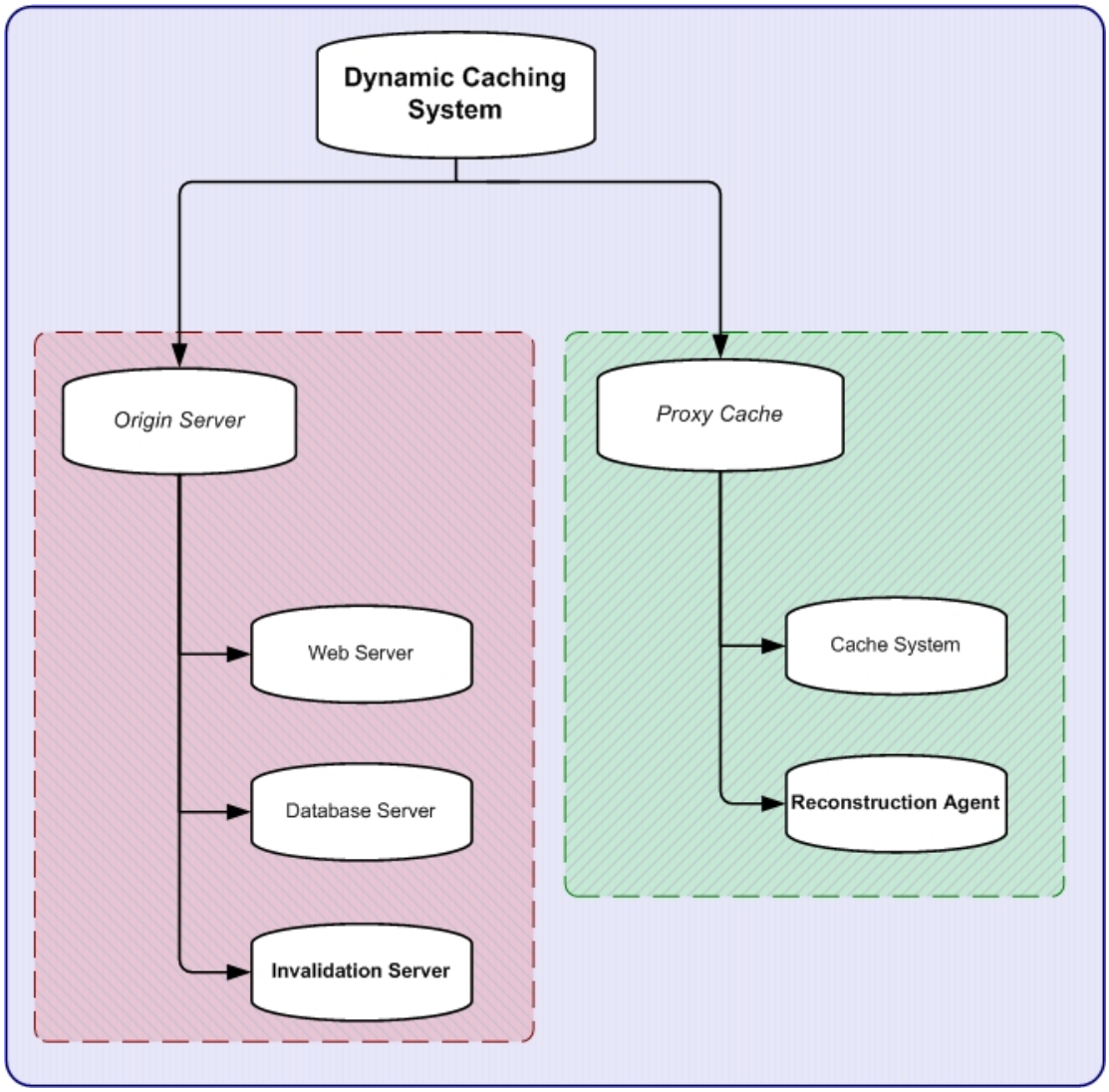
σε μια παλιότερη έκδοση του draft, οι συγγραφείς πρότειναν τη χρήση της μεθόδου PUT του HTTP για την αποστολή του νέου πόρου στην cache, παρ' ότι οι περισσότερες caches δεν υποστηρίζουν τη συγκεκριμένη λειτουργικότητα με την άφιξη μιας αίτησης PUT σε αυτές. Μόνο για την cache ανοιχτού λογισμού Squid έχει φτιαχτεί ένα patch [Squid Push Patch] το οποίο επιτρέπει στη μέθοδο PUT να εισάγει έγγραφα στον αποθηκευτικό της χώρο.

ΚΕΦΑΛΑΙΟ 5 : ΝΕΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗ – DE & Push Caching

5.1 – Γενική Επισκόπηση Αρχιτεκτονικής

Σε αυτή την εργασία γίνεται η παρουσίαση μιας αρχιτεκτονικής που συνδυάζει το Delta Encoding με το μοντέλο του push caching στα πλαίσια του caching δυναμικών ιστοσελίδων. Από όσο γνωρίζουμε, αυτή είναι η πρώτη προσπάθεια συνδιασμού των δύο μεθόδων.

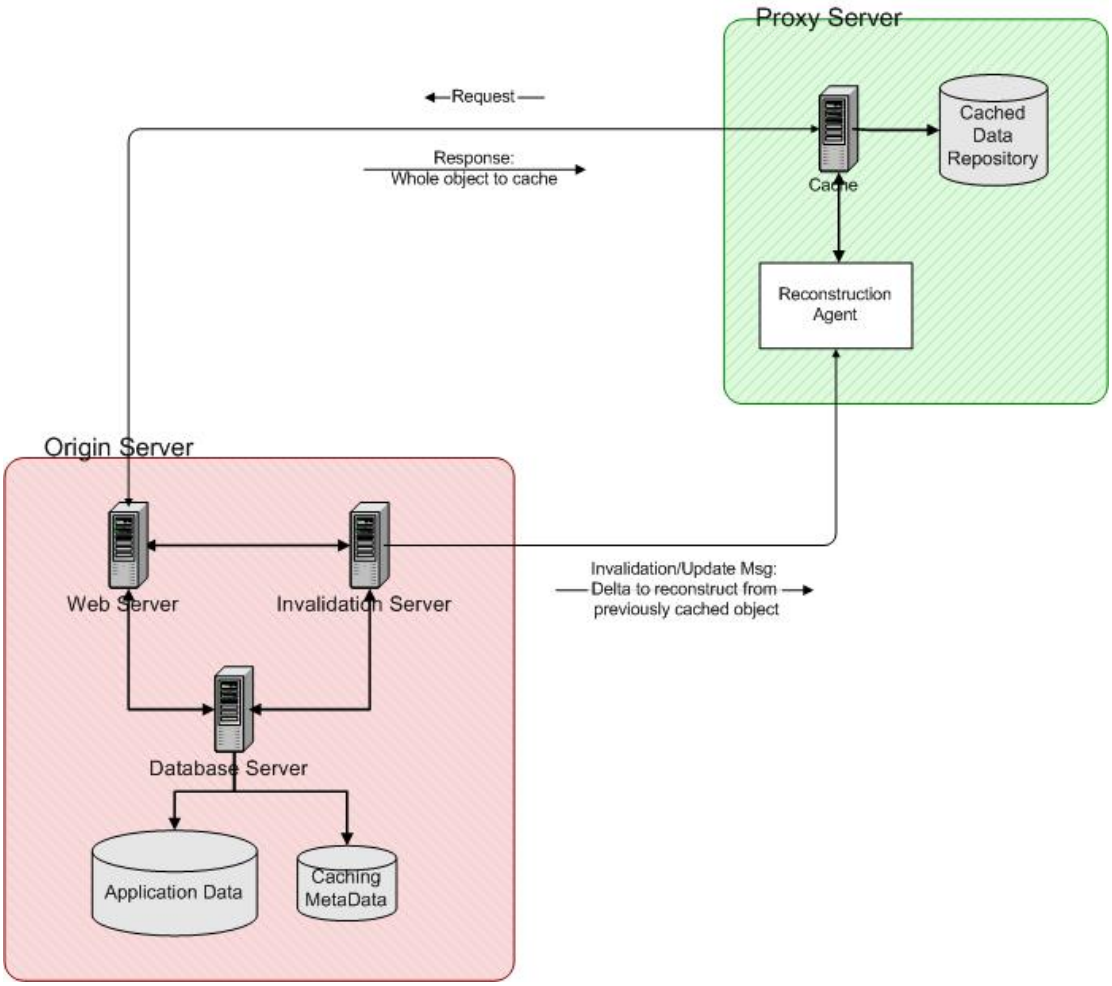
Η όλη αρχιτεκτονική περιλαμβάνει διάφορα συστήματα που μάλιστα βρίσκονται σε απομακρισμένες τοποθεσίες μέσα στο δίκτυο, όπως άλλωστε είναι αναμενόμενο στις αρχιτεκτονικές caching. Το σύστημα αποτελείται από υποσυστήματα τα οποία βρίσκονται είτε στα όρια του server, είτε στα όρια του proxy. Έτσι, μπορεί να χωριστεί λογικά, αλλά και χωρικά σε δύο βασικά υποσυστήματα: τον origin server και τον proxy. Κάθε ένα από αυτά τα υποσυστήματα αποτελείται από άλλα με τη σειρά του, τα οποία επιτελούν συγκεκριμένη λειτουργικότητα στην αρχιτεκτονική caching.



Ο origin server αποτελείται στην ουσία από τρεις εξυπηρέτες, τον Web Server, τον Data Base Server και τον Invalidation Server. Οι δύο πρώτοι είναι οι συνηθισμένοι servers που τρέχουν στις δικτυακές τοποθεσίες όλων των εταιρειών και οργανισμών που παρέχουν δυναμικές υπηρεσίες. Ο τρίτος, είναι ένας server που εισάγεται με την προτεινόμενη αρχιτεκτονική. Η λειτουργία του είναι μια συνένωση των λειτουργιών που επιτελούν ο delta server στο [Psounis 2002], ο invalidation server στο WCIP [Li et al. 2001] και το invalidator module στο [Candan et al. 2001]. Ο σκοπός αυτού του server είναι, κάθε φορά που γίνεται μια αλλαγή στο δυναμικό περιεχόμενο της εφαρμογής, να υπολογίζει τη διαφορά από την προηγούμενη έκδοση της σελίδας και να την στέλνει στον proxy. Στην προτεινόμενη αρχιτεκτονική επιλέχθηκε η ονομασία του server αυτού να είναι invalidation server καθότι αντικατοπτρίζει καλύτερα την κύρια λειτουργικότητά του: η δραστηριότητα του να προωθεί τις ενημερώσεις στον proxy είναι η βασική λειτουργία, και η χρήση του delta encoding είναι απλά ο τρόπος που χρησιμοποιείται για αυτό, τρόπος ο οποίος επιλέγεται καθαρά για λόγους βελτιστοποίησης χωρίς να αλλάζει τη λογική ροή των δεδομένων στο σύστημα.

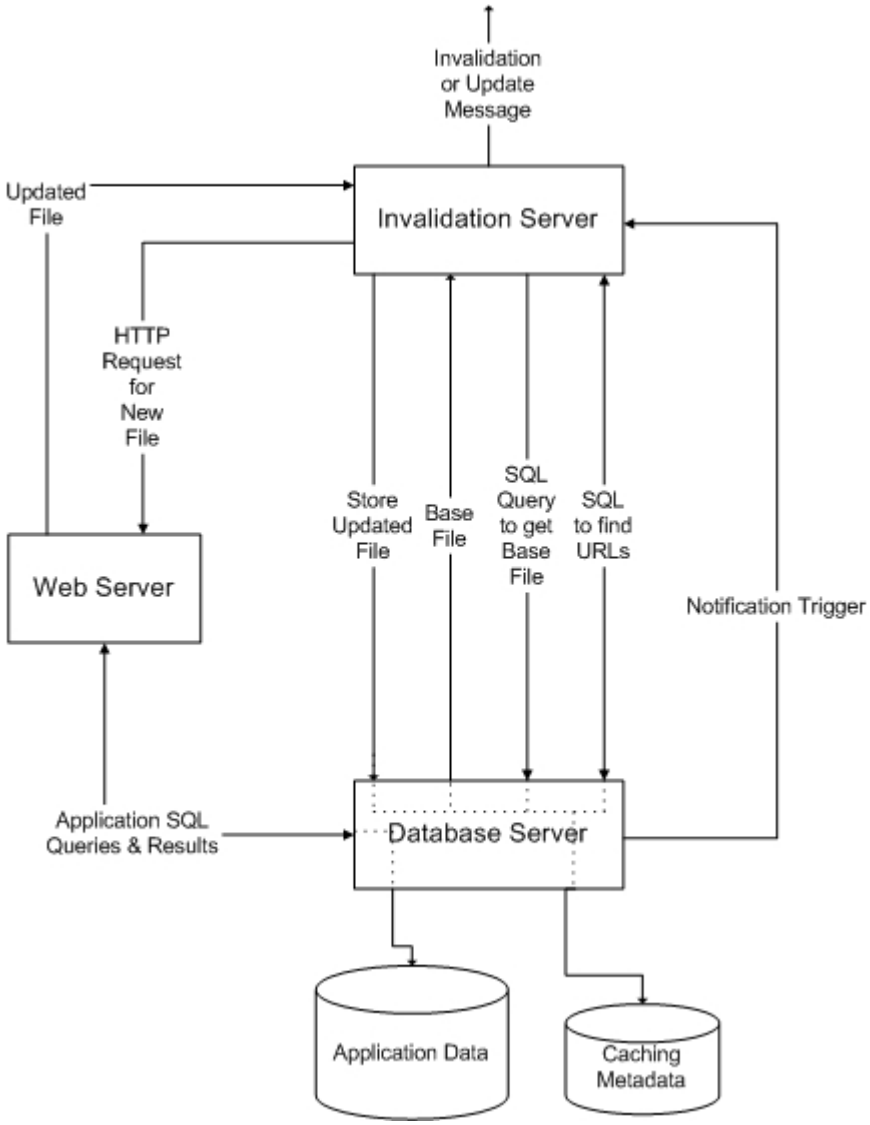
Ο proxy server είναι ένα υποσύστημα του οποίου η δομή καθορίζεται από την λειτουργία που πρέπει να διεκπεραιώνει. Συγκεκριμένα, στην διαδικασία της λήψης των ενημερώσεων, ο proxy server καλείται να ανακτήσει από την cache το βασικό αρχείο στο το οποίο θα εφαρμόσει τη διαφορά, να κατασκευάσει την νέα έκδοση του αρχείου και να το τοποθετήσει στην cache (πιθανότητα με μια αίτηση PUT ή PUSH). Επίσης, θα πρέπει να υποστηρίζει και κάποιο τρόπο για την απλή ακύρωση (invalidation) των σελίδων που βρίσκονται ήδη μέσα στην cache. Ο λογικός, λοιπόν, διαχωρισμός σε υποσυστήματα του proxy server είναι όπως φαίνεται στο παραπάνω διάγραμμα, όπου υπάρχει το σύστημα της cache που αποθηκεύονται οι σελίδες και τίθενται σε εφαρμογή οι διάφορες τεχνικές αντικατάστασης, και ένα σύστημα που εκτελεί τις υπόλοιπες διεργασίες που γίνονται στον proxy, όπως η ανακατασκευή του νέου πόρου.

Ακολουθεί ένα γενικό διάγραμμα που απεικονίζει την γενική δομή του συστήματος, καθώς και τις μεταξύ τους διαδράσεις. Αξίζει να σημειωθεί, ότι το σύστημα λειτουργεί σαν ένα παραδοσιακό σύστημα caching, ενώ παράλληλα υλοποποιεί και το μοντέλο push caching. Η παραδοσιακή λειτουργία είναι αυτή που απαιτεί την κλασική client – server ανταλλαγή μηνυμάτων μεταξύ web server και cache, με την όλη επικοινωνία να ξεκινά από την cache που παίζει το ρόλο του client για τον server και αιτεί μια σελίδα, όπως φαίνεται με την αίτηση και απάντηση από και προς τον web server στο σχήμα. Αντίθετα, η μέθοδος του pushing λαμβάνει χώρα με την επικοινωνία μεταξύ invalidation server και reconstruction agent, και την οποία εκκινεί σε όλες τις περιπτώσεις ο invalidation server από τη μεριά του εξυπηρέτη.



5.2 – Στην πλευρά του Server

Όπως παρουσιάστηκε στην προηγούμενη παράγραφο, στα όρια του origin server τρέχουν τρεις εξυπηρέτες: ο web server, ο εξυπηρέτης της βάσης δεδομένων και ο invalidation server. Αυτοί επικοινωνούν μεταξύ τους προκειμένου να γίνουν σωστά οι διεργασίες του δυναμικού συστήματος caching στην πλευρά του server. Η επικοινωνία μεταξύ τους φαίνεται στο παρακάτω διάγραμμα:



Ο Web Server δέχεται κανονικά αιτήσεις στην προκαθορισμένη πόρτα (port) από τους χρήστες του διαδικτύου που θέλουν να δουν και να χρησιμοποιήσουν την εφαρμογή, ενώ, όπως συνήθως, υποβάλλει (βέβαια, σε εφαρμογές μεγάλης κλίμακας χρησιμοποιείται και κάποιος/οι application server(s) που παρεμβάλλεται μεταξύ web και data base server) τις επερωτήσεις στη βάση και παίρνει τα αποτελέσματα που στη συνέχεια ενσωματώνει στις δυναμικά παραγόμενες σελίδες.

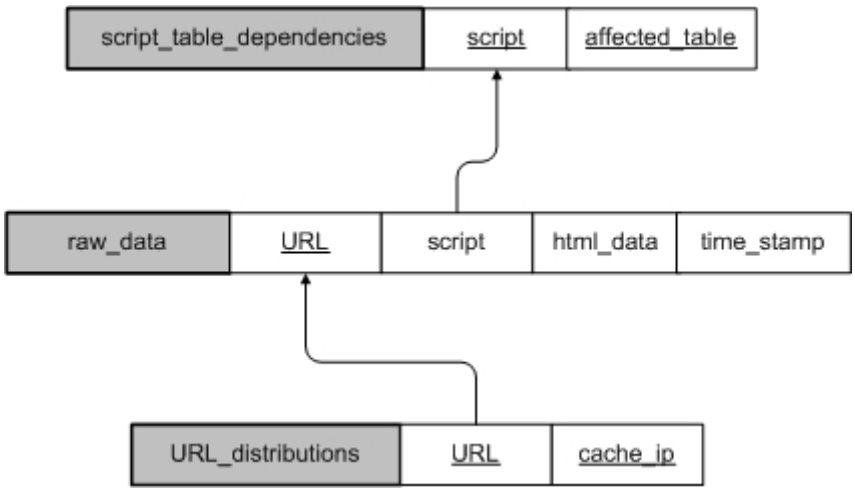
Αυτό που παρουσιάζει μεγαλύτερο ενδιαφέρον είναι η επικοινωνία του invalidation server με τους άλλους δύο. Η διαδικασία που ακολουθείται είναι η εξής: όλη η δραστηριότητα του συστήματος εκκινείται όταν στη βάση ανιχνευτεί κάποια αλλαγή στα δεδομένα κάποιου πίνακα της εφαρμογής. Τότε, στη βάση χτυπάει μια σκανδάλη (notification trigger) που ενημερώνει

τον inavlidation server για το ποιά scripts που παράγουν δυναμικές σελίδες αναμένεται να έχουν αλλάξει μετά τις διαφοροποιήσεις των δεδομένων της βάσης. Εκείνος με τη σειρά του υποβάλλει μια SQL επερώτηση στη βάση, προκειμένου να εντοπίσει στα μεταδεδομένα του caching ποια URLs, δηλαδή ποιες σελίδες έχουν αλλαγές στο περιεχόμενό τους. Στη συνέχεια, στέλνει μια αίτηση HTTP στον Web Server προκειμένου να λάβει σαν απάντηση την ενημερωμένη έκδοση της σελίδας. Αφού τη λάβει, ανακτά από τη βάση, με μια SQL επερώτηση, την παλιότερη έκδοση της σελίδας, ίδια με αυτή που βρίσκεται και στις caches. Στη συνέχεια, ο invalidation server υπολογίζει την διαφορά του παλιού με το νέο αρχείο και τη στέλνει στον proxy. Τέλος, στέλνει την καινούργια έκδοση του αρχείου πίσω στη βάση δεδομένων για να αποθηκευτεί και να χρησιμοποιηθεί σαν βασικό αρχείο στην επόμενη αλλαγή που θα πρέπει να προωθηθεί στον proxy.

5.3 – Η Βάση Δεδομένων

Ούτως ή άλλως η βάση δεδομένων που περιέχει τα αντικείμενα που τελικά εμφανίζονται στις δυναμικές σελίδες παίζει πολύ σημαντικό ρόλο για τις σύγχρονες εφαρμογές. Τον ίδιο σημαντικό ρόλο παίζει και στη διαδικασία του caching, όπου αφ' ενός παρακολουθεί τις αλλαγές στα δεδομένα για να εκκινήσει την λειτουργία του invalidation server και αφ' εταίρου φυλάει διάφορες πληροφορίες (caching metadata information) που είναι απαραίτητες για το caching.

Οι πληροφορίες που πρέπει να αποθηκεύονται στη βάση σχετικά με το caching είναι κατ'αρχάς οι σχέσεις μεταξύ δυναμικών scripts και πινάκων της βάσης, τα base files με βάση τα οποία υπολογίζονται οι διαφορές, και οι διευθύνσεις των proxy caches στις οποίες βρίσκεται κάθε URL, όπου και θα πρέπει να προωθηθούν οι παραγόμενες διαφορές. Παρακάτω φαίνεται το σχήμα της βάσης που έχει τα μεταδεδομένα του caching:



Στον πίνακα script_table_dependencies φαίνονται οι εξαρτήσεις μεταξύ των scripts τα οποία παράγουν τις δυναμικές σελίδες και των πινάκων της βάσης των οποίων δεδομένα παρουσιάζουν στους χρήστες της εφαρμογής. Αν για παράδειγμα στον πίνακα υπάρχει καταχωρημένη η εγγραφή (my_page.php, my_table) σημαίνει ότι το script my_page.php χρησιμοποιεί τα δεδομένα του πίνακα my_table της βάσης και πιθανότατα, αν αλλάξουν τα δεδομένα του τελευταίου, αλλάζει και το περιεχόμενο της δυναμικής σελίδας, οπότε και πρέπει να ξεκινήσει η διαδικασία του invalidation.

Ο τρόπος με τον οποίον αρχικοποιείται ο πίνακας script_table_dependencies είναι χειρονακτικός. Αυτό σημαίνει ότι ο διαχειριστής πρέπει να εισάγει στον πίνακα τις παραπάνω συσχετίσεις πριν το σύστημα αρχίσει να λειτουργεί. Η διαδικασία αυτή θα μπορούσε να αυτοματοποιηθεί, ώστε να μην υπάρχει η ανάγκη της παρέμβασης του διαχειριστεί, αλλά στη γενική περίπτωση αυτό δεν μπορεί να γίνει εύκολα. Σε μια απλή εφαρμογή, όπου το script της δυναμικής σελίδας περιέχει και τις SQL επερωτήσεις που αυτή υποβάλλει στην βάση, θα μπορούσε απλά να ελέγχονται τα “FROM” τμήματα των επερωτήσεων ενός script και να καταχωρούνται οι συσχετίσεις στον πίνακα. Αυτό βέβαια δεν μπορεί να γίνει τόσο απλά στις πολυστρωματικές (multi-tier) εφαρμογές, καθώς οι SQL επερωτήσεις βρίσκονται σε εντελώς διαφορετικά συστατικά (components) του συστήματος. Στο [Candan et al. 2001], οι συσχετίσεις αυτές υπολογίζονται με την παρατήρηση των αιτήσεων που φθάνουν στον web server και των SQL επερωτήσεων που υποβάλλονται στη βάση στο ίδιο χρονικό διάστημα. Βέβαια, παρ’ ότι ο μηχανισμός αυτός αυτοματοποιεί σε ένα βαθμό την διαδικασία, στερείται της αμφιμονοσήμαντης σχέσης μεταξύ πινάκων και scripts, αφού σε ένα χρονικό διάστημα, μπορεί να υποβληθεί στη βάση μια επερώτηση από μια διαφορετική εφαρμογή αντί εκείνης που γίνεται caching, με αποτέλεσμα οι συσχετίσεις να μην είναι πάντα αξιόπιστες. Τελικά, πάντα χρειάζεται (τουλάχιστον με τα σημερινά δεδομένα) η συμμετοχή του διαχειριστή στο συγκεκριμένο ζήτημα, καθώς και στο [Candan et al. 2001] δίνεται η δυνατότητα στον διαχειριστή να εισάγει QI/URL συσχετίσεις στο σύστημα.

Ο πίνακας raw_data περιέχει, για κάθε διαφορετικό URL που έχει φθάσει στον web server με κάποια αίτηση, το script από το οποίο κατασκευάζεται η απάντηση στο URL αυτό, η HTML της σελίδας που δώθηκε σαν απάντηση στην αίτηση και μια χρονοσφραγίδα (timestamp). Η καταγραφή των URLs που φθάνουν στον web server μπορεί να γίνει με διάφορους τρόπους, όπως με την δημιουργία ενός φίλτρου των αιτήσεων που φθάνουν ή φεύγουν από τον server (ο apache web server δίνει αυτή την δυνατότητα). Σε μια τέτοια περίπτωση, μόλις φθάνει μια αίτηση, θα καταγράφεται στη βάση το URL της, και όταν θα υπολογίζεται και επιστρέφεται η απάντηση, η αντίστοιχη HTML της σελίδας θα αποθηκεύεται στο URL το οποίο την δημιούργησε. Η εξακρίβωση του script από το οποίο κατασκευάζεται η απάντηση σε μια αίτηση είναι μια εύκολη και τετριμένη διαδικασία, καθώς πάντα στο URL υπάρχει αυτή η πληροφορία.

Ο πίνακας URL_distributions, είναι εκείνος που καταγράφει την πληροφορία του που (σε ποιες caches) βρίσκεται αποθηκευμένο κάθε URL. Η συμπλήρωση του πίνκα αυτού γίνεται κατά το φιλτράρισμα των αιτήσεων του

Web Server, όπως περιγράφηκε πριν. Κάθε cache έχει αποθηκευμένα πολλά URLs από διάφορες δικτυακές τοποθεσίες, ενώ και κάθε συγκεκριμένο URL μπορεί να βρίσκεται ταυτόχρονα σε πολλές caches. Αυτός ο πίνακας χρησιμοποιείται ώστε να ξέρει ο invalidation server που να στείλει τα μηνύματα ακύρωσης ή ενημέρωσης όταν κάποιο URL έχει αλλάξει περιεχόμενο. Η IP διεύθυνση της cache θα μπορούσε να είναι ένα ακόμα πεδίο στον πίνακα raw_data, όμως τότε θα υπήρχε πλεονασμός στη βάση, αφού θα αποθηκευόταν η ίδια HTML για κάθε διαφορετική cache όπου βρίσκεται.

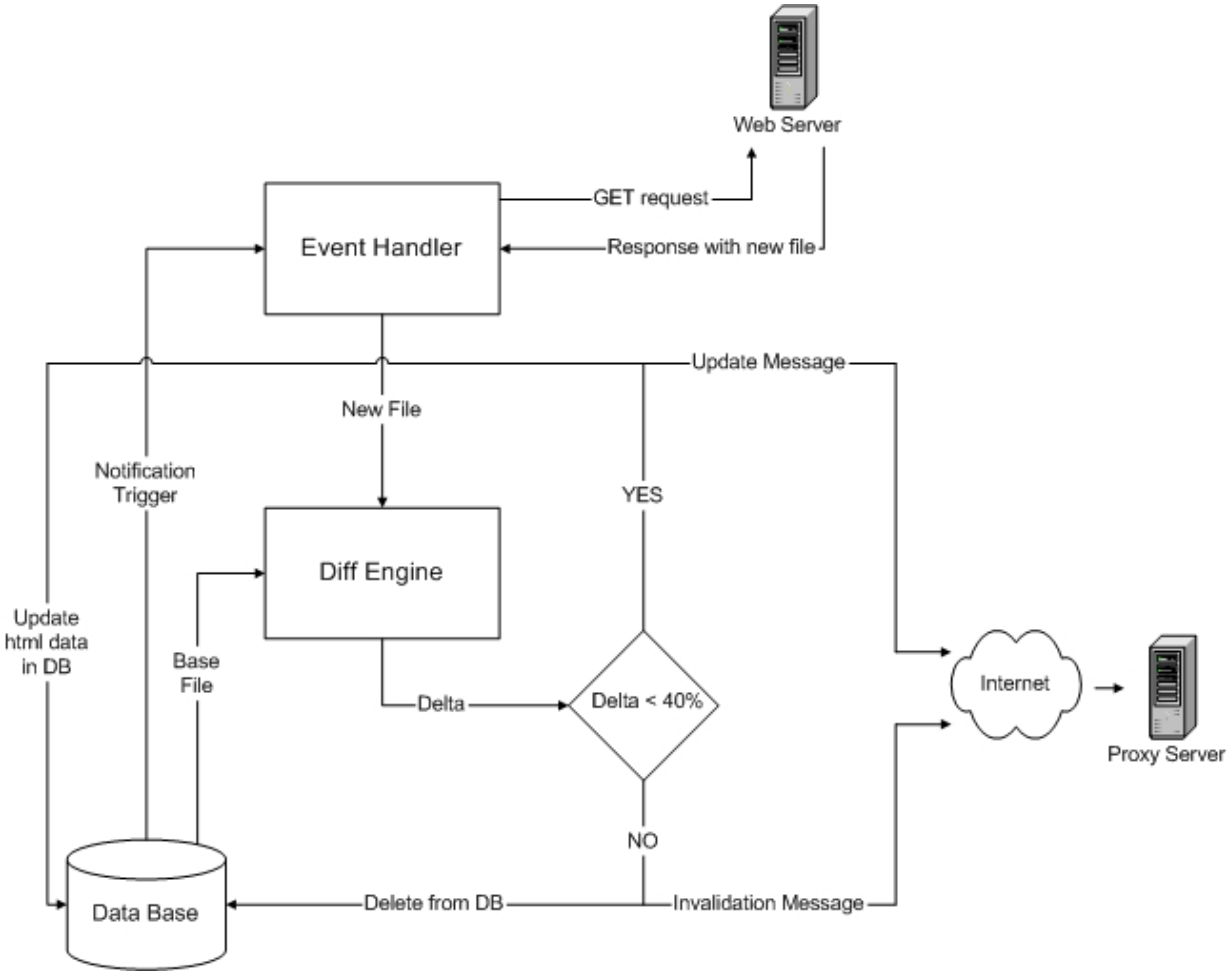
Η δεύτερη αποστολή του εξυπηρέτη βάσης δεδομένων εκτός από το να φυλάει τα μεταδεδομένα του caching είναι ότι παρακολουθεί τις αλλαγές που συμβαίνουν στα δεδομένα. Αυτό γίνεται με τον ορισμό ενός συνόλου από σκανδάλες (triggers). Συγκεκριμένα, ορίζονται τόσες σκανδάλες, όσοι και οι πίνακες του σχήματος της βάσης που χρησιμοποιεί η εφαρμογή (εξαιρουμένων των πινάκων των μεταδεδομένων του caching Που προαναφέρθησαν). Για κάθε πίνακα, λοιπόν, ορίζεται μια σκανδάλη που ενεργοποιείται κάθε φορά που τα δεδομένα του πίνακα ενημερώνονται (εισαγωγή, διαγραφή, ενημέρωση).

Ο κώδικας που τρέχει στη βάση με την ενεργοποίηση της σκανδάλης φροντίζει και συλλέγει από τον πίνακα script_table_dependencies, με βάση το όνομα του πίνκα στον οποίο κλήθηκε η σκανδάλη, τα scripts τα οποία πιθανόν επηρεάζονται με τις αλλαγές που έλαβαν χώρα. Στη συνέχεια, η μέθοδος που υλοποιεί την σκανδάλη στέλνει τα αποτελέσματα αυτά στον invalidation server που αναλαμβάνει να εκτελέσει ότι είναι απαραίτητο για την επίτευξη του δυναμικού caching.

5.4 – Ο Εξυπηρέτης Προώθησης Ενημερώσεων

Ο Εξυπηρέτης Προώθησης Ενημεερώσεων (Invalidation Server) είναι η καρδιά του δυναμικού συστήματος caching που προτείνεται στην εργασία αυτή, στη πλευρά του server. Είναι εκείνο το υποσύστημα που αναλαμβάνει να φέρει εις πέρας την ποιο απαιτητική, από πλευράς φόρτου εργασίας, διεργασία, εκείνη του υπολογισμού των διαφορών μεταξύ της παλιάς έκδοσης των αρχείων και της νέας.

Στο παρακάτω διάγραμμα απεικονίζονται οι λειτουργίες που επιτελούνται στα πλαίσια του invalidation server. Όλα αρχίζουν με την ενεργοποίησης κάποιας σκανδάλης στη βάση και την άφιξη του μηνύματος που περιέχει τα επηρεασμένα scripts, στον invalidation server. Σε πρώτη φάση, το μήνυμα δέχεται και επεξεργάζεται ο χειριστής γεγονότων (event handler), ο οποίος για κάθε script που μαθαίνει πως άλλαξε ξεκινά ένα νέο νήμα (thread), στο οποίο εκτελούνται όλες οι επόμενες φάσεις του συστήματος δυναμικού caching.



Για κάθε script, ο χειριστής γεγονότων αναζητά στη βάση ποια URLs έχουν απαντήσεις που κατασκευάζονται από αυτό το script (από τον πίνακα raw_data) και στέλνει για κάθε ένα από αυτά τα διαφορετικά URLs μια HTTP αίτηση στον Web Server προκειμένου να λάβει σαν απάντηση την νέα έκδοση του δυναμικού αρχείου. Αφού λάβει την απάντηση με το νέο περιεχόμενο, ανακτά από την βάση, πάλι από τον πίνακα raw_data το βασικό αρχείο (base file), σύμφωνα με το οποίο θα υπολογίσει τη διαφορά, και τα παιρνάει και τα δύο στη μηχανή διαφορών (diff engine).

Η μηχανή διαφορών υπολογίζει την διαφορά (delta) και ελέγχει μια συνθήκη, η οποία καθορίζει αν θα σταλλεί η διαφορά αυτή στον proxy. Ο σκοπός αυτού του ελέγχου είναι ότι αν η διαφορά είναι πολύ μεγάλη, τότε δεν υπάρχει κάποιο κέρδος στην αποστολή της στον proxy σε σχέση με την αποστολή ολόκληρης της σελίδας, δεδομένου και ότι η ανακατασκευή της απάντησης στον proxy εισάγει επιπλέον καθυστέρηση. Μια εμπειρική συνθήκη, λοιπόν, η οποία χρησιμοποιείται στο συγκεκριμένο σύστημα, είναι το μέγεθος της διαφοράς να μην υπερβαίνει το 40% του μεγέθους ολόκληρης της σελίδας. Βέβαια, η συνθήκη αυτή μπορεί να επιλεχθεί μεπολύ πιο αυστηρά κριτήρια, όπως αναλυτικά αποτελέσματα, πειραματικές μελέτες ή ακόμα και ευριστικά

που τρέχουν σε πραγματικό χρόνο και προσαρμόζουν την συνθήκη ανάλογα με το φόρτο του δικτύου και του server.

Στην κανονική περίπτωση που η διαφορά είναι μικρότερη από 40%, τότε γίνεται η προώθηση της διαφοράς στον proxy και η ενημέρωση της βάσης για το ποια έκδοση βρίσκεται αποθηκευμένη στις caches (πεδίο html_data γίνεται update στη βάση με τα νέα δεδομένα). Η προώθηση της διαφοράς γίνεται με μια POST HTTP αίτηση που αποστέλλεται στον proxy και η οποία περιέχει στο σώμα της την πληροφορία του ότι είναι ένα update μήνυμα (και πρέπει να γίνει push των νέων δεδομένων στην cache) και επίσης περιέχει την διαφορά (delta) που θα χρησιμοποιηθεί για να ανακατασκευαστεί η νέα απάντηση με βάση την παλιά.

Στην αντίθετη περίπτωση, όπου η διαφορά είναι μεγαλύτερη από 40%, υπάρχουν διάφορες εναλλακτικές λύσεις. Μπορεί είτε να αποσταλλεί ολόκληρη η απάντηση για να γίνει push στην cache, είτε απλά να ενημερωθεί η cache ότι το αρχείο έχει αλλάξει. Στη πρώτη επιλογή, πρέπει όπως και παραπάνω να ενημερωθούν τα δεδομένα που φυλάσσονται στη βάση, ενώ στη δεύτερη πρέπει να διαγραφεί από τα μεταδεδομένα του caching η εγγραφή του συγκεκριμένου URL, αφού θα διαγραφεί και από την cache. Στη δεύτερη περίπτωση, πρέπει να σημειωθεί ότι για τον συγκεκριμένο πόρο γίνεται εναλλαγή από το push μοντέλο caching σε αυτό του pull, το οποίο είναι και το κλασικό. Αν η cache δεχθεί αίτηση για αυτόν τον πόρο, επειδή δεν τον έχει αποθηκευμένο τοπικά, θα τον αναζητήσει από τον origin server, και θα το ανακτήσει με μια HTTP αίτηση, κατά τα γνωστά του pull μοντέλου. Αφού, όμως, παραλάβει την απάντηση θα επιστρέψει σε push caching mode καθώς ο server σημειώνει ποιοι πόροι υπάρχουν σε ποιες caches και φροντίζει να στέλνει τις διαφορές.

5.5 – Στη μεριά του Proxy

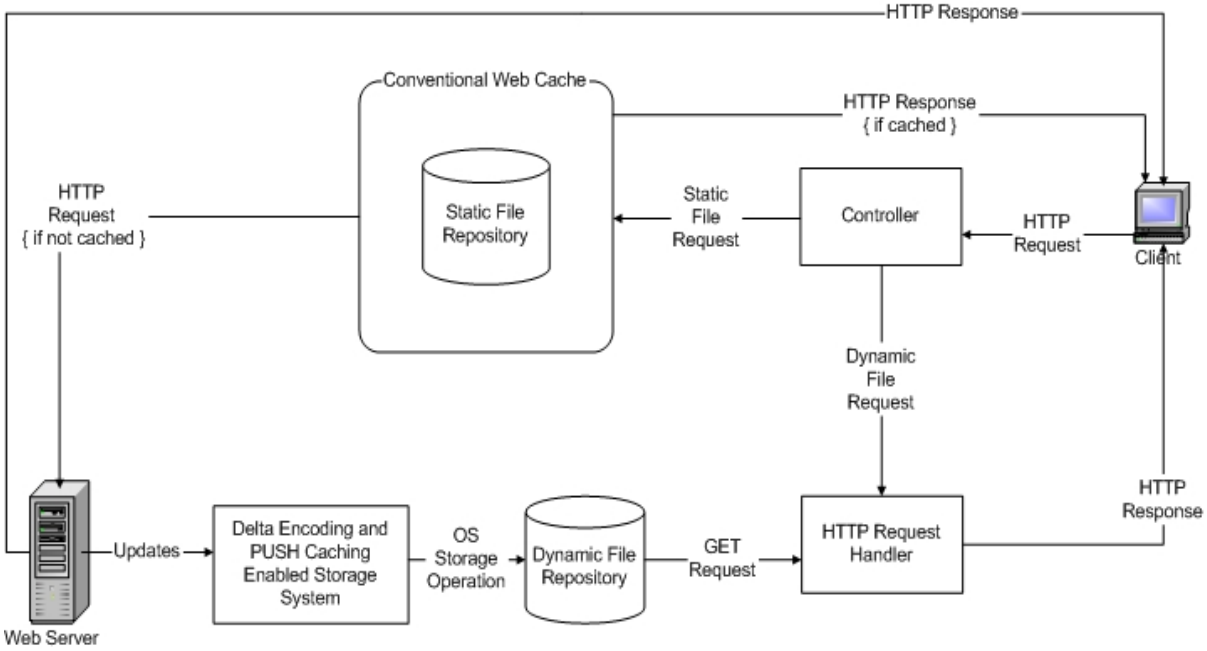
Στην πλευρά του proxy πρέπει να γίνει η ανακατασκευή του δυναμικού πόρου, με βάση το διαθέσιμο βασικό αρχείο και την διαφορά που έστειλε ο server, καθώς επίσης και η αποθήκευση της νέας έκδοσης στην cache. Συνεπώς, πρέπει κάπου να γίνει η διαδικασία του Delta Encoding και αυτή του του Push Caching. Οι δύο αυτές ανάγκες είναι εκείνες που εισηγούνται την αρχιτεκτονική του δυναμικού συστήματος caching στην πλευρά του proxy.

Ιδανικά, η ύπαρξη μιας push και DE enabled cache θα έκανε την αρχιτεκτονική του συστήματος τετριμμένη. Ο invalidation server θα επικοινωνούσε κατ' ευθείαν με την cache, και με την αποστολή του μηνύματος με τη διαφορά, η ανακατασκευή του αρχείου και η τοποθέτησή του στον αποθηκευτικό χώρο θα γινόταν εσωτερικά στο σύστημα της cache. Δυστυχώς, αυτή τη στιγμή δεν υπάρχουν κάποιες cache που να υποστηρίζουν και τις δύο αυτές λειτουργικότητες. Μάλιστα, με αρκετή δυσκολία μπορεί κάποιος να εντοπίσει κάποια cache που να υποστηρίζει οποιαδήποτε από τις δύο. Κατά τη γνώση μας, υπάρχει το xProxy [Ionescu 2000], ένα σύστημα cache που υποστηρίζει την μετάδοση μόνο της διαφοράς πάνω από τις συνδέσεις του δικτύου, και η Squid Cache [Squid Push Patch]

με το συγκεκριμένο patch της υποστήριξης της μεθόδου PUT για την τοποθέτηση κατά βούληση πόρων στην cache.

Το γεγονός της έλλειψης μιας cache που να ικανοποιεί και τις δύο ανάγκες του προτεινόμενου συστήματος οδηγεί στην εξέταση δύο εναλλακτικών αρχιτεκτονικών που μπορούν να χρησιμοποιηθούν για να υλοποιηθεί το σύστημα. Η πρώτη είναι εκείνη που το υλοποιεί χρησιμοποιώντας μια παραδοσιακή cache που δεν έχει κάποια χαρακτηριστικά επέκτασης σχετικά με τις ανάγκες του συστήματος. Η δεύτερη χρησιμοποιεί μια cache που υποστηρίζει το μοντέλο του Push caching, και στην οποία μπορούν να τοποθετηθούν έγγραφα στον αποθηκευτικό της χώρο με την υποβολλή μια απλής αίτησης.

Στην πρώτη επιλογή, για την υλοποίηση του συστήματος αρκεί η ανάπτυξη ενός ξεχωριστού από την κανονική cache αποθηκευτικού χώρου (dynamic resources repository), με περιορισμένη λειτουργικότητα όσον αφορά το παραδοσιακό caching αλλά με την δυνατότητα του υπολογισμού αρχείων με βάση τις διαφορές τους από κάποια άλλα και την αποδοχή και εξυπηρέτηση αιτήσεων PUSH. Ο χώρος αυτός θα προορίζεται μονάχα για τα δυναμικά αρχεία που θα γίνονται cache στο σύστημα. Επίσης, θα πρέπει να υπάρχει ένας ελεγκτής (controller), ο οποίος θα αναλαμβάνει να στείλλει κάθε εισερχόμενη αίτηση στο σωστό υποσύστημα. Ακολουθεί ένα ενδεικτικό διάγραμμα που επεξηγεί αυτή την αρχιτεκτονική:

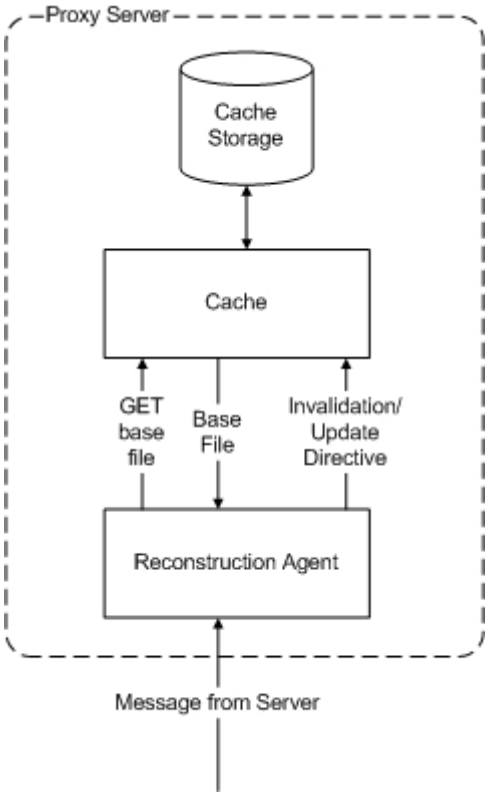


Βέβαια, η υλοποίηση ενός επιπλέον αποθηκευτικού χώρου caching δυναμικών πόρων είναι μια πολύπλοκη λύση που ίσως να προσεγγίζει σε δυσκολία την εξ' αρχής δημιουργία μιας cache που να υποστηρίζει το DE και το μοντέλο push. Ωστόσο, αυτή δεν είναι προς το παρόν τουλάχιστον μια καλή ιδέα, αφού τόσο οι αιτήσεις PUSH (ή PUT), όσο και ο μηχανισμός διαφορών που στηρίζεται στο Delta Encoding στο HTTP δεν έχουν

προτοτυποποιηθεί ακόμη. Για αυτό το λόγο, στην παρούσα εργασία θα δωθεί μεγαλύτερη έμφαση στην αρχιτεκτονική που χρησιμοποιεί την cache που υποστηρίζει το μοντέλο του push.

5.6 – Η Αρχιτεκτονική του Proxy

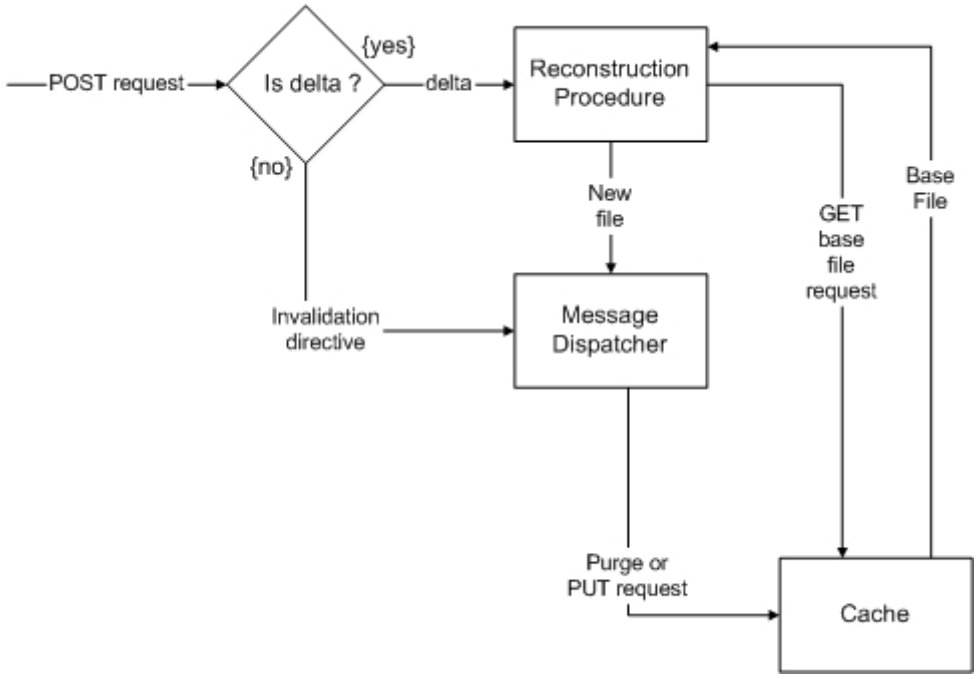
Στην αρχιτεκτονική αυτή υπάρχει μια cache που δέχεται αιτήσεις PUT και ένας Reconstruction Agent που εκτελεί την λειτουργία της ανακατασκευής του πόρου με βάση την διαφορά του από μια παλιότερη έκδοση. Ένα γενικό σχεδιάγραμμα φαίνεται παρακάτω:



Ο Web Server από την πλευρά του εξυπηρέτη επικοινωνεί αποκλειστικά με τον Reconstruction Agent (RA) όταν πρόκειται για την προώθηση ενημερώσεων, οπότε, ο RA αποτελεί τον πραγματικό server στη μεταξύ τους επικοινωνία. Η cache επικονωνεί με τον web server μόνο στην περίπτωση που δέχεται κάποια αίτηση για έναν πόρο που δεν έχει ήδη αποθηκεύσει στο παρελθόν (ή έχει ξεπεράσει τον χρόνο ζωής του), στην οποία περίπτωση το μοντέλο του caching που ακολουθεί το σύστημα εναλλάσσεται σε εκείνου του pull caching.

5.7 – Ο Πράκτορας Ανακατασκευής

Με την cache να υποστηρίζει εγγενώς το μοντέλο push, η πολυπλοκότητα του συστήματος μετατοπίζεται στο υποσύστημα του Πράκτορα Ανακατασκευής (Reconstruction Agent). Οι διαδικασίες του Reconstruction Agent και οι αλληλεπιδράσεις του με την cache φαίνονται στο παρακάτω διάγραμμα:



Ο Reconstruction Agent παρακολουθεί για αιτήσεις POST που έρχονται από τον server και μόλις ανιχνεύσει μία ξεκινά τις διαδικασίες του. Πρώτα, από την POST αίτηση εξετάζει αν πρόκειται για μήνυμα ενημέρωσης (update) ή απλά για ακύρωση (invalidation) ενός πόρου από την cache. Ανάλογα με την απάντηση στο ερώτημα αυτό, περνάνε στον διανομέα μηνυμάτων (Message Dispatcher) διαφορετικά δεδομένα. Στην περίπτωση του μηνύματος invalidation, ο Message Dispatcher στέλνει μια εντολή αφαίρεσης του πόρου (η εντολή αυτή λέγεται PURGE στην Squid Cache), που αναγράφεται στην POST αίτηση από τον server. Στην αντίθετη περίπτωση του μηνύματος ενημέρωσης, στον Message Dispatcher δίδεται το νέο αρχείο, το οποίο και φορτώνει στον αποθηκευτικό χώρο της cache με μια αίτηση PUSH (που στην [Squid Push Patch] είναι μια αίτηση PUT Που όμως έχει τη διαφορά ότι απευθύνεται στην ίδια την cache.

Για την κατασκευή του αρχείου που τελικά εισάγεται στην cache τίθεται σε εφαρμογή η διαδικασία της ανακετασκευής (reconstruction procedure), όπως άλλωστε φαίνεται στο παραπάνω σχήμα. Η διαδικασία αυτή δέχεται σαν είσοδο την διαφορά που στέλνει στην περίπτωση του update message ο server και τη χρησιμοποιεί για να δημιουργήσει το νέο αρχείο. Για να γίνει αυτό, πρέπει να ανακτηθεί η παλιά έκδοση του αρχείου. Για το κσοπό αυτό αξιοποιείται ότι, λόγω του Push μοντέλου, η παλιότερη έκδοση του αρχείου βρίσκεται ήδη στην cache. Έτσι, η reconstruction procedure στέλνει μια GET HTTP αίτηση στην cache για το εν λόγω αρχείο και ανακτά στην απάντηση της cache την βασική έκδοση του. Στη συνέχεια εφαρμόζει την διαδικασία της ανακατασκευής, και έχοντας στη διάθεσή του το βασικό αρχείο και την διαφορά δημιουργεί το νέο αρχείο.

5.8 – Πειραματικά Δεδομένα

Το προτεινόμενο σύστημα δυναμικού caching υλοποιήθηκε στα πλαίσια αυτής της εργασίας, μέχρι ένα βασικό επίπεδο λειτουργικότητας. Στην πλευρά του origin server χρησιμοποιήθηκαν ο Apache Web Server και η PostgreSQL σαν web server και database server αντιστοιχα, ενώ ο invalidation server αναπτύχθηκε σαν ξεχωριστό συστατικό γραμμένο σε γλώσσα Java για την εργασία. Για τον proxy υλοποιήθηκε εξ' αρχής ο reconstruction agent και επεκτάθηκε μια απλή web cache ώστε να υποστηρίζει τη λειτουργία του push.

Στον web server εγκαταστάθηκε μια απλοϊκή εφαρμογή διαδικτύου γραμμένη σε PHP που εμφανίζει όλα τα περιεχόμενα ενός πίνακα από τη βάση δεδομένων σε μια απλή HTML σελίδα.

Στη βάση δεδομένων, εκτός από τους πίνακες που είναι απαραίτητοι για το παρόν σύστημα δυναμικού caching, βρίσκονται οι πίνακες που χρησιμοποιεί η δοκιμαστική web εφαρμογή. Επίσης, πάνω σε κάθε πίνακα της εφαρμογής ορίζεται από ένα trigger το οποίο ενεργοποιείται σε κάθε αλλαγή που συμβαίνει πάνω στα δεδομένα του εν λόγω πίνακα. Οι σκανδάλες είναι και αυτές γραμμένες σε γλώσσα Java, κάνοντας χρήση μιας επέκτασης της PostgreSQL που δίνει τη δυνατότητα στους προγραμματιστές της βάσης να γράφουν αποθηκευμένες διαδικασίες και σκανδάλες στη γλώσσα αυτή. Μάλιστα, η επέκταση αυτή υπάρχει σε δύο εκδόσεις: μια «ασφαλή»(trusted) και μια «ανασφαλή»(untrusted). Για το σύστημα caching χρησιμοποιήθηκε η δεύτερη έκδοση, καθώς έπρεπε να μπορεί να συνδεθεί με τον invalidator server που βρίσκεται εκτός των ορίων του database server. Η διάκριση trusted-untrusted αφορά μονάχα την παρεχόμενη δυνατότητα επικοινωνίας με εξωτερικά της βάσης συστήματα, χωρίς να έχει να κάνει τίποτα με την πραγματική ασφάλεια του συστήματος.

Ο Invalidation server όταν ειδοποιείται από τις σκανδάλες της βάσης καλεί το πρόγραμμα diff (γνωστό utility του Linux που υπάρχει και για Windows) για να υπολογίσει τη διαφορά της νέας έκδοσης από την παλαιά, την οποία και στέλνει έπειτα στον proxy, σε συμπιεσμένη μορφή(ZIP).

Ο reconstruction agent στον proxy αποσυμπιέζει και εφαρμόζει την διαφορά αυτή πάνω στην παλιά έκδοση του αρχείου που έχει με την κλήση του προγράμματος patch (το συμπληρωματικό utility του diff).

Ακολουθούν μερικά αποτελέσματα και μετρήσεις που λήφθηκαν κατά τις δοκιμές του συστήματος. Η δοκιμή του συστήματος προέβλεπε την καταγραφή και επεξεργασία δεδομένων όπως το μέγεθος της διαφοράς (delta size), το μέγεθος της συμπιεσμένης διαφοράς(compressed delta size) και το ποσοστό αυτών σε σχέση με το μέγεθος ολόκληρου του αρχείου όπως αυτό θα στελνόταν σε περίπτωση που δεν χρησιμοποιόταν delta encoding.

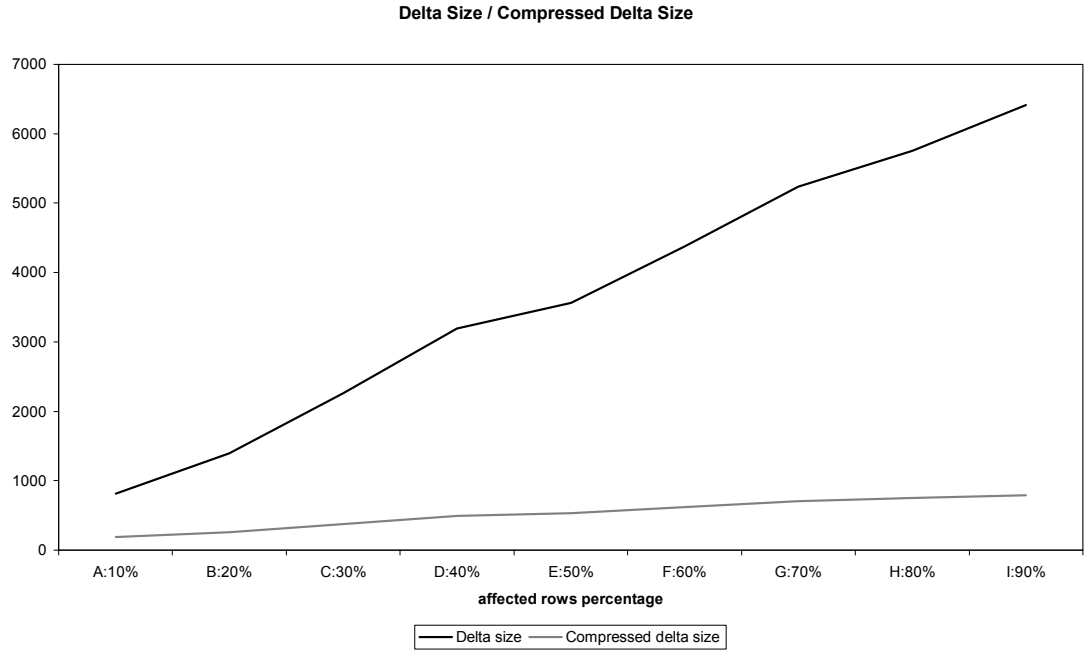
Αξίζει να σημειωθεί ότι το ποσοστό αυτό (delta size / new file size) εξαρτάται τόσο από το μέγεθος της διαφοράς, όσο και από το νέο μέγεθος του αρχείου σε σχέση με το παλιό. Προκειμένου να μελετηθεί καλύτερα η επιρροή του

μεγέθους της διαφοράς, το μέγεθος του νέου αρχείου φροντίστηκε να είναι πάντα το ίδιο (συγκεκριμένα 8079 bytes).

Το σενάριο της δοκιμής προέβλεπε την εκτέλεση μιας ενημέρωσης (update sql statement) στον πίνακα του οποίου τα δεδομένα αντικατοπτρίζονταν στην HTML σελίδα, η οποία ενημέρωνε μόνο τις εγγραφές που εκπληρούσαν μια συγκεκριμένη συνθήκη. Σε κάθε μια από τις διαφορετικές περιπτώσεις, τα δεδομένα του πίνακα είχαν εισαχθεί με τέτοιο τρόπο ώστε η πιθανότητα κάθε γραμμή να εκπληροί την συνθήκη αυτή να είναι {10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%}.

Ακολοθεί ένας πίνακας με το μέσο όρο των αποτελεσμάτων των διαφορετικών εκτελέσεων της δοκιμής για κάθε διαφορετικό ποσοστό επηρεασμένων εγγραφών:

Affected rows	Mean delta size	Mean compressed delta
10%	816,2	189,6
20%	1391,4	260,4
30%	2263,2	376
40%	3191,2	490,2
50%	3564,2	530,8
60%	4380,4	618,4
70%	5240	702,8
80%	5753,8	750
90%	6410,8	793,6

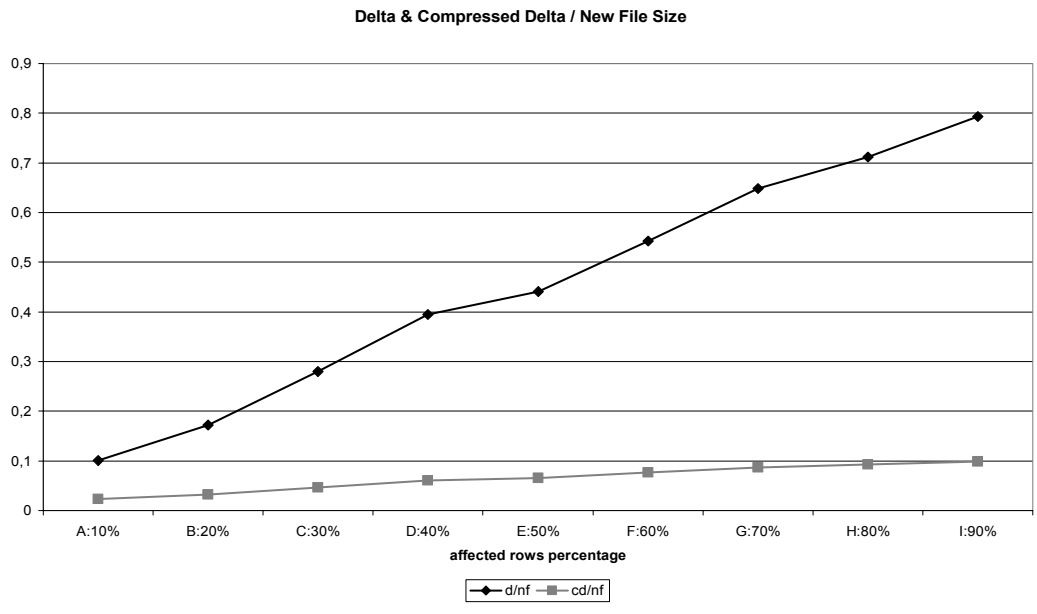


Όπως φαίνεται από το παραπάνω διάγραμμα, το μέγεθος των διαφορών (σε απλή και συμπιεσμένη μορφή) είναι ανάλογο (οι καμπύλες προσεγγίζουν την ευθεία γραμμή) του ποσοστού των εγγραφών που αλλάζουν στον πίνακα.

Αυτό άλλωστε ήταν και το αναμενόμενο αποτέλεσμα, καθώς οι αλλαγές του πίνακα εμφανίζονται όλες στη σελίδα. Αυτό που φαίνεται επιπλέον είναι μια πολύ μικρή καμπυλότητα που εμφανίζουν (κοίλα προς τα κάτω – είναι πιο εμφανές στην καμπύλη της συμπιεσμένης διαφοράς). Η εξήγηση για το φαινόμενο αυτό είναι ότι με την αύξηση των εγγραφών που συμμετέχουν στη διαφορά, αυξάνει και η πιθανότητα να υπάρχουν πολλές διαδοχικές εγγραφές που έχουν αλλάξει. Αυτό έχει σαν αποτέλεσμα η έξοδος του προγράμματος diff να έχει λιγότερα commands σε σχέση με τα δεδομένα της διαφοράς (το delta που προκύπτει από το diff αποτελείται από εντολές και δεδομένα π.χ. 4c New line of text.).

Επιπλέον, φαίνεται ξεκάθαρα πόσο μεγάλο είναι το κέρδος με τη χρήση συμπίεσης, γεγονός που καθιστά μονόδρομο την ενσωμάτωση της συμπίεσης για την αποστολή της διαφοράς σε κάθε σύστημα caching με delta encoding. Η συμπίεση έχει τόσο καλά αποτελέσματα διότι η διαφορά ειναι απλό κείμενο, που ως γνωστό έχει πολλά περιθώρια συμπίεσης. Αυτό που είναι ακόμα πιο ενθαρυντικό είναι ότι τα δυναμικά δεδομένα στο διαδίκτυο είναι κυρίως απλό κείμενο HTML. Υπάρχουν και δυναμικά παραγόμενα έγγραφα pdf, excel ή ακόμα και δυναμικές εικόνες, αλλά όλα αυτά σε πολύ πιο μικρή συχνότητα από ότι τα HTML έγγραφα. Τα αντίστοιχα ποσοστά των παραπάνω μεγεθών των διαφορών σε σχέση με το μέγεθος του αρχείου είναι στον παρακάτω πίνακα και διάγραμμα:

Affected rows	Mean delta size / new file size	Mean compressed delta size / new file size
10%	0,10102735	0,02346825
20%	0,17222429	0,03223171
30%	0,28013368	0,04654041
40%	0,39499938	0,06067583
50%	0,44116846	0,0657012
60%	0,54219582	0,07654413
70%	0,64859512	0,08699096
80%	0,7121921	0,09283327
90%	0,79351405	0,09822998



Το διάγραμμα δείχνει ότι οι καμπύλες είναι ακριβώς ίδιες σε μορφή με αυτές του προηγούμενου διαγράμματος. Αυτό συμβαίνει διότι, όπως αναφέρθηκε παραπάνω, το μέγεθος του νέου αρχείου παραμένει πάντα το ίδιο και έτσι ο λόγος του οποίου η τιμή παρουσιάζεται στο σχήμα εξαρτάται μονάχα από το μέγεθος της διαφοράς.

Μια δεύτερη παρατήρηση είναι ότι το ποσοστό δεν φθάνει το 100%, ακόμα και όταν δοκιμάστηκε για ενημέρωση όλου του πίνακα της εφαρμογής(100% affected rows – δεν υπάρχει στο διάγραμμα). Αυτό ωφείλεται στο ότι στην HTML σελίδα υπάρχουν εκτός από τα δεδομένα της βάσης και διάφορα στοιχεία που καθορίζουν το layout της (<table><tr><td> κτλ). Όπως διαπιστώθηκε στη δοκιμή που άλλαξε όλος ο πίνακας, το μέγεθος της διαφοράς (πάντα ασυμπίεστης) φθάνει μέχρι το 86%. Βέβαια, το σύστημα κανονικά σε μεγέθη διαφοράς άνω του 40% δεν στέλνει τη διαφορά στον proxy για να εξοικονομήσει το κόστος της ανακατασκευής. Ωστόσο, το άνω φράγμα του 86% δείχνει ότι το μέγεθος της διαφοράς δεν μπορεί να είναι μεγαλύτερο ολόκληρου του αρχείου. Αυτό το ποσοστό μάλιστα σε πραγματικές εφαρμογές με πολύπλοκο layout και πολλές συναρτήσεις javascript που συμπεριλαμβάνονται στον κώδικα της HTML Μπορεί να είναι αρκετά μικρότερο.

Τέλος, βλέποντας το ποσοστό της συμπιεσμένης διαφοράς, μπορεί κανείς να καταλάβει σε τι επίπεδα κινείται περίπου το μέγεθος των δεδομένων που στέλνεται στον proxy: όχι πάνω από 10% των δεδομένων που θα στέλνονταν χωρίς delta encoding στο παρόν παράδειγμα. Αυτό το εντυπωσιακό αποτέλεσμα δείχνει ότι μπορούν σίγουρα να εξοικονομηθούν σε μεγάλο βαθμό οι πόροι του δικτύου.

5.9 – Πιθανά οφέλη από την εφαρμογή του Συστήματος

Στην παράγραφο αυτή θα παρουσιαστούν συνοπτικά μερικές από τις εφαρμογές που μπορεί να έχει ένα σύστημα δυναμικού caching με delta encoding και υποστήριξη push αρχιτεκτονικής, δίνοντας έμφαση κυρίως σε οφέλη που προκύπτουν σε σχέση με τα απλά παραδοσιακά συστήματα caching. Πέρα από αυτό, θα αναλυθεί ποια από τα υπάρχοντα προβλήματα του delta encoding αντιμετωπίζονται με τη χρήση της push αρχιτεκτονικής και γιατί αυτές οι δύο τεχνικές μπορούν να συνδιαστούν με επιτυχή αποτελέσματα. Τέλος, δίνονται κάποιες ιδέες για αρχιτεκτονικές caching με τις δύο τεχνικές που προτείνονται στην εργασία, καθώς και προτάσεις για μελλοντική εξέλιξη των σχημάτων αυτών.

Κύρια προβλήματα του Delta Encoding

Αρχίζοντας, θα γίνει αναφορά των κύριων μειονεκτημάτων του Delta Encoding που ως σήμερα το εμπόδιζαν από το να εφαρμοστεί εκτεταμένα σε συστήματα web caching: 1) οι υπερβολικές απαιτήσεις ενός τέτοιου συστήματος σε αποθηκευτικό χώρο, και 2) η επιπλέον υπολογιστική ισχύς που χρειάζεται για την κατασκευή της διαφοράς και της ανακατασκευής του επιθυμητού αρχείου στον proxy.

Οι απαιτήσεις σε αποθηκευτικό χώρο προέρχονται από το γεγονός ότι στον server θα πρέπει να υπάρχουν όλα τα δυνατά βασικά αρχεία πάνω στα οποία θα υπολογίζεται η διαφορά. Αυτό σημαίνει ότι πρέπει να υπάρχουν αποθηκευμένες όλες οι διαφορετικές εκδόσεις κάθε πόρου που βρίσκονται σε όλες τις διαφορετικές caches σε όλο το διαδίκτυο! Το πρόβλημα γίνεται ακόμα πιο έντονο από το γεγονός ότι αυτοί καθ' εαυτοί οι δυναμικοί πόροι είναι πολλοί, για παράδειγμα μια δυναμική σελίδα αναζήτησης μπορεί να παράγει πολλές διαφορετικές σελίδες-αποτελέσματα ανάλογα με τα κριτήρια αναζήτησης.

Η υπολογηστική ισχύς είναι κάτι το απαραίτητο για τη λειτουργία ενός συστήματος caching με delta encoding καθώς η όλη διαδικασία αποτελείται από την επαναληπτική κλήση προγραμμάτων υπολογισμού διαφορών μεταξύ συμβολοσειρών, μια διαδικασία που είναι από μόνη της αρκετά χρονοβόρα και έχει υψηλό κόστος σε υπολογιστική ισχύ.

Αποθηκευτικός χώρος – μόνο μια κοινή έκδοση του πόρου σε κάθε cache
Το πρώτο μειωνέκτημα του αποθηκευτικού χώρου αντιμετωπίζεται σε μεγάλο βαθμό στην προτεινόμενη αρχιτεκτονική, αφού η τεχνική του push εξασφαλίζει ότι όλες οι caches θα έχουν μία και μόνο κοινή έκδοση του αρχείου, αφού ο invalidation server θα έχει φροντίσει για την ενημέρωση όλων μετά από κάθε μεταβολή. Με τον τρόπο αυτό δεν απαιτείται η αποθήκευση ενός βασικού αρχείου ανά cache και URL, αλλά μόνο ανά URL. Βέβαια, οι δυναμικοί πόροι που απομένουν είναι πάλι πολλοί (όχι όμως εξωφρενικοί), ωστόσο πάντα μπορεί να εφαρμοστεί σε συνδιασμό με το προτεινόμενο σύστημα και η τεχνική του Class Based Delta Encoding ([Psounis 2002]) που είναι ειδικά σχεδιασμένη για την αντιμετώπιση του προβλήματος του μεγάλου αποθηκευτικού χώρου στο DE.

Υπολογιστική ισχύς – προβλέψιμο το φορτίο του server
Όσον αφορά την απαιτούμενη υπολογιστική ισχύ, είναι ένα πρόβλημα που είναι εγγενές στα συστήματα DE με αποτέλεσμα να μην μπορεί να αντιμετωπιστεί τελείως. Παρ' όλα αυτά, η βασική βελτίωση σε αυτό τον τομέα με την προτεινόμενη αρχιτεκτονική είναι ότι ο υπολογιστικός φόρτος είναι προβλέψιμος και ελέγξιμος. Συγκεκριμένα, σε ένα σύστημα που το delta encoding συνδυάζεται με την κλασσική pull αρχιτεκτονική, ο φόρτος αυξάνεται με τον αριθμό των αφικνυόμενων αιτήσεων στον server, αφού για κάθε διαφορετική αίτηση εκτελείται και μια κατασκευή διαφοράς. Αυτό έχει σαν άμεσο αποτέλεσμα σε ώρες αιχμής να κινδυνεύει ο εξυπηρέτης από υπερφόρτωση και κατάρρευση (αν δεν ληφθούν άλλα μέτρα).

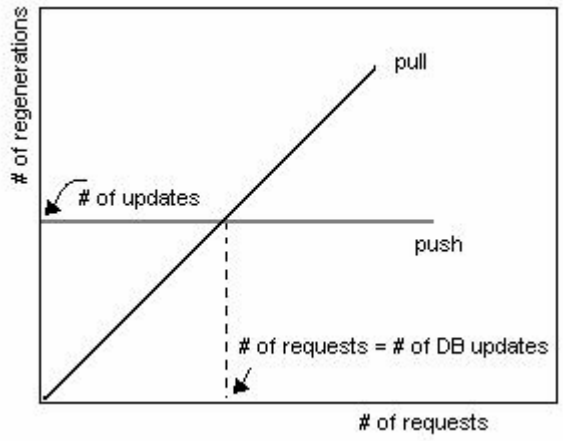
Αυτός ο κίνδυνος δεν υπάρχει (στην ίδια οξύτητα και με την ίδια πιθανότητα) στο προτεινόμενο σύστημα. Αυτό συμβαίνει διότι, αν θεωρήσουμε ότι κάθε αίτηση προς τον server περνά από μια cache που συμμετέχει στην push αρχιτεκτονική, τότε όλες οι αιτήσεις των clients εξυπηρετούνται από τον proxy και όχι από τον server. Στον server εναπόκειται η δημιουργία μιας νέας απάντησης για ένα πόρο και η δημιουργία της αντίστοιχης διαφοράς από την προηγούμενη έκδοση μόνο στην περίπτωση που συμβαίνει κάποια ενημέρωση στη βάση. Έτσι, οι κατασκευές διαφορών στον server είναι

ανεξάρτητες από τον αριθμό των αιτήσεων ανά πάσα στιγμή και επηρεάζονται μονάχα από τον αριθμό (συχνότητα) των ενημερώσεων της βάσης.

Ποιοτική σύγκριση του Pull και του Push μοντέλου

Σε γενικές γραμμές, η αρχιτεκτονική του push caching συμφέρει μόνο όταν οι αιτήσεις για τους εμπλεκόμενους πόρους είναι πιο πολλές από τις ενημερώσεις που γίνονται πάνω στα δεδομένα που αυτός ο πόρος παρουσιάζει. Αυτή βέβαια είναι μια απόλυτα λογική ιδιότητα που έχουν όλες οι εφαρμογές στις οποίες οποιοσδήποτε θα σκεφτόταν να εφαρμόσει τεχνικές caching: αν κάποιος πόρος αλλάζει πάρα πολύ συχνά, ενώ παράλληλα δεν είναι και πολύ δημοφιλής μεταξύ των χρηστών του διαδικτύου, δεν αποτελεί καλό υποψήφιο για να συμμετέχει σε διαδικασία προσωρινής αποθήκευσης (αυτό εξετάστηκε και σε προηγούμενο κεφάλαιο που αναλύθηκε ποιοι δυναμικοί πόροι είναι κατάλληλοι για caching).

Έτσι, λοιπόν, για εφαρμογές όπου οι ενημερώσεις των πόρων είναι πιο σπάνιες από τις αιτήσεις για αυτούς, ο server καλείται να φτιάξει λιγότερες φορές τις δυναμικές σελίδες με την προτεινόμενη αρχιτεκτονική αντί με την υπάρχουσα κατάσταση που δεν γίνεται καθόλου caching και οι δυναμικές σελίδες κατασκευάζονται τόσες φορές όσες και οι αιτήσεις για αυτές. Παρακάτω παρουσιάζεται ένα ποιοτικό διάγραμμα που αισθητοποιεί όλα τα παραπάνω:



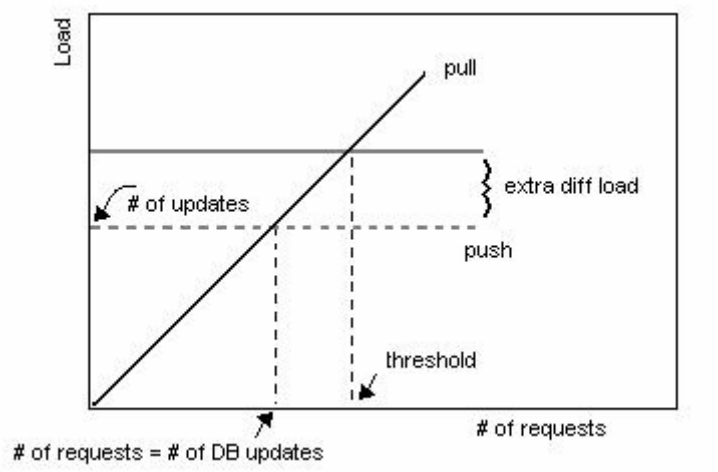
Στο παραδοσιακό μοντέλο των αιτήσεων και απαντήσεων του Web (που στην ουσία είναι pull μοντέλο αφού ο server στέλνει στον client κάποιο πόρο μόνο αν εκείνος εκκινήσει την επικοινωνία), όπως ισχύει σήμερα, ο αριθμός των δημιουργιών ενός δυναμικού πόρου (HTML regenerations) είναι ανάλογος του αριθμού των αιτήσεων που φθάνουν στον server για τον πόρο αυτό.

Αντίθετα, στο μοντέλο push, ο αριθμός των δημιουργιών δυναμικής σελίδας είναι πάντα σταθερός και ίσος με τον ρυθμό ενημέρωσης της βάσης. Βέβαια, και οι ενημερώσεις της βάσης δεν έρχονται πάντα με σταθερό ρυθμό, αλλά όμως ο μεταβλητός ρυθμός είναι ένα από τα στατιστικά της βάσης και μπορεί να προσεγγιστεί με τον μέσο ρυθμό ενημερώσεων.

Για αριθμό αιτήσεων μικρότερου του αριθμού ενημερώσεων είναι φανερό ότι δεν συμφέρει να χρησιμοποιήσουμε το μοντέλο του push caching. Ωστόσο,

για μεγαλύτερο αριθμό αιτήσεων το μοντέλο του push caching είναι σαφώς καλύτερο.

Κάτι που δεν φαίνεται ξεκάθαρα στο παραπάνω διάγραμμα είναι ότι πρόκειται για τον αριθμό των δημιουργιών δυναμικών σελίδων και όχι συνολικού φόρτου που θα πρέπει να σηκώσει ο server. Στην περίπτωση του pull μοντέλου, ο φόρτος καθορίζεται μόνο από τον αριθμό των δημιουργιών (και κατ' επέκταση των αιτήσεων), ενώ στο μοντέλο push προσαυξάνεται με το φορτίο που εισάγεται με την κατασκευή των διαφορών. Συμφωνα με αυτά, το αντίστοιχο διάγραμμα φόρτου θα είναι ποιοτικά όπως παρακάτω:



Αυτό σημαίνει ότι το σημείο από το οποίο συμφέρει η χρήση του μοντέλου push δεν είναι ακριβώς στο σημείο στο οποίο οι αιτήσεις είναι όσες και οι ενημερώσεις της βάσης, αλλά κατά τι μετατοπισμένο προς τα δεξιά (το threshold στο διάγραμμα). Για αριθμό αιτήσεων κάτω από αυτό το όριο συμφέρει μια αρχιτεκτονική pull, ενώ για περισσότερες συμφέρει το μοντέλο push. Μια αναλυτική ή εμπειρική εκτίμηση αυτού του ορίου και πως σχετίζεται με τις υπόλοιπες παραμέτρους του συστήματος θα ήταν κάτι πολύ ενδιαφέρον προς υπολογισμό.

Εκείνο που έχει ιδιαίτερη σημασία στα δύο παραπάνω ποιοτικά διαγράμματα είναι ότι το φορτίο που πρόκειται να αντιμετωπίσει ο server είναι φραγμένο, και μάλιστα μπορεί να εκτιμηθεί ξεχωριστά για κάθε εφαρμογή, ανάλογα με τη μεταβλητότητα των δεδομένων. Έτσι, όποιος σχεδιάζει το σύστημα του server και καλείται να επιλέξει το κατάλληλο hardware, μπορεί να κάνει την πιο συμφερουσα επιλογή που, χωρίς να υπερβαίνει κατά πολύ τις πραγματικές ανάγκες της εφαρμογής, να μπορεί να σηκώσει τον μέγιστο φόρτο που μπορεί να αντιμετωπίσει το σύστημα.

Χειρισμός έξαρσης ενημερώσεων στη Βάση Δεδομένων

Μάλιστα, αν οι ενημερώσεις τις βάσης είναι πάρα πολλές προκαλώντας έτσι μεγαλύτερο φόρτο, μπορεί χρησιμοποιηθεί κάποια τεχνική batching για τις ανακατασκεθές των επηρεαζόμενων HTML σελίδων. Αυτό μπορεί να γίνει αν η εφαρμογή επιτρέπει κάποιο ελάχιστο χρόνο στον οποίο να υπάρχει ασυνέπεια (inconsistency) μεταξύ server και cache. Για παράδειγμα, μπορεί να τεθεί μια παράμετρος στο σύστημα που να δηλώνει ότι οι ανακατασκευές

των σελίδων δεν μπορούν να γίνουν πιο συχνά από π.χ. 2 sec. Σε μια τέτοια περίπτωση, αν οι ενημερώσεις γίνονται πιο αραιά από αυτό το χρόνο τότε το σύστημα θα λειτουργεί κανονικά. Αντίθετα, αν γίνουν πολλές ενημερώσεις απότομα, αυτές θα ομαδοποιούνται και θα εκτελούνται όλες μαζί στο τέλος της περιόδου των 2 sec. Αυτό έχει το πλεονέκτημα ότι αν στο διάστημα των 2 δευτερολέπτων γίνουν πάνω από μία ενημερώσεις στη βάση που να επηρεάζουν την ίδια σελίδα, η δημιουργία της σελίδας θα γίνει μόνο μια φορά, εξοικονομώντας τις επιπλέον δυναμικές δημιουργίες της σελίδας που θα γινόντουσαν χωρίς την ομαδοποίηση αυτή. Το αρνητικό είναι ότι για το διάστημα των 2 δευτερολέπτων θα υπάρχει ασυνέπεια μεταξύ server και cache.

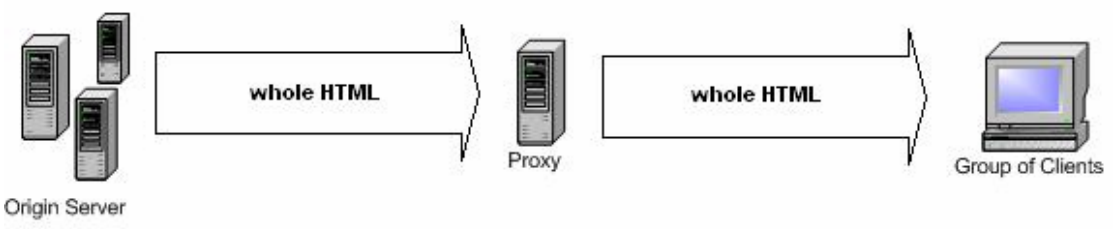
Το φορτίο των Proxies

Βέβαια, είναι σίγουρο ότι το βάρος μιας αύξησης αιτήσεων σε κάποια πιθανή ώρα αιχμής για την εφαρμογή μπορεί να μη φθάσει το server αλλά θα πρέπει κάποιο άλλο μηχάνημα να σηκώσει το φορτίο. Στο προτεινόμενο σύστημα αυτό το κομβικό σημείο είναι ο proxy, ο οποίος καλείται να εξυπηρετήσει όλες τις αιτήσεις των χρηστών. Το πλεονέκτημα όμως είναι ότι οι proxies μπορεί να είναι πολλοί και κάθε ένας να αναλαμβάνει την εξυπηρέτηση ενός μέρους μόνο των clients. Επίσης, ο φόρτος του proxy για την εξυπηρέτηση μιας αίτησης είναι πολύ μικρότερος από αυτόν του server, αφού απλά επιστρέφει στους χρήστες την σελίδα HTML που έχει αποθηκευμένη. Όπως και στον server, για την προτεινόμενη αρχιτεκτονική υπάρχει και στον proxy το επιπλέον κόστος για την ανακατασκευή των πόρων κάθε φορά που φθάνει κάποια ενημέρωση από τον server. Με τον ίδιο τρόπο όμως το φορτίο είναι και αυτό υπολογίσιμο και φραγμένο.

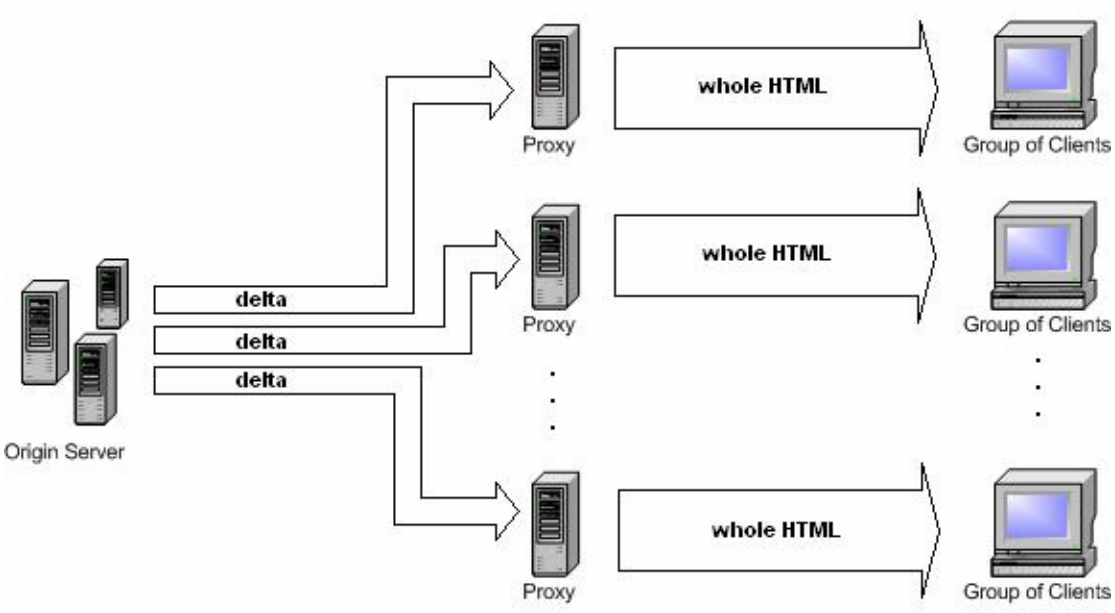
Τα κέρδη σε εύρος ζώνης

Πέρα από το θέμα του υπολογιστικού φόρτου που όντως είναι ένα σημείο σκεπτικισμού για ένα σύστημα delta encoding, το μεγάλο κέρδος που προκύπτει με ένα τέτοιο σύστημα είναι η τεράστια εξοικονόμηση σε εύρος ζώνης (bandwidth savings). Στη δοκιμαστική εφαρμογή, το κέρδος σε εύρος ζώνης ήταν πάνω από 90% (η συμπιεσμένη διαφορά είχε μέγεθος το 2%-9% του μεγέθους της κανονικής σελίδας). Η χρησιμότητα της εξοικονόμησης σε εύρος ζώνης είναι αυτονόητη, ωστόσο το παρακάτω παράδειγμα αρχιτεκτονικής δείχνει πόσο σημαντική είναι:

Έστω ένα σύστημα caching στο οποίο δεν χρησιμοποιείται delta encoding για εξοικονόμηση εύρους ζώνης, και έστω ότι ο server που φιλοξενεί την εφαρμογή έχει εύρος ζώνης R. Αν με αυτό το εύρος ζώνης μπορεί να στείλει την απάντηση σε μια αίτηση για μια HTML σελίδα σε ένα proxy, όπως στο σχήμα:



τότε με το delta encoding μπορεί να επιτευχθεί σημαντική κλιμάκωση του συστήματος. Αυτό μπορεί να γίνει διότι με τη χρήση ακριβώς του ιδίου εύρους ζώνης μπορούν να ενημερωθούν πολύ περισσότεροι proxies, όπως στο σχήμα:



Αν για παράδειγμα οι διαφορές είναι γύρω στο 20% ολόκληρης της σελίδας HTML (μια ιδιαίτερα μετριοπαθής εκτίμηση), τότε με το ίδιο bandwidth μπορούν να ενημερωθούν πέντε αντί του ενός proxies. Αυτό το γεγονός, σε συνδιασμό και με μια σωστή επιλογή της τοποθεσίας των proxies μπορεί να βοηθήσει ώστε να μπορούν να εξυπηρετηθούν 5 φορές περισσότεροι clients, χωρίς επιπρόσθετη επιβάρυνση σε bandwidth του server.

Συμπεράσματα

Τα τελικά συμπεράσματα από την μελέτη και υλοποίηση ενός συστήματος caching δυναμικού περιεχομένου που ενσωματώνει την τεχνική του Delta Encoding και ακολουθεί το μοντέλο του push caching είναι συνοπτικά τα παρακάτω:

- Λόγω του Delta Encoding επιτυγχάνονται αξιοσημείωτα κέρδη σε εύρος ζώνης, από τον server μέχρι τον proxy.
- Λόγω του μοντέλου push σχεδόν όλες οι αιτήσεις εξυπηρετούνται από τους proxies με αποτέλεσμα να αποφεύγεται σε μεγάλο βαθμό η κατάρρευση του server σε ώρες αιχμής.
- Επίσης λόγω του μοντέλου push και οι clients του δικτύου αντιλαμάνονται βελτίωση του χρόνου απόκρισης καθώς εξυπηρετούνται πάντα από τους proxies, που στη γενική περίπτωση βρίσκονται πιο κοντά τους.
- Στα θέματα προς εξέταση είναι οι επιπλέον απαιτήσεις ενός τέτοιου σχήματος σε αποθηκευτικό χώρο και υπολογιστική ισχύ. Ωστόσο, ο συνδιασμός των δύο τεχνικών (DE & Push Caching) δείχθηκε να μετριάζει ως ένα βαθμό τα προβλήματα αυτά, αφήνοντας ελπίδες και για περαιτέρω βελτιώσεις.

- Ανοιχτές κατευθύνσεις προς βελτίωση της απόδοσης του συστήματος είναι η επινόηση νέων αρχιτεκτονικών θα επεκτεινουν το προτεινόμενο σύστημα με ιδέες που κλιμακώνουν τη λειτουργία του στο επίπεδο της βάσης, ή του invalidation server ή του πράκτορα ανακατασκευής του proxy.
- Επιπλέον διερεύνηση χρειάζονται κάποια θέματα όπως η σχέση μεταξύ των μοντέλο pull & push και πότε συμφέρει η χρήση του κάθε ενός, καθώς επίσης και το φορτίο που εισάγει η εκτέλεση του diff και του patch σε server και proxy αντιστοιχα.

ΑΝΑΦΟΡΕΣ

[Attar et al. 2002] H. S. Attar, Y. Yang. Providing Strong Consistency in Dynamic Web Caching. December 2002.

[Banga et al. 1997] Gaurav Banga, Fred Douglis and Michael Rabinovich, Optimistic Deltas for WWW Latency Reduction, In proceedings of *USENIX Technical Conference*, '97.

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

[Brabrand et al. 2001] C. Brabrand, A. Mller, S. Olesen, and M. Schwartzbach. Language-Based Caching of Dynamically Generated HTML, 2001.

[Broder 1997] A. Broder. On resemblance and Containment of Documents. In *Proceedings of SEQUENCES-97*, 1997.

[CacheFlow] CacheFlow. Accelerating e-commerce with cacheflow internet caching appliances (a cacheflow white paper), October 1999.

[Cao et al. 1998] P. Cao, J. Zhang, and K. Beach. Active Cache: Caching Dynamic Contents on the Web. In Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98), pages 373-388, 1998.

[Candan et al. 2001] K. Candan, W. Li, Q. Luo, W. Hsiung, and D. Agrawal. Enabling Dynamic Content Caching for Database-Driven Web Sites. In Proceedings of ACM International Conference on Management of Data (SIGMOD), pages 532-543, 2001.

[Challenger et al. 1997] J. Challenger, and A. Iyengar. Improving Web Server Performance by Caching Dynamic Data. In Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, California, December 1997.

[Challenger et al. 1999] J. Challenger, P. Dantzig, and A. Iyengar. A Scalable System for Consistently Caching Dynamic Web Data. In Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies, New York, New York, 1999.

[Challenger et al. 2000] J. Challenger, A. Iyengar, K. Witting, C. Ferstat, and P. Reed. A Publishing System for Efficiently Creating Dynamic Web Content. In Proceedings of IEEE INFOCOM 2000, pages 844-853, 2000.

[Challenger et al. 2004] J. Challenger, P. Dantzig, and K. Witting. A fragment-based approach for efficiently creating dynamic web content. *ACM Transactions on Internet Technology*, 4(4), November 2004.

[Chen 1998] Hua Chen, etc. "HTTP Push: A New Method for Squid-Based Cache Engine," A&T Systems White Paper, June 1998.

[Chen et al. 1999] Hua Chen, Marc Abrams, Tommy Johnson, Anup Mathur, Ibraz Anwar, and John Stevenson, Wormhole caching with HTTP PUSH method for a satellite-based web content multicast and replication system, in Proceedings of the 4th International Web Caching Workshop, San Diego, CA, March 1999.

[Copeland and McClain] Copeland, G., McClain, M. Web Caching With Dynamic Content. IBM

[Datta et al. 2001a] A. Datta, K. Dutta, K. Ramamritham, H. Thomas, and D. VanderMeer. Dynamic content acceleration: A caching solution to enable scalable dynamic web page generation. In Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, May 2001.

[Datta et al. 2001b]A. Datta, K. Dutta, D. Fishman, K. Ramamritham, H. Thomas, and D. VanderMeer. A comparative study of alternative middle tier caching solutions. In Proceedings of the 2001 VLDB Conference, Rome, Italy, September 2001.

[Datta et al. 2002] A. Datta, K. Dutta, H. Thomas, D. VanderMeer, Suresha, and K. Ramamritham. Proxy-Based Acceleration of Dynamically Generated Content on the World Wide Web: An Approach and Implementation. In Proc. of ACM International Conference on Management of Data (SIGMOD), pages 97-108, 2002.

[Douglis et al. 1997] Douglis, F., Feldmann, A., Krishnamurthy, B., and Mogul, J. (1997a). Rate of change and other metrics: A live study of the World Wide Web. In Proceedings of the USENIX Symposium on Internet Technologies and Systems, pp. 147–158.

[Douglis et al. 1997b] F. Douglis, A. Haro, and M. Rabinovich. HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching. In USENIX Symposium on Internet Technologies and Systems, 1997.

[ESI Technical Specs] ESI - Accelerating E-Business Applications: Technical Specification. http://www.esi.org/, September 2002.

[ESI W3C submission] Edge Side Includes W3C submission. http://www.w3.org/Submission/2001/09/, September 2001.

[Feldmann et al. 1999] Feldmann, A., Cáceres, R., Douglis, F., Glass, G., and Rabinovich, M. (1999). Performance of Web proxy caching in heterogeneous bandwidth environments. In Proceedings of INFOCOM, pp. 107–116.

[FineGround] FineGround Networks. Breaking New Ground in Content Acceleration. http://www.fineground.com/pdf/FGCWhitepaper.pdf.

[Florescu et al. 1999] D. Florescu, A. Levy, D. Suciu, and K. Yagoub. Optimization of Run-time Management of Data Intensive Web Sites. In Proceedings, 25th International Conference on Very Large Data Bases, pages 627-638, Edinburgh, Scotland, September 1999.

[Gadde et al. 1997] S. Gadde, M. Rabinovich, and J. Chase. Reduce, reuse, recycle: An approach to building large internet caches. In Proceedings of the Workshop on Hot Topics in Operating Systems, May 1997.

[Gauthier et al. 1998] P.Gauthier, J.Cohen, M.Dunsmuir, and C.Perkins. The Web proxy auto discovery protocol.
http://www.ietf.org/internet-drafts/draft-ietf-wrec-wpad-00.txt, 1998.

[Gray and Cheriton 1989] Gray, C. G., and Cheriton, D. R. (1989). Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In Proceedings of the 12th ACM Symposium on Operating System Principles, pp. 202–210.

[Gwetzman and Seltzer 1996] J. Gwetzman and M. Seltzer, World Wide Web cache consistency, Proceedings of the USENIX Technical Conference, pp. 141-152, January 1996.

[Holmedahl et al. 1998] V. Holmedahl, B. Smith, and T. Yang. Cooperative Caching of Dynamic Content on a Distributed Web Server. Technical Report TRCS98-12, 14, 1998.

[Housel and Lindquist 1996] Barron Housel and David Lindquist, WebExpress: A System for Optimizing Web Browsing in a Wireless Environment, In proceedings of the *ACM/IEEE 2nd annual international conference on Mobile computing and networking, MOBICOM*, November '96.

[HTTP 1.1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. Hypertext Transfer Protocol -- HTTP/1.1, June 1999.

[InktomiCorp.] Inktomi network products.
http://www.inktomi.com/products/network/.

[Ionescu 2000] Mihut D. Ionescu. xProxy: A transparent caching and delta transfer system for web objects. Master's thesis, University of California at Berkeley, December 2000.

[Jupiter 2001] Jupiter internet access model (US only). Broadband.
Jupiter Media Metrix, August 2001.

[Krishnamurthy and Rexford 2001] Krishnamurthy, B., and Rexford, J. (2001). Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement. Addison-Wesley, Boston.

[Labrinidis-Roussopoulos 2000] Alexandros Labrinidis and Nick Roussopoulos. WebView Materialization. ACM SIGMOD International Conference on Management of Data, Dallas, Texas, 2000.

[Li et al. 2001] D. Li, P. Cao, M. Dahlin. WCIP: Web Cache Invalidation Protocol, March 2001. Work In Progress. http://www.watersprings.org/pub/id/draft-danli-wrec-wcip-01.txt.

[Luo et al. 2000] Q. Luo, J. Naughton, R. Krishnamurthy, P. Cao, and Y. Li. Active query caching for database web servers. In Proceedings of WebDB 2000, 2000.

[Luo et al. 2001] Q. Luo and J. Naughton. Form-Based Proxy Caching for Database-Backed Web Sites. In The VLDB Journal, pages 191-200, 2001.

[Luo et al. 2002] Q. Luo, S. Krishnamurthy, C. Mohan, H. Pirahesh, H. Woo, B. Lindsay, and J. Naughton. Middle-tier Database Caching for e-Business. In Proceedings of ACM International Conference on Management of Data SIGMOD, pages 600-611, 2002.

[MicrosoftISA] Microsoft ISA Server. http://www.microsoft.com/isaserver

[Mogul et al. 1997] Jeffrey Mogul, Fred Douglis, Anja Feldmann and Balachander Krishnamurthy, Potential benefit of delta encoding and data compression for HTTP, In proceedings of *ACM SIGCOMM*, '97.

[Mogul et al. 2001] Jeffrey Mogul, Balachander Krishnamurthy, Fred Douglis, Anja Feldmann, Yaron Goland, Arthur Van Hoff, and Daniel Hellerstein. Delta Encoding in HTTP, March 2001. Work In Progress. http://www.ietf.org/internet-drafts/draft-mogul-http-delta-08.txt.

[Naaman 2002] Mor Naaman, Hector Garcia-Molina, and Andreas Paepcke. Evaluation of Delivery Techniques for Dynamic Web Content (Extended Version). Technical Report Number 2002-31. Stanford University, 2002. Available at http://dbpubs.stanford.edu/pub/2002-31.

[Naaman et al. 2003] M. Naaman, H. Garcia-Molina, and A. Paepcke. Evaluation of ESI and Class-Based Delta Encoding. In *Proceedings of WCW - 2003*.

[NetworkAppliance] Network Appliance. http://www.netapp.com

[Network Appliance NetCache4.0] Network Appliance Inc. http://www.netapp.com/products/netcache/.

[OracleWebCache 2002] J. Anton, L. Jacobs, X. Liu, J. Parker, Z. Zeng, and T. Zhong. Web Caching for Database Applications with Oracle Web Cache. In Proceedings of ACM International Conference on Management of Data (SIGMOD), pages 594-599, 2002.

[Psounis 2002] Konstantinos Psounis. Class-based delta-encoding: a scalable scheme for caching dynamic Web content. 22nd International Conference on Distributed Computing Systems Workshops, 2-5 July 2002, Vienna, Austria.

[Rabinovich et al. 1999] M. Rabinovich and A. Aggarwal. Radar: A scalable architecture for a global web hosting service. WWW8 / Computer Networks, 31(11-16):1545-1561, 1999.

[Rabinovich et al. 2003] Rabinovich, M., Xiao, Z., Douglis, F., and Kalmanek, C. R. 2003. Moving Edge-Side Includes to the Real Edge—the Clients. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*.

[Ramaswamy et al. 2004] Lakshmish Ramaswamy , Arun Iyengar , Ling Liu , Fred Douglis, Automatic detection of fragments in dynamically generated web pages, Proceedings of the 13th conference on World Wide Web, May 17-20, 2004, New York, NY, USA.

[Rhea et al. 2003] S. C. Rhea, K. Liang, and E. Brewer. Value-Based Web Caching. In Proceedings of 12th WWW Conference, 2003.

[RHK 2002] Network traffic& revenue analysis. market update. RHK, Inc., July 2002.

[Squid Push Patch] Squid Cache Push Patch: Merging push-cache functionality - 'push' support and hint caching - into Squid-HEAD. http://devel.squid-cache.org/stale_projects.html#push

[TenTimesTen 2000] The Times Ten Team. High Performance and Scalability through Application-Tier In-Memory Data Management. In Proceedings of International Conference on Very Large Data Bases (VLDB), pages 677-680, 2000.

[Wessels 1998] Duane Wessels, etc. "ICP and the Squid Web Cache," IEEE J. Selected Areas in Communications, V16, No.3, April 1998.

[Williams et al. 1996] Williams, S., Abrams, M., Standridge, C.R., Abdulla, G., and Fox, E.A. (1996). Removal policies in network caches for World-Wide Web documents. In Proceedings of the ACM SIGCOMM Conference, pp. 293–305.

[Wolman et al. 1999a] Wolman, A., Voelker, G., Sharma, N., Cardwell, N., Brown, M., Landray, T., Pinnel, D., Karlin, A., and Levy, H. (1999). Organization-based analysis of Web-object sharing and caching. In Proceedings of the USENIX Symposium on Internet Technologies and Systems, pp. 25–36.

[Yagoub et al. 2000] K. Yagoub, D. Florescu, V. Issarny, and P. Valduriez. Caching strategies for data-intensive web sites. In VLDB 2000, Cairo, Egypt, pages 188-199, 2000.

[Zhu and Yang 2001] Huican Zhu and Tao Yang, “Class-based Cache Management for Dynamic Web Content”, In proceedings of IEEE INFOCOM, ’01.