

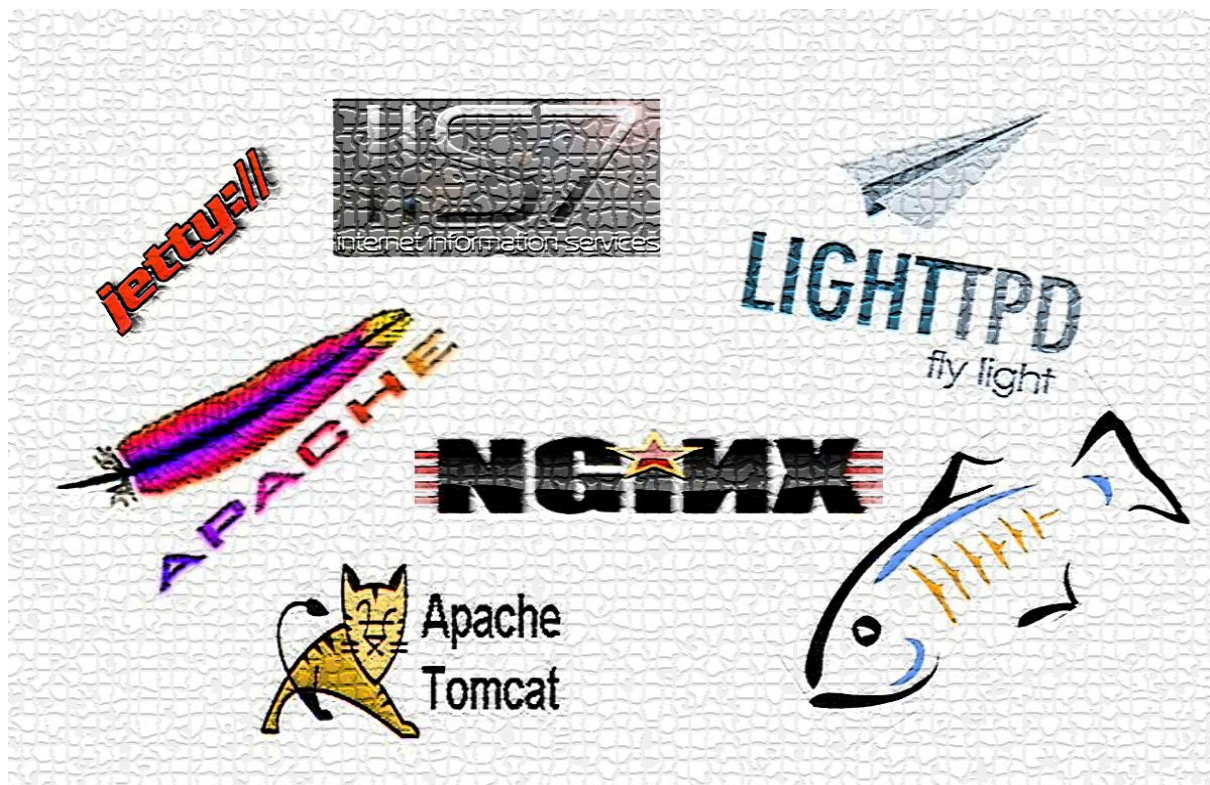


ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



Πτυχιακή εργασία

«Μέτρηση απόδοσης Web Server»



Του φοιτητή
Καπανιάρη Ευστάθιου
Αρ. Μητρώου: 06/3074

Επιβλέπων καθηγητής
Σιδηρόπουλος Αντώνιος

Περιεχόμενα

Πρόλογος.....	1
Κεφάλαιο 1 - Εισαγωγή.....	2
1.1 Τι είναι ο web server	2
1.2 Μέτρηση απόδοσης ενός web Server	3
1.2.1 Γιατί μετράμε την απόδοση;	3
1.2.2 Προετοιμασία για την μέτρηση. Τι προσέχουμε.....	3
1.2.3 Τι μετράμε.....	4
1.2.4 Πως μετράμε	4
Κεφάλαιο 2 - Μετρήσεις	6
2.1 Ταυτότητα μέτρησης.....	6
2.2 Εργαλεία μέτρησης	9
Κεφάλαιο 3 - Apache Web Server	11
3.1 Ιστορικά στοιχεία	11
3.2 Αποτελέσματα μετρήσεων	12
3.2.1 Στατικές ιστοσελίδες	12
3.2.2 Δυναμικές ιστοσελίδες	16
3.3 Συμπεράσματα.....	26
Κεφάλαιο 4 - IIS (Internet Information Services).....	28
4.1 Ιστορικά στοιχεία	28
4.2 Αποτελέσματα μετρήσεων	29
4.2.1 Στατικές ιστοσελίδες	29
4.2.2 Δυναμικές ιστοσελίδες	30
4.3 Συμπεράσματα.....	35
Κεφάλαιο 5 - Nginx	36
5.1 Ιστορικά στοιχεία	36
5.2 Αποτελέσματα μετρήσεων	37

5.2.1 Στατικές ιστοσελίδες	37
5.2.2 Δυναμικές ιστοσελίδες	38
5.3 Συμπεράσματα.....	44
Κεφάλαιο 6 - Lighttpd	45
6.1 Ιστορικά στοιχεία	45
6.2 Αποτελέσματα μετρήσεων	46
6.2.1 Στατικές ιστοσελίδες	46
6.2.2 Δυναμικές ιστοσελίδες	49
6.3 Συμπεράσματα.....	57
Κεφάλαιο 7 - Συγκρίσεις - Ανάλυση αποτελεσμάτων – Συμπεράσματα	58
7.1 Εργαλεία μέτρησης	58
7.2 Συγκρίσεις – Ανάλυση αποτελεσμάτων - Συμπεράσματα	60
Κεφάλαιο 8 - Java Servlets – Αρχεία JSP	88
8.1 Εισαγωγή στα Servlets.....	88
8.2 Ιστορικά στοιχεία	89
8.3 Αποτελέσματα μετρήσεων	90
8.4 Συγκρίσεις – Ανάλυση αποτελεσμάτων - Συμπεράσματα	98
Επίλογος.....	107
Παράρτημα.....	108
Βιβλιογραφία.....	127

Πρόλογος

Το 1990, με την δημιουργία του World Wide Web (WWW), το διαδίκτυο έκανε το μεγαλύτερο του βήμα για την γνωριμία του με το ευρύ κοινό, μετατρέποντάς το από ένα κλειστό δίκτυο επικοινωνίας υπολογιστών σε ένα παγκόσμιο φαινόμενο. Σήμερα, το σύνολο σχεδόν του παγκόσμιου πληθυσμού χρησιμοποιεί τις υπηρεσίες του διαδικτύου για εργασία, ενημέρωση, ψυχαγωγία, ενώνοντας εκατομμύρια ανθρώπους από διαφορετικά μέρη του κόσμου. Ένα από τα στοιχεία του WWW που έκαναν εφικτή αυτή την προσπάθεια είναι οι Web Servers, οι οποίοι έχουν τη δική τους ιστορία σε αυτή τη διαδρομή. Σκοπός αυτής της πτυχιακής εργασίας είναι η προσπάθεια σύγκρισης των δημοφιλέστερων αυτών προγραμμάτων, παρέχοντας μία σφαιρική άποψη γύρω από τη λειτουργία τους και την αποδοτικότητα τους.

Κεφάλαιο 1 - Εισαγωγή

1.1 Τι είναι ο web server

Ένας web server αποτελεί ένα ηλεκτρονικό υπολογιστή (υλικό και λογισμικό), ο οποίος χρησιμοποιείται για την μεταφορά δεδομένων μέσω του Διαδικτύου. Χρησιμοποιείται κατά κύριο λόγο για την αποθήκευση και παροχή πρόσβασης σε ιστοσελίδες. Επίσης μπορεί να χρησιμοποιηθεί για την αποθήκευση και μεταφορά αρχείων. Με την ανάπτυξη των ευζωνικών συνδέσεων, έγινε δυνατή και η εκτέλεση εφαρμογών σε έναν web server μετά από αίτηση του χρήστη. Όλα αυτά με την προϋπόθεση ότι ο web server είναι συνδεδεμένος στο διαδίκτυο.

Οι αιτήσεις σε ένα web server γίνονται με την χρήση πρωτοκόλλων (Ανάλογα με το αίτημα). Το πιο συχνό πρωτόκολλο που χρησιμοποιείται είναι το HTTP (**H**yper**T**ext **T**ransfer **P**rotocol), το οποίο επεξεργάζεται αιτήματα εμφάνισης ιστοσελίδων. Άλλα διαδεδομένα πρωτόκολλα είναι το IMAP(**I**nternet **M**essage **A**ccess **P**rotocol) για την μεταφορά δεδομένων e-mail και τα FTP(**F**ile **T**ransfer **P**rotocol) και SMB(**S**erver **M**essage **B**lock) για μεταφορά αρχείων παντός τύπου.

Γνωστό λογισμικό web servers

Για την λειτουργία ενός web server απαιτείται και το ανάλογο λογισμικό το οποίο θα διαχειρίζεται τα διάφορα αιτήματα. Τα γνωστά λογισμικά που υπάρχουν σήμερα είναι τα εξής:

- Apache της εταιρίας Apache Software Foundation
 - Apache Tomcat : Ανοιχτού κώδικα web server αποκλειστικά για Java servlets και JavaServer Pages.
- ISS της Microsoft
- Nginx του Igor Sysoev
- GlassFish της Sun Microsystems
- Lighttpd της Lighttpd Developers

Την μερίδα του λέοντος κατέχει ο Apache με το 59,36% των sites και ακολουθούν με 22,70% ο ISS, με 6,04% ο Nginx, με 0,94% ο GlassFish και με 0,83% ο Lighttpd.

1.2 Μέτρηση απόδοσης ενός web Server

1.2.1 Γιατί μετράμε την απόδοση;

Με την μέτρηση ενός web server ο σκοπός μας είναι ένας: να δούμε μέσω συγκριτικών δοκιμών εάν ο server μπορεί να διαχειριστεί με επιτυχία τα διαφορετικά αιτήματα, ιδιαίτερα σε περιπτώσεις όπου ο αριθμός των ταυτόχρονων αιτημάτων είναι πάρα πολύ μεγάλος.

Η μέτρηση του δεν είναι μία απλή, εύκολη διαδικασία. Οι μετρήσεις έχουν ένα συγκεκριμένο χαρακτήρα. Δεν έχει νόημα να μετρήσουμε ένα web server την στιγμή που απλώς θα στείλει μία σελίδα στον client. Δεν μας ενδιαφέρει εάν ο χρήστης θα έχει την σελίδα που επιθυμεί σε 1 ή 2 ms. Μας ενδιαφέρει ο μέσος χρόνος που χρειάζεται όταν ο μέγιστος αριθμός χρηστών επιθυμεί την εμφάνιση της ίδιας ιστοσελίδας ταυτόχρονα. Παράλληλα, μας ενδιαφέρει ακόμα περισσότερο πως θα αποδώσει ο server, όταν ο αριθμός των αιτήσεων είναι διπλάσιος ή και παραπάνω από την αρχική μέτρηση.

1.2.2 Προετοιμασία για την μέτρηση. Τι προσέχουμε.

Για να πραγματοποιήσουμε μετρήσεις σε διαφορετικά λογισμικά πρέπει να λάβουμε υπόψη μερικούς, τεχνικής φύσεως παράγοντες. Αρχικά, είναι απαραίτητο να χρησιμοποιήσουμε το ίδιο hardware, με τις ίδιες ρυθμίσεις και το ίδιο λογισμικό για όλες τις μετρήσεις. Φυσικά αυτό ισχύει και για τις ρυθμίσεις δικτύου. Χρησιμοποιούμε δηλαδή κάρτες δικτύου μόνο ενός τύπου π.χ 100Mbps Ethernet ή 1Gbps (Gigabit Ethernet).

Το επόμενο βήμα είναι να πραγματοποιήσουμε 4 με 5 load test. Με αυτά βλέπουμε την ταχύτητα απόκρισης του server, στέλνοντας ένα απλό αίτημα και περιμένοντας την απάντησή του. Η διαδικασία αυτή γίνεται 4 με 5 φορές ώστε να καταγραφεί το καλύτερο αποτέλεσμα, το οποίο και κρατάμε (καλύτερο αποτέλεσμα θεωρείται αυτό με την μικρότερη τιμή). Πριν από κάθε νέα μέτρηση κάνουμε επανεκκίνηση του server, ώστε να καθαρίσει από την προηγούμενη, που ενδεχομένως να επέφερε αλλαγές στις ρυθμίσεις. Φυσικά ξανατρέχουμε νέα load tests ώστε να καταγράψουμε νέα αποτελέσματα.

Για τις μετρήσεις χρησιμοποιούμε τόσο δυναμικές όσο και στατικές ιστοσελίδες.

1.2.3 Τι μετράμε

Στην μέτρηση λαμβάνουμε υπόψη τα εξής πράγματα:

- Αριθμός αιτημάτων ανά δευτερόλεπτο
 - Αφορά τον αριθμό των αιτημάτων που μπορεί να διαχειριστεί ένας server μέσα σε ένα δευτερόλεπτο. Η εν λόγω μέτρηση μπορεί να γίνει είτε μέσω μίας σύνδεσης, η οποία στέλνει όλα τα αιτήματα, είτε μέσω πολλών συνδέσεων.
- Ταυτόχρονες συνδέσεις
 - Σημαντική μέτρηση που δείχνει πόσες ταυτόχρονες συνδέσεις μπορεί να διαχειριστεί ένας server χωρίς προβλήματα
- Ταχύτητα μεταφοράς δεδομένων
 - Ο αριθμός των byte που μεταφέρονται ανά δευτερόλεπτο. Η μέτρηση αυτή έχει διάφορες πλευρές, καθώς είναι διαφορετική η μεταφορά ενός (μεγάλου) αρχείου από την μεταφορά πολλών (μικρότερων), λόγω των διαφορετικών ταυτόχρονων συνδέσεων
- Ο αριθμός των «σωστών συνδέσεων»
 - Πρόκειται για μέτρηση που αφορά συγκριμένους τύπους δοκιμών. Ελέγχει πόσες ταυτόχρονες συνδέσεις μπορεί να υποστηρίξει ένας server, ώστε αυτές να αποδίδουν ταχύτητα μεταφοράς δεδομένων τουλάχιστον 320.000 bits ανά δευτερόλεπτο.

1.2.4 Πως μετράμε

Η διαδικασία μέτρησης παλιότερα ήταν μία πολύ πολύπλοκη διαδικασία καθώς όποιος ήθελε να μετρήσει, έπρεπε να φτιάξει τα δικά του εργαλεία. Σήμερα αυτό έχει αλλάξει σημαντικά καθώς πλέον υπάρχουν εργαλεία τα οποία αυτοματοποιούν σε μεγάλο βαθμό τις διαδικασίες. Τα δημοφιλέστερα και πιο έγκυρα εργαλεία είναι τα παρακάτω:

- ApacheBench: Εργαλείο μέτρησης από τον Apache Foundation (Περιλαμβάνεται στο πακέτο εγκατάστασης του Apache Web Server). Χρησιμοποιεί γραμμή εντολών.
- JMeter: Ανοιχτού κώδικα λογισμικό γραμμένο σε Java. Δημιουργεί σενάρια για διαφορετικούς τύπους πρωτοκόλλων
- Pyload: Ένα καινούργιο εργαλείο μέτρησης, γραμμένο σε γλώσσα Python

Μέτρηση απόδοσης Web Server

- OpenSta: Δημιουργεί scripts σε γλώσσα προγραμματισμού SCL και εφαρμόζει εξαντλητικά load tests στα πρωτόκολλα HTTP και HTTPS
- HTTP Test Tool: Δημιουργία scripts και δοκιμές στο πρωτόκολλο HTTP
- Httpperf: Εργαλείο που πραγματοποιεί απλές δοκιμές μέχρι δοκιμές πολλαπλών συνδέσεων
- Autobench: Γραμμένο σε γλώσσα προγραμματισμού Perl, αυτοματοποιεί τα τεστ του εργαλείου httpperf. Παρέχει δυνατότητα εξαγωγής αποτελεσμάτων, ώστε αυτά να μπορούν εισαχθούν σε άλλο πρόγραμμα για περαιτέρω ανάλυση.

Κεφάλαιο 2 - Μετρήσεις

2.1 Ταυτότητα μέτρησης

Οι μετρήσεις που έγιναν είναι οι ίδιες για όλους τους web server, με τα ίδια εργαλεία και με τα ίδια αρχεία. Επίσης, όλοι οι web server εξετάστηκαν σε δύο διαφορετικά λειτουργικά συστήματα στη πλευρά του server (Ubuntu Server 10.10, Windows Server 2008). Ο server ως υπολογιστής είχε τα εξής χαρακτηριστικά

Πίνακας 1. Χαρακτηριστικά Server

Επεξεργαστής	AMD Athlon 64 3500+ 1 Ghz
Μνήμη RAM	1 gb
Μητρική Κάρτα	MSI K8N Neo Platinum
Σκληρός δίσκος	Για τις ανάγκες της μέτρησης χρησιμοποιήθηκε ένας δίσκος χωρητικότητας 1 TB , χωρισμένος σε διαμερίσματα των 25 GB

Οι μετρήσεις έγιναν με τη χρήση ενός σταθερού υπολογιστή που είχε το ρόλο του client (Επεξεργαστής AMD Phenom x4 425, Μνήμη 4gb). Η σύνδεση μεταξύ τους έγινε με τη χρήση καλωδίων utp, τα οποία ξεκινούσαν από τις Ethernet θύρες κάθε υπολογιστή και κατέληγαν σε ένα Fast Ethernet Switch.

Τα εργαλεία που χρησιμοποιήθηκαν είναι τα παρακάτω:

- Ab tool
- Jmeter
- Pylot

Οι μέθοδοι που ακολουθήθηκαν είναι οι εξής:

- Αποστολή πολλαπλών Request με παράλληλες συνδέσεις χρηστών, όσες μπορούσε ο web server να αντέξει μέχρι να κλείσει η σύνδεση.
- Μετρήσεις με το Pylot με βάση το χρόνο (60 και 120 δευτερόλεπτα).

Οι web servers που εξετάστηκαν είναι οι :

- Apache 2.2
- IIS 7
- Nginx 1.0.4
- Lighttpd 1.4.28

Οι παραπάνω web server παρουσιάζουν τις ιδιομορφίες τους, πράγμα που σημαίνει ότι **δεν είναι εφικτό να τρέξουμε όλα τα είδη αρχείων σε όλους τους web server**. Το ίδιο ισχύει και στην περίπτωση που ο ίδιος web server χρησιμοποιείται σε διαφορετικό λειτουργικό σύστημα. Αναλυτικότερα:

- Apache web server
 - Τρέχει και στα δύο λειτουργικά συστήματα Ubuntu Server και Windows Server 2008
 - Σε Ubuntu Server μπορεί να τρέξει τα παρακάτω αρχεία
 - Html , PHP, Asp.Net, Perl
 - Σε Windows Server 2008 μπορεί να τρέξει τα παρακάτω
 - Html, PHP, Perl
- IIS web server
 - Σε Windows Server 2008 μπορεί να τρέξει τα παρακάτω
 - Html, PHP, Asp.Net
 - Δεν υπάρχει διαθέσιμη έκδοση για Ubuntu Server (και Linux γενικά)
- nginx web server
 - Τρέχει και στα δύο λειτουργικά συστήματα Ubuntu Server και Windows Server 2008
 - Σε Ubuntu Server μπορεί να τρέξει τα παρακάτω αρχεία
 - Html , PHP, Asp.Net, Perl
 - Σε Windows Server 2008 μπορεί να τρέξει τα παρακάτω
 - Html, PHP
- Lighttpd web server
 - Τρέχει και στα δύο λειτουργικά συστήματα Ubuntu Server και Windows Server 2008

- Σε Ubuntu Server μπορεί να τρέξει τα παρακάτω αρχεία
 - Html , PHP, Asp.Net, Perl
- Σε Windows Server 2008 μπορεί να τρέξει τα παρακάτω
 - Html, PHP, Perl

Οι διαδικασίες για να τρέξουν οι web server τα διάφορα αρχεία δεν είναι απλή διαδικασία. Για παράδειγμα για να λειτουργήσει η Asp.Net εκτός του IIS (αποτελούν και τα δύο υλοποίηση της Microsoft), απαιτείται η χρήση του mono project, το οποίο ουσιαστικά είναι ανεξάρτητη υλοποίηση. Σε περιβάλλον Ubuntu Server μπορεί να λειτουργήσει έπειτα από αρκετή παραμετροποίηση και χωρίς δέσμευση επιτυχίας. Αυτό ίσως οφείλεται (ίσως) στο γεγονός ότι το mono project έχει μείνει πίσω σχετικά με τις νεότερες εκδόσεις των web server, καθώς οι οδηγίες που παρέχονται μέσα από την επίσημη σελίδα του project παρουσιάζουν διάφορες ανομοιότητες με την πραγματικότητα (χαρακτηριστικό παράδειγμα ότι αναφέρονται συχνά σε διαδρομές αρχείων που δεν υφίστανται!). Επίσης σε περιβάλλον Windows δεν ήταν δυνατό να λειτουργήσει κάτω από οποιοδήποτε web server (εξαιρέση φυσικά αποτελεί ο IIS ο οποίος δεν το χρειάζεται).

Όσων αφορά τα αρχεία jsp, για αυτά ισχύει ότι χρειάζονται συγκεκριμένο server για να τρέξουν. Αν παραστεί ανάγκη να τρέχουν μαζί με άλλα αρχεία (πχ PHP), οι υπόλοιποι web server παρέχουν δυνατότητα χρησιμοποίησης ως proxy server των server αυτών. Οι server, οι οποίοι ονομάζονται web containers, που εξετάζονται είναι οι εξής

- Apache Tomcat 6
- GlassFish 3.1
- Jetty 8

Τα αρχεία που χρησιμοποιήθηκαν στις μετρήσεις είναι τα παρακάτω

- HTML
 - Simpletest.html – Απλό αρχείο Html που περιέχει μία γραμμή κειμένου.
 - Hardtest.htm – Σελίδα Html που περιέχει σχεδόν όλα τα στοιχεία που μπορεί να διαθέτει μία σελίδα (πίνακες, flash, εικόνες κτλ).

- PHP
 - Simpletest.php – Αρχείο το οποίο κάνει κλήση στη βάση δεδομένων MySQL και εμφανίζει τα περιεχόμενα ενός πίνακα.
 - Mediumtest.php – Με την κλήση δημιουργεί δυναμικά μία εικόνα.
 - Hardtest.php – Πρόκειται για αρχείο στο οποίο γίνεται ανάγνωση ενός αρχείου κειμένου txt που περιέχει ονόματα, επίθετα και ένα συνθηματικό. Στη συνέχεια αυτά τα στοιχεία τοποθετούνται σε ένα πίνακα, ταξινομούνται με βάση το όνομα του κάθε προσώπου και εμφανίζονται σε έναν πίνακα HTML στον browser του χρήστη.
- Asp.Net
 - Simpletest.aspx – Εμφανίζει ένα ημερολόγιο μέσω κλήσης μίας συνάρτησης της Asp.Net
 - Hardtest.aspx – Κάνει ότι ακριβώς και το αρχείο Hardtest.php, γραμμένο στην αντίστοιχη γλώσσα.
- Perl
 - Simpletest.pl – Με την κλήση πραγματοποιεί μία For και εμφανίζει τα νούμερα από το 1 μέχρι το 10
 - Hardtest.pl - Κάνει ότι ακριβώς και τα αρχεία Hardtest.php και Hardtest.aspx, γραμμένο στην αντίστοιχη γλώσσα.
- JSP – JavaServer Pages
 - Simpletest.jsp – Με την κλήση του αρχείου εμφανίζεται στο χρήστη η τρέχουσα ώρα.
 - Hardtest.jsp - Κάνει ότι ακριβώς και τα αρχεία Hardtest.php και Hardtest.aspx, γραμμένο στην αντίστοιχη γλώσσα.

2.2 Εργαλεία μέτρησης

- Apache Bench (AB Tool)

Το AB Tool πρόκειται για ένα εργαλείο μέτρησης του Apache Foundation. Δεν χρησιμοποιεί γραφικό περιβάλλον, αλλά όλες οι εντολές δίνονται μέσω του command line. Στις πρώτες εκδόσεις του ήταν συμβατό μόνο με την μέτρηση ενός Apache Web Server. Στις επόμενες επανασχεδιάστηκε ώστε να είναι συμβατό με οποιοδήποτε web server. Η λογική που ακολουθεί είναι η αποστολή μαζικών

requests προς τον web server και μετράει την ταχύτητα απόκρισής του. Περιλαμβάνεται στην εγκατάσταση του Apache web server, και πρόκειται για ένα δωρεάν, ανοιχτού κώδικα πρόγραμμα.

- JMeter

Το Apache Jmeter πρόκειται και αυτό για ένα πρόγραμμα μέτρησης της εταιρίας Apache Foundation. Είναι και αυτό ανοιχτού κώδικα, ενώ είναι γραμμένο εξ ολοκλήρου σε Java. Χρησιμοποιεί γραφικό περιβάλλον για τις μετρήσεις, παρέχοντας παράλληλα διάφορα εργαλεία αποθήκευσης αποτελεσμάτων μέτρησης όπως αποθήκευση των πινάκων με τα αποτελέσματα και παραγωγή γραφημάτων. Στα μειονεκτήματά του συγκαταλέγονται τα γεγονότα ότι παρουσιάζει σημάδια αστάθειας ως προς τη λειτουργία του σαν πρόγραμμα (όχι στις μετρήσεις του), καθώς σε μετρήσεις με πολλά requests «κατέρρευε». Επίσης τα γραφήματά του δεν είναι τόσο σαφή, μην παρέχοντας και δυνατότητα σημαντικής τροποποίησής τους.

- Pylot

Το Pylot πρόκειται για σχετικά καινούργιο εργαλείο το οποίο σχεδιάστηκε από τον Corey Goldberg. Είναι γραμμένο σε γλώσσα Python, ενώ δίνει τη δυνατότητα μέτρησης είτε μέσω κονσόλας, είτε μέσω γραφικού περιβάλλοντος. Διαφέρει με τα δύο προηγούμενα εργαλεία στο ότι τις ρυθμίσεις των μετρήσεων (Διεύθυνση IP η URL, μέθοδος GET ή POST κτλ) τις δέχεται μέσω αρχείων XML, το οποίο διευκρινίζουμε στο πρόγραμμα όταν ξεκινάμε τις μετρήσεις μας. Επίσης πραγματοποιεί καταγραφή των αποτελεσμάτων σε αρχείο csv, παράγει διαγράμματα και τα παρουσιάζει ομαδοποιημένα σε ένα φιλικό προς το χρήστη html αρχείο.

Κεφάλαιο 3 - Apache Web Server

3.1 Ιστορικά στοιχεία

Ο Apache Web Server αποτελεί δημιούργημα της ομάδας του Apache Software Foundation. Η ανάπτυξη του ξεκίνησε στις αρχές του 1995 και βασίστηκε πάνω στο δημοφιλέστερο πρόγραμμα web server της εποχής, το HTTP daemon, κατασκευασμένο από τον Rob McCool του τμήματος Ανάπτυξης λογισμικού υπολογιστών στο Πανεπιστήμιο του Ιλινόις. Το σχέδιο όμως εγκαταλείφθηκε με την αποχώρηση του McCool από το πανεπιστήμιο στα μέσα του 1994. Τότε, ανεξάρτητοι προγραμματιστές άρχισαν να δουλεύουν πάνω στην επέκταση και διόρθωση του web server. Μία ομάδα από αυτούς, άρχισε την επικοινωνία μέσω email ανταλλάσσοντας ιδέες και απόψεις πάνω στην καλύτερη ανάπτυξη του λογισμικού. Αυτό οδήγησε στην δημιουργία του Apache Group με κύριους εκπροσώπους τους Brian Behlendorf και Cliff Skolnick.

Η πρώτη έκδοσή του χρονολογεί τον Απρίλιο του 1995 με την έκδοση 0.6.2. Ο νέος server κέντρισε αμέσως το ενδιαφέρον του πανεπιστημίου και η παλιά ομάδα εργασίας του HTTP daemon άρχισε να δουλεύει πάνω στην περαιτέρω ανάπτυξή του. Το αποτέλεσμα ήταν μέσα σε ένα χρόνο, και με την κυκλοφορία της έκδοσης 1.0, ο Apache είχε γίνει ο δημοφιλέστερος web server του διαδικτύου. Το 1999, η ομάδα του Apache group βλέποντας τη ραγδαία άνοδο του, δημιούργησε τον Apache Software Foundation ώστε να υπάρχει οργανωτική, νομική και οικονομική κάλυψη του όλου εγχειρήματος. Κάτω από την ομπρέλα του ιδρύματος γίνεται συνεχής αναβάθμισή του, στην οποία μπορούν να συμμετέχουν προγραμματιστές από όλο τον κόσμο καθώς πρόκειται για ανοιχτού κώδικα λογισμικό. Σήμερα ο Apache παραμένει ο δημοφιλέστερος web server, κατέχοντας πάνω από το 50% των σελίδων του διαδικτύου.

3.2 Αποτελέσματα μετρήσεων

Στη συνέχεια παρουσιάζονται τα συγκεντρωτικά αποτελέσματα που προέκυψαν από την διενέργεια των μετρήσεων.

3.2.1 Στατικές ιστοσελίδες

Ι. Μετρήσεις σε Linux

Στις μετρήσεις που διεξήχθησαν με τα τρία διαφορετικά εργαλεία, το μέγιστο των χρηστών που εξυπηρετήθηκαν ταυτόχρονα ήταν 1000. Το όριο αυτό επιτεύχθηκε μόνο με το Jmeter, καθώς τα άλλα δύο εργαλεία δεν κατάφεραν να μετρήσουν πάνω από 200 χρήστες (ο server έκλεινε μόνος του τη σύνδεση). Επίσης όταν διενεργούνταν δοκιμές με περισσότερους χρήστες στο Jmeter, το αποτέλεσμα ήταν η εμφάνιση συνεχών σφαλμάτων όπως επίσης και πολύ αργή εξυπηρέτηση αιτημάτων (πχ. Ο μέσος όρος εξυπηρέτησης των αιτημάτων για 5000 ταυτόχρονων χρηστών έφτανε μέχρι και 15 δευτερόλεπτα ανά αίτημα). Μικρά σφάλματα παρουσιάστηκαν μόνο στην περίπτωση στις δοκιμές του `complextest.htm` με 1000 χρήστες (της τάξης του 0,3%).

Στα κυρίως αποτελέσματα, παρατηρήθηκε πολλές φορές το φαινόμενο της ταχύτερης εξυπηρέτησης των αιτημάτων όσο ο συνολικός αριθμός αυτός αυξανόταν. Οι διαφορές για μικρό αριθμό χρηστών ήταν των 2 με 3 ms οπότε δεν κρίνονται άξια σχολιασμού. Όμως, όσο ο αριθμός των χρηστών αυξάνονταν, τόσο αυξανόταν και η εν λόγω διαφορά. Για το μέγιστο των 1000 χρηστών η διαφορά έφτασε τα 92 ms. Πιο συγκεκριμένα, από τις μετρήσεις του `simpletest.html` προέκυψαν τα παρακάτω:

Πίνακας 2. Αποτελέσματα για `simpletest.html` σε περιβάλλον Linux/Apache

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	3 – 9 ms
20	6 – 7 ms
50	16 - 18ms
100	31 – 35 ms

Μέτρηση απόδοσης Web Server

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
200	39 - 80 ms
500	100 - 140 ms
1000	82 - 174 ms

Παρατηρούμε, ότι όσο οι χρήστες αυξάνονται, οι χρόνοι ανεβαίνουν, το οποίο θεωρείται φυσιολογικό από την στιγμή που ο φόρτος είναι μεγαλύτερος. Επίσης, από το εργαλείο μέτρησης Pylot, παρουσιάστηκε ένα επίσης ενδιαφέρον στοιχείο. Στις μετρήσεις των 60 δευτερολέπτων, η αύξηση του αριθμού των χρηστών είχε ως αποτέλεσμα την μείωση των συνολικού αριθμού των αιτημάτων που εξυπηρετούνταν. Αντίθετα, στα 120 δευτερόλεπτα, οι αυξομειώσεις ήταν απειροελάχιστες. Αυτό, ίσως, μπορεί να μας οδηγήσει στο συμπέρασμα ότι ο web server ανταποκρίνεται καλύτερα μετά το πέρας ενός χρονικού διαστήματος, γεγονός που ενδέχεται να οφείλεται στην συνεχή του λειτουργία. Να σημειωθεί βέβαια ότι η απλότητα του υπό εξέταση αρχείου να οδηγεί σε αυτή την συμπεριφορά. Ο μέσος όρος των αιτημάτων που εξυπηρετήθηκαν σε 60 δευτερόλεπτα είναι ~15000 ενώ των 120 δευτερολέπτων είναι τα ~30000.

Στην συνέχεια ακολουθούν τα αποτελέσματα του hardtest.htm:

Πίνακας 3. Αποτελέσματα για hardtest.htm σε περιβάλλον Linux/Apache

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	46 - 60 ms
20	76 - 95 ms
50	117 - 216ms
100	198 - 530 ms
200	93 - 805 ms
500	1356 - 1778 ms

Μέτρηση απόδοσης Web Server

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
1000	3661- 4294 ms

Εδώ βλέπουμε ότι η πολυπλοκότητα του αρχείου αυξάνει το χρόνο απόκρισης. Επίσης, σε αντίθεση με την προηγούμενη μέτρηση, όταν ο αριθμός του συνολικού αριθμού των αιτημάτων μεγάλωνε, δεν γινόταν μείωση του χρόνου απόκρισης (εξαιρέση αποτελούν οι δοκιμές με 500 και 1000 χρήστες). Δηλαδή, περισσότεροι χρήστες, περισσότερη καθυστέρηση απόκρισης. Παράλληλα, οι αυξομειώσεις στις δοκιμές 60 και 120 δευτερολέπτων δεν ήταν σημαντικές. Οι μέσοι όροι που προέκυψαν είναι ~8100 (60 δευτερόλεπτα) και ~16500 αιτήματα (για 120 δευτερόλεπτα). Τέλος, στα αποτελέσματα που καταγράφηκαν, εμφανίστηκαν και τα πρώτα σφάλματα τα οποία κυμάνθηκαν σε πολύ χαμηλά επίπεδα (1000 χρήστες – σφάλματα 0,3%).

II. Μετρήσεις σε Windows

Ο Apache, ως ο πιο δημοφιλής web server, συγκεντρώνει και το μεγαλύτερο ενδιαφέρον των web developers. Αυτό έχει σαν αποτέλεσμα να δημιουργηθούν ανεξάρτητες ομάδες οι οποίες δημιουργούν ένα συγκεντρωτικό «πακέτο», το οποίο περιλαμβάνει τον Apache web server, τη MySQL και την PHP, προρυθμισμένες και έτοιμες προς χρήση, με μοναδική απαίτηση τις μικρές αλλαγές που απαιτούνται ανάλογα με την περίπτωση. Σε περιβάλλον windows, οι υλοποιήσεις αυτές είναι αρκετά δημοφιλείς λόγω των παραπάνω. Τα «πακέτα» που υπάρχουν είναι τα WAMP (από το Windows-Apache-Mysql-Php) και XAMPP (το X συμβολίζει τη λέξη cross, δηλώνοντας ότι είναι εφαρμογή cross-platform δηλ. ότι υπάρχει διανομή και για windows και για Linux. Τα υπόλοιπα είναι Apache-Mysql-PHP-Perl). Η ομάδα του WAMP παρέχει και σε Linux το αντίστοιχο πακέτο με το όνομα LAMP (Linux-Apache-Mysql-PHP).

Στις δοκιμές που πραγματοποιήθηκαν δεν χρησιμοποιήθηκε κανένα από τα παραπάνω πακέτα, αλλά προτιμήθηκε η ξεχωριστή εγκατάσταση και ρύθμιση. Οι λόγοι που έγινε αυτό είναι ώστε να χρησιμοποιηθούν οι πιο πρόσφατες εκδόσεις

Μέτρηση απόδοσης Web Server

των Apache-Mysql-PHP και επίσης να μην επιβαρυνθεί ο server με τυχόν διάφορες - ίσως και αχρείαστες - υπηρεσίες.

Όσον αφορά τις μετρήσεις, αυτές έγιναν για τα αρχεία HTML, PHP και Perl. Για την ASP.Net, το mono-project δεν έκανε εφικτό να τρέξει η γλώσσα, παρ' όλες τις διάφορες ρυθμίσεις και οδηγίες που ακολουθήθηκαν. Ο συνολικός αριθμός των χρηστών που εξυπηρετήθηκαν δεν ξεπέρασε τους 2000 (όμως σε μία μόνο περίπτωση). Σφάλματα δεν παρουσιάστηκαν πουθενά στα καταγεγραμμένα αποτελέσματα.

Πιο συγκεκριμένα, από τις μετρήσεις του simpletest.html προέκυψαν τα παρακάτω:

Πίνακας 4. Αποτελέσματα για simpletest.html σε περιβάλλον Windows/Apache

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	8 – 12 ms
20	16 – 21 ms
50	36- 41ms
100	73 – 83 ms
200	127 – 168 ms
500	223 - 421 ms
1000	422 - 811 ms
2000	1724 - 1812 ms

Παρατηρώντας τα αποτελέσματα βλέπουμε ότι με την αύξηση των χρηστών, η ψαλίδα των αποκλίσεων ανοίγει. Η διαφορές αυτές προκύπτουν μεταξύ των διαφορετικών εργαλείων μέτρησης (οι μετρήσεις με το ίδιο εργαλείο είχαν απειροελάχιστες διαφορές). Επίσης η μη ύπαρξη σφαλμάτων θα μπορούσε να οδηγήσει στο συμπέρασμα ότι γίνεται η κάθε δυνατή προσπάθεια ώστε τα αιτήματα να εξυπηρετηθούν, και για αυτό το λόγο οι μέσοι χρόνοι ανεβαίνουν τόσο

πολύ (ουσιαστικά διπλασιάζονται παράλληλα με τους χρήστες). Ο μέσος όρος των αιτημάτων που εξυπηρετήθηκαν σε 60 δευτερόλεπτα είναι ~16000 ενώ των 120 δευτερολέπτων είναι τα ~31000.

Στην συνέχεια ακολουθούν τα αποτελέσματα του hardtest.htm:

Πίνακας 5. Αποτελέσματα για hardtest.htm σε περιβάλλον Windows/Apache

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	38 - 47 ms
20	84 - 93 ms
50	155 - 235ms
100	313 - 470 ms
200	467 - 942 ms
500	1375 - 2413 ms

Εδώ βλέπουμε ότι ο αριθμός των χρηστών δεν κατάφερε να ξεπεράσει τους 500, αλλά όπως και στη προηγούμενη περίπτωση παρατηρούνται αποκλίσεις όσο αυτοί αυξάνονται. Οι μέσοι όροι που προέκυψαν είναι ~12500 (60 δευτερόλεπτα) και ~25000 αιτήματα (για 120 δευτερόλεπτα).

3.2.2 Δυναμικές ιστοσελίδες

I. Μετρήσεις σε Linux

a. PHP

Στις μετρήσεις των 3 PHP αρχείων το μέγιστο των χρηστών που επιτεύχθηκε ήταν και εδώ οι 1000. Σφάλματα παρουσιάστηκαν από τους 500 χρήστες σε μικρή κλίμακα (ακόμα και στην δοκιμή του simpletest.php), ενώ για τους 1000 το ποσοστό εκτοξεύτηκε ακόμα και στο 90%. Στην περίπτωση της μέτρησης του hardtest.php, ο επεξεργαστής του server χρησιμοποιούσε το σύνολο των πόρων (η χρήση έφτασε στο 100%) σε πολλές δοκιμές λόγω της πολυπλοκότητας του αρχείου και των υπολογισμών που έπρεπε να

Μέτρηση απόδοσης Web Server

πραγματοποιήσει. Ο συνολικός αριθμός των αιτημάτων που αποστέλλονταν παρουσίαζαν άνοδο των μέσων όρων όσο αυτά αυξάνονταν. Αναλυτικότερα για το simpletest.php τα αποτελέσματα είναι:

Πίνακας 6. Αποτελέσματα για simpletest.php σε περιβάλλον Linux/Apache

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	16 - 41 ms
20	16 - 53 ms
50	32 - 56 ms
100	54 - 59 ms
200	59 - 477 ms
500	640 - 12826 ms
1000	257 - 613 ms

Από τον παραπάνω πίνακα είναι εμφανές ότι η αύξηση των χρηστών προκαλεί και άνοδο της απόκρισης. Εδώ, πρέπει να σημειωθεί, ότι οι χρόνοι που προέκυψαν για τους 1000 χρήστες δεν μπορούν να θεωρηθούν αξιόπιστοι. Ο λόγος είναι ο πολύ μεγάλος αριθμός των σφαλμάτων που το χαμηλότερο που άγγιξαν ήταν το 78%. Τα αιτήματα που εξυπηρετήθηκαν για 60 δευτερόλεπτα ήταν ~9800 (εξαίρεση αποτελεί η μέτρηση των 200 χρηστών όπου τα αιτήματα έφτασαν τα 17700) και για 120 δευτερόλεπτα ~19600 (στους 200 χρήστες έφτασε τα 44600).

Ακολουθούν τα αποτελέσματα του mediumtest.php

Πίνακας 7. Αποτελέσματα για mediumtest.php σε περιβάλλον Linux/Apache

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	68 - 78 ms

Μέτρηση απόδοσης Web Server

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
20	79 - 142 ms
50	151 - 355 ms
100	506 - 714 ms
200	469 - 1414 ms
500	82929 - 155621 ms
1000	269 - 379 ms

Εδώ είναι εμφανές ότι η επεξεργαστική ισχύ που χρειάζεται επηρεάζει τους χρόνους. Χαρακτηριστικό παράδειγμα οι δοκιμές των 500 χρηστών, όπου ένα αίτημα στην καλύτερη περίπτωση χρειάζεται 82 δευτερόλεπτα. Όπως και στην προηγούμενη μέτρηση έτσι και εδώ τα αποτελέσματα των 1000 χρηστών δεν είναι αξιόπιστα. Τα σφάλματα κυμάνθηκαν μεταξύ 80% και 90%. Σε 60 δευτερόλεπτα στάλθηκαν ~8500 αιτήσεις και σε 120 ~16000.

Τέλος τα αποτελέσματα του `hardtest.php` είναι:

Πίνακας 8. Αποτελέσματα για `hardtest.php` σε περιβάλλον Linux/Apache

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	288 - 301 ms
20	466 - 598 ms
50	864 - 1492 ms
100	1997 - 2914 ms
200	3083 - 5977 ms
500	1821568 ms
1000	1586 - 8556 ms

Η αυξημένη επεξεργαστική ισχύ, είχε ως αποτέλεσμα την εμφάνιση μικρών σφαλμάτων ακόμα και από τους 200 χρήστες, ενώ οι δοκιμές 500 χρηστών με πάνω από 5000 αιτήματα συνολικά δεν ήταν εφικτές. Στους 1000 χρήστες τα σφάλματα έφτασαν μέχρι και το 93%. Σε 60 δευτερόλεπτα στάλθηκαν ~2000 αιτήματα και σε 120 λίγο πάνω από 4000.

b. ASP.net

Η ASP.net στον Apache μπορεί να «τρέξει» με τη ενσωμάτωση του mono-project, που κάνει εφικτή (αν και αρκετά προβληματική) τη χρήση της γλώσσας στον web server. Χαρακτηριστικό της αστάθειάς του είναι η εμφάνιση σφαλμάτων από τους μόλις 50 χρήστες, ενώ δεν μπόρεσε να εξυπηρετήσει πάνω από 500. Επίσης ο συνολικός αριθμός των αιτημάτων δεν επηρέασε σημαντικά τους μέσους όρους.

Πίνακας 9. Αποτελέσματα για simpletest.aspx σε περιβάλλον Linux/Apache

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	34 - 69 ms
20	67 - 110 ms
50	30 - 216 ms
100	37 - 267 ms
200	74 - 377 ms
500	299 - 365 ms

Παρατηρώντας τον παραπάνω πίνακα, είναι εύκολο να εντοπίσουμε ότι τα σφάλματα επηρεάζουν τα αποτελέσματα καθώς οι αποκλίσεις για ίδιες ομάδες χρηστών είναι μεγάλες. Επίσης, υπήρξε και δυσκολία καταγραφής των σφαλμάτων καθώς δεν παρουσιάστηκε μεγάλος αριθμός αυτών που να δικαιολογεί τα

Μέτρηση απόδοσης Web Server

αποτελέσματα. Αυτό ίσως είναι δείγμα της αστάθειας που προαναφέρθηκε. Σε 60 δευτερόλεπτα σταλήκαν ~8000 αιτήματα και σε 120 ~15500.

Ακολουθούν τα αποτελέσματα του `hardtest.aspx`:

Πίνακας 10. Αποτελέσματα για `hardtest.aspx` σε περιβάλλον Linux/Apache

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	120 - 173 ms
20	226 - 343 ms
50	34 - 398 ms
100	73 - 238 ms
200	149 - 243 ms
500	211 - 280 ms

Όπως και στην προηγούμενη μέτρηση, έτσι και εδώ η εμφάνιση σφαλμάτων επηρεάζει άμεσα τα αποτελέσματα, τα οποία έκαναν την εμφάνισή τους από τους 50 χρήστες. Όσο ανέβαινε ο συνολικός αριθμός των αιτημάτων, τόσο παρατηρούταν άνοδος των σφαλμάτων που έφτασαν μέχρι και το 95%. Είναι εμφανές ότι ο περισσότερος φόρτος προκαλεί κατάρρευση του `mono-project`, πράγμα που οδηγεί στο συμπέρασμα της προβληματικής του λειτουργίας στον Apache. Να σημειωθεί τέλος ότι ο επεξεργαστής χρησιμοποιούσε και εδώ το 100% της ισχύς του.

II. Perl

Η Perl είναι μία αρκετά παλιά και αργή γλώσσα. Για να τρέξει στον Apache απαιτεί κάποιες ρυθμίσεις, η οποίες όμως χαρακτηρίζονται από υψηλό βαθμό δυσκολίας και πολυπλοκότητας. Κατάφερε να αγγίξει τους 1000 χρήστες όπου ήταν και η πρώτη φορά που εμφάνισε σφάλματα (σε μικρή κλίμακα). Παρόλα αυτά δεν κατάφερε να ξεπεράσει αυτό το όριο. Αναλυτικότερα για το `simpletest.pl` προέκυψαν τα παρακάτω:

Μέτρηση απόδοσης Web Server

Πίνακας 11. Αποτελέσματα για simpletest.pl σε περιβάλλον Linux/Apache

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	48 - 78 ms
20	86 - 150 ms
50	122 - 377 ms
100	87 – 758 ms
200	266 - 1542 ms
500	15050 - 31515 ms

Από τον παραπάνω πίνακα είναι και εδώ εμφανές η αστάθεια της Perl. Ίδιες μετρήσεις με τα ίδια εργαλεία παρήγαγαν διαφορετικά αποτελέσματα. Χαρακτηριστικό παράδειγμα είναι η συμπεριφορά της ανάλογα με τον αριθμό των αιτημάτων. Σε άλλες περιπτώσεις περισσότερα αιτήματα σήμαινε περισσότερη καθυστέρηση, σε άλλα πάλι το ακριβώς αντίθετο. Επίσης όσο οι χρήστες αυξάνονταν, μεγάλωναν και οι αποκλίσεις των μέσων όρων αλλά και ο αριθμός των αιτημάτων που εξυπηρετούνταν. Πέρα από όλα αυτά, από τους χρόνους μπορούμε να διακρίνουμε πόσο αργή γλώσσα είναι.

Για hardtest.pl τα αποτελέσματα είναι:

Πίνακας 12. Αποτελέσματα για hardtest.pl σε περιβάλλον Linux/Apache

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	445 - 566 ms
20	672 - 1135 ms
50	1753 - 2267 ms
100	4328 – 5724 ms

Μέτρηση απόδοσης Web Server

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
200	7256 - 11457 ms
500	70612 - 78724 ms

Σε αντίθεση με την προηγούμενη μέτρηση οι αποκλίσεις δεν μπορούν να θεωρηθούν μεγάλες για το μέγεθος του αρχείου. Και η εδώ η επεξεργαστική ισχύει άγγιξε το 100%, μικρά σφάλματα εμφανίστηκαν στους 500 χρήστες και ο αριθμός των αιτημάτων αυξάνονταν όσο αυξάνονταν και οι χρήστες. Φυσικά οι χρόνοι μόνο ικανοποιητικοί δεν μπορούν να χαρακτηριστούν.

II. Μετρήσεις σε Windows

a. PHP

Στις μετρήσεις των 3 PHP αρχείων το μέγιστο των χρηστών που επιτεύχθηκε ήταν στα 2 από αυτά (simpletest.php, medium.php) ήταν 1000. Αντίθετα η περίπτωση του hardest.php έφτασε μόλις τους 200. Σφάλματα παρουσιάστηκαν για πρώτη φορά στην περίπτωση του simpletest.php των 500 χρηστών σε ένα ποσοστό γύρω στο 10%, το οποίο, δεν οφείλεται πιθανώς στην PHP αλλά στη συνεχή χρήση της βάσης δεδομένων. Στην περίπτωση του mediumtest.php σφάλματα εμφανίστηκαν στους 1000 χρήστες (γύρω στο 85% των αιτημάτων).

Αναλυτικότερα για το simpletest.php τα αποτελέσματα είναι:

Πίνακας 13. Αποτελέσματα για simpletest.php σε περιβάλλον Windows/Apache

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	35 - 48 ms
20	71 - 97 ms
50	141 - 246 ms
100	258 - 489 ms

Μέτρηση απόδοσης Web Server

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
200	449 - 972 ms
500	1687 - 2680 ms
1000	257 - 613 ms

Από τον παραπάνω πίνακα είναι εμφανές ότι η αύξηση των χρηστών προκαλεί και άνοδο της απόκρισης. Εδώ, πρέπει να σημειωθεί, ότι οι χρόνοι που προέκυψαν για τους 1000 χρήστες δεν μπορούν να θεωρηθούν ούτε εδώ αξιόπιστοι. Ο λόγος είναι τα σφάλματα που παρουσιάστηκαν και ήταν γύρω στο 40%. Τα αιτήματα που εξυπηρετήθηκαν για 60 δευτερόλεπτα ήταν ~12000 και για 120 δευτερόλεπτα ~22000.

Ακολουθούν τα αποτελέσματα του `mediumtest.php`

Πίνακας 14. Αποτελέσματα για `mediumtest.php` σε περιβάλλον Windows/Apache

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	48 - 87 ms
20	150 - 519 ms
50	438 - 1325 ms
100	877 - 2271 ms
200	1757 - 7193 ms
500	2214 - 4384 ms
1000	269 - 379 ms

Σε αυτή τη μέτρηση παρατηρήθηκε στις μετρήσεις του Jmeter ανώμαλες αυξομειώσεις. Σε άλλες περιπτώσεις μικρότερος συνολικός αριθμός αιτημάτων είχε μεγαλύτερους χρόνους απόκρισης για τις ίδιες ομάδες χρηστών, και σε άλλες

Μέτρηση απόδοσης Web Server

ακριβώς το αντίθετο έχοντας παράλληλα και μεγάλες αποκλίσεις. Κατά τα άλλα η συμπεριφορά του web server κυμάνθηκε όπως και στις προηγούμενες μετρήσεις σε Windows. Όπως και στην προηγούμενη μέτρηση έτσι και εδώ τα αποτελέσματα των 1000 χρηστών δεν είναι αξιόπιστα, με τα σφάλματα να φτάνουν το 90%. Σε 60 δευτερόλεπτα στάλθηκαν ~6700 αιτήσεις και σε 120 ~13000.

Τέλος τα αποτελέσματα του hardtest.php είναι:

Πίνακας 15. Αποτελέσματα για hardtest.php σε περιβάλλον Windows/Apache

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	940 - 1195 ms
20	1733 - 2392 ms
50	3987 - 5990 ms
100	7997 - 11956 ms
200	15561 - 23837 ms

Οι χρήστες που κατάφεραν να εξυπηρετηθούν σε αυτή την περίπτωση ήταν μόλις 200. Σφάλματα δεν εμφανίστηκαν στα καταγεγραμμένα αποτελέσματα. Αυτά συνέβησαν σε προσπάθειες δοκιμών πάνω από τους 200 χρήστες, όπου εκεί ο server έκλεινε η σύνδεση ή επιστρέφονταν το μήνυμα Time out. Σε 60 δευτερόλεπτα στάλθηκαν ~500 αιτήσεις και σε 120 ~1000.

b. Perl

Όπως και στην περίπτωση του Linux, η ρύθμιση της Perl στον Apache αποτέλεσε μία δύσκολη περίπτωση. Στις μετρήσεις δεν κατάφερε να ξεπεράσει τους 200 χρήστες, χωρίς ωστόσο να εμφανίσει κάποιο σφάλμα. Αναλυτικότερα για το simpletest.pl προέκυψαν τα παρακάτω:

Πίνακας 16. Αποτελέσματα για simpletest.pl σε περιβάλλον Windows/Apache

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	140 - 413 ms

Μέτρηση απόδοσης Web Server

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
20	281 - 839 ms
50	711 - 2147 ms
100	1481 – 3767 ms
200	2864 - 5920 ms

Από τον παραπάνω πίνακα παρατηρούνται μεγάλες αποκλίσεις για τους ίδιους χρήστες, η οποία οφείλετε πάλι στα εργαλεία μέτρησης. Σε 60 δευτερόλεπτα στάλθηκαν ~4100 αιτήσεις και σε 120 ~8200.

Για `hardtest.pl` τα αποτελέσματα είναι:

Πίνακας 17. Αποτελέσματα για `hardtest.pl` σε περιβάλλον Windows/Apache

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	591 - 848 ms
20	857 - 1696 ms
50	2155 - 4244 ms
100	5828 – 8510 ms
200	10985 - 16979 ms

Και εδώ ισχύει ότι και προηγουμένως σχετικά με τις αποκλίσεις. Σε 60 δευτερόλεπτα στάλθηκαν ~700 αιτήσεις και σε 120 ~1400.

3.3 Συμπεράσματα

I. Εγκατάσταση / Παραμετροποίηση

Η εγκατάσταση του Apache web server, τόσο σε περιβάλλον Linux όσο και σε Windows, είναι μία πάρα πολύ απλή και κατανοητή διαδικασία χωρίς πολύπλοκες δοκιμασίες. Το ίδιο ισχύει και για την ρύθμιση του καθώς δεν απαιτεί μεγάλες αλλαγές για τη λειτουργία του, ενώ περαιτέρω αλλαγές είναι πολύ εύκολο να γίνουν. Επίσης η ύπαρξη των WAMP, LAMP και XAMPP κάνει ακόμα ευκολότερη την παραπάνω διαδικασία.

II. Σύγκριση μεταξύ των περιβαλλόντων Linux και Windows

Κοιτάζοντας τα παραπάνω αποτελέσματα, βλέπουμε ότι στην πλειονότητα των μετρήσεων ο Apache εξυπηρετεί γρηγορότερα σε περιβάλλον Linux τα αιτήματα των χρηστών. Αντίθετα, σε περιβάλλον Windows τα αιτήματα που εξυπηρετήθηκαν σε 60 και 120 δευτερόλεπτα ήταν περισσότερα για τα αρχεία HTML και PHP (εξαίρεση αποτελούν τα αρχεία Perl).

Παίρνοντας κάθε ομάδα αρχείων ξεχωριστά, βλέπουμε ότι οι χρήστες που εξυπηρετήθηκαν στις περισσότερες περιπτώσεις ήταν οι ίδιοι. Διαφορές υπήρξαν μεταξύ των αρχείων `hardtest.php`, όπου σε περιβάλλον Linux έφτασαν τους 1000 και σε Windows μόλις του 200. Το ίδιο συνέβη και στην περίπτωση των Perl αρχείων. Από την άλλη μεριά, δεν πρέπει να παραλείψουμε το γεγονός ότι στα Windows τα σφάλματα που καταγράφηκαν ήταν λιγότερα.

Φυσικά, κοιτώντας τους μέσους όρους είναι εμφανή η υπεροχή σε περιβάλλον Linux. Σε μικρό αριθμό χρηστών οι διαφορές δεν είναι τόσο μεγάλες, αλλά όσο αυτοί αυξάνονται οι διαφορές αποκτούν μεγαλύτερες διαστάσεις.

Τέλος, δεν πρέπει να παραβλέψουμε την περίπτωση της ASP.Net. Το φιλόδοξο `mono-project`, που υπόσχεται τη λειτουργία της γλώσσας στον Apache (και όχι μόνο), τελικά λειτούργησε μόνο στην περίπτωση του Linux. Σε περιβάλλον Windows, παρ' όλες τις ρυθμίσεις και τις οδηγίες που ακολουθήθηκαν δεν υπήρξε κάποιο αποτέλεσμα. Επίσης και σε περιβάλλον Linux δεν μπορούμε να το χαρακτηρίσουμε αξιόπιστο, καθώς πολλές φορές η λειτουργία του σταματούσε ακόμα και σε μικρό αριθμό χρηστών, προκαλώντας την κατάρρευση του ίδιου του

Μέτρηση απόδοσης Web Server

Apache. Γενικά, καταλήγουμε στη διαπίστωση ότι το mono-project αποτελεί μία αρκετά ασταθής υλοποίηση.

Κεφάλαιο 4 - IIS (Internet Information Services)

4.1 Ιστορικά στοιχεία

Ο IIS είναι ένας web server που δημιουργήθηκε από την Microsoft με αποκλειστική χρήση σε περιβάλλον Windows. Αποτελεί εξέλιξη του πρώτου web server της εταιρίας, που δημιουργήθηκε από το Ερευνητικό Ακαδημαϊκό Κέντρο του Ευρωπαϊκού σκέλους της Microsoft, μέρος του πανεπιστημίου του Εδιμβούργου (Σκωτία). Όμως η ραγδαία εξέλιξη του διαδικτύου, σε συνδυασμό με την αδυναμία του server να εξυπηρετήσει το μεγάλο φόρτο προς τη σελίδα της εταιρίας (www.microsoft.com), οδήγησαν στη δημιουργία του IIS.

Οι πρώτες του εκδόσεις αποτελούσαν ξεχωριστή προσθήκη στο λειτουργικό σύστημα. Με την έκδοση 2.0 έγινε και η εισαγωγή της ASP γλώσσας, ενώ από την 5.0 και έπειτα βρίσκεται ενσωματωμένος μέσα στα Windows, ως ανενεργή υπηρεσία ενεργοποιούμενη από το χρήστη. Η έκδοση 7.0 επανασχεδιάστηκε από την αρχή, παρέχοντας ένα πιο λειτουργικό γραφικό περιβάλλον, επιτρέποντας την εύκολη ρύθμιση. Επίσης διευθέτησε πολλά ζητήματα ασφαλείας που τον είχαν κάνει ευάλωτο σε επιθέσεις.

Σήμερα, ο IIS βρίσκεται στην έκδοση 7.5, ενώ υπάρχει διαθέσιμη και η Express έκδοσή του, που είναι πιο ελαφριά, με μόνο τα απαραίτητα στοιχεία διαθέσιμα. Επίσης είναι ο δεύτερος δημοφιλέστερος web server πίσω από τον Apache, κατέχοντας το 20% των σελίδων του διαδικτύου.

Στις μετρήσεις που ακολουθούν όπως είναι φυσικό αυτές έγιναν μόνο σε περιβάλλον Windows καθώς δεν υπάρχει έκδοση διαθέσιμη για άλλο λειτουργικό σύστημα. Παράλληλα δεν έγιναν ούτε μετρήσεις των Perl αρχείων καθώς δεν βρέθηκε κάποιος ώστε η γλώσσα να υποστηριχτεί από τον IIS.

4.2 Αποτελέσματα μετρήσεων

Στη συνέχεια παρουσιάζονται τα συγκεντρωτικά αποτελέσματα που προέκυψαν από την διενέργεια των μετρήσεων.

4.2.1 Στατικές ιστοσελίδες

Στις μετρήσεις που διεξήχθησαν, σε όλες τις περιπτώσεις ο IIS κατάφερε να εξυπηρετήσει μέχρι 2000 χρήστες. Βέβαια σε όλες τις μετρήσεις των 2000 χρηστών καταγράφηκαν σφάλματα.

Πιο συγκεκριμένα, από τις μετρήσεις του simpletest.html προέκυψαν τα παρακάτω:

Πίνακας 18. Αποτελέσματα για simpletest.html σε περιβάλλον Windows/IIS

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	1 – 5 ms
20	3 – 5 ms
50	7 - 9ms
100	12 – 18 ms
200	17 – 32 ms
500	65 - 101 ms
1000	131 - 196 ms
2000	583 - 634 ms

Από τα παραπάνω, βλέπουμε την γρήγορη ανταπόκριση του server και παράλληλα κατάφερε να φτάσει τους 2000 χρήστες. Επίσης δεν παρουσιάστηκαν και μεγάλες αποκλίσεις μεταξύ των αποτελεσμάτων. Σφάλματα εμφανίστηκαν στους 2000 χρήστες σε ποσοστό περίπου στο 30%. Ο μέσος όρος των αιτημάτων που εξυπηρετήθηκαν σε 60 δευτερόλεπτα είναι ~16500 ενώ των 120 δευτερολέπτων είναι τα ~33000.

Στην συνέχεια ακολουθούν τα αποτελέσματα του hardtest.htm:

Πίνακας 19. Αποτελέσματα για hardtest.htm σε περιβάλλον Windows/IIS

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	45 - 48 ms
20	89 - 97 ms
50	226 - 236ms
100	373 - 473 ms
200	613 - 947 ms
500	2348 - 3034 ms
1000	3770 - 5179 ms
2000	23262 - 23960 ms

Εδώ τα αποτελέσματα δεν διαφοροποιήθηκαν από την προηγούμενη μέτρηση εκτός μόνο από τη φυσική αύξηση των μέσων όρων λόγω του μεγέθους του αρχείου. Σφάλματα είχαμε και εδώ στους 2000 χρήστες σε μεγάλο ποσοστό που έφτασε στο 85% των αιτημάτων. Οι μέσοι όροι που προέκυψαν είναι ~12500 (60 δευτερόλεπτα) και ~25300 αιτήματα (για 120 δευτερόλεπτα).

4.2.2 Δυναμικές ιστοσελίδες

a. PHP

Η PHP υποστηρίζεται από τον IIS μετά συγκεκριμένες ρυθμίσεις που πρέπει να γίνουν οι οποίες χαρακτηρίζονται από μεσαίο βαθμό δυσκολίας. Στις μετρήσεις τώρα και στα 3 αρχεία εξυπηρετήθηκαν ταυτόχρονα μέχρι 2000 χρήστες, όπου και εμφανίστηκαν σφάλματα.

Μέτρηση απόδοσης Web Server

Αναλυτικότερα για το simpletest.php τα αποτελέσματα είναι:

Πίνακας 20. Αποτελέσματα για simpletest.php σε περιβάλλον Windows/IIS

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	66 - 86 ms
20	124 - 138 ms
50	267 - 329 ms
100	406 - 641 ms
200	709 - 1251 ms
500	2303 - 3156 ms
1000	5107 - 6320 ms
2000	1615 - 1716 ms

Από τον παραπάνω πίνακα είναι εμφανές ότι η αύξηση των χρηστών προκαλεί και άνοδο της απόκρισης. Η μέτρηση των 2000 χρηστών δεν θεωρείται αξιόπιστη λόγω των σφαλμάτων που άγγιξαν το 25% και προφανώς δεν μπορούν να θεωρηθούν αξιόπιστα. Τα αιτήματα που εξυπηρετήθηκαν για 60 δευτερόλεπτα ήταν ~8500 και για 120 δευτερόλεπτα ~16500.

Ακολουθούν τα αποτελέσματα του mediumtest.php

Πίνακας 21. Αποτελέσματα για mediumtest.php σε περιβάλλον Windows/IIS

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	111 - 146 ms
20	249 - 282 ms
50	509 - 793 ms
100	866 - 1420 ms

Μέτρηση απόδοσης Web Server

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
200	1807 - 3246 ms
500	6068 - 7498 ms
1000	12231 - 16231 ms
2000	1735 - 1888 ms

Τα αποτελέσματα είναι χαρακτηριστικά του αρχείου που μετρήθηκε. Παρατηρούμε ότι με την αύξηση των χρηστών αυξήθηκαν και οι αποκλίσεις. Σφάλματα και εδώ είχαμε στους 2000 που έφτασαν στο 25% των αιτημάτων. Φυσικά και αυτή η μέτρηση όπως και σε παρόμοιες περιπτώσεις δεν θεωρείται αξιόπιστη. Σε 60 δευτερόλεπτα στάλθηκαν ~4300 αιτήσεις και σε 120 ~8700.

Τέλος τα αποτελέσματα του `hardtest.php` είναι:

Πίνακας 22. Αποτελέσματα για `hardtest.php` σε περιβάλλον Windows/IIS

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	239 - 312 ms
20	573 - 623 ms
50	979 - 1573 ms
100	1685 - 3135 ms
200	3526 - 6285 ms
500	13447 - 15406 ms
1000	26653 - 31214 ms
2000	1682 - 2571 ms

Μέτρηση απόδοσης Web Server

Οι χρήστες που κατάφεραν να εξυπηρετηθούν σε αυτή την περίπτωση ήταν και εδώ 2000. Ο IIS παρόλο την επεξεργαστική ισχύ που χρειάστηκε κατάφερε να εξυπηρετήσει των εν λόγω αριθμό χρηστών, όπως και στις προηγούμενες 2 μετρήσεις. Τα σφάλματα και σε αυτή την περίπτωση εμφανίστηκαν στους 2000 χρήστες σε ποσοστό γύρω στο 50%. Σε 60 δευτερόλεπτα στάλθηκαν ~2000 αιτήσεις και σε 120 ~3900.

b. ASP.Net

Η ASP.Net αποτελεί μία δυναμική γλώσσα διαδικτυακών εφαρμογών η οποία δημιουργήθηκε από την Microsoft. Επομένως η συνεργασία της με τον IIS είναι αυτονόητη και χωρίς να χρειαστεί κάποια ιδιαίτερη ρύθμιση ώστε να λειτουργήσει. Η λογική λέει ότι η γλώσσα θα έχει την καλύτερη επίδοσή της στον εν λόγω web server. Αυτό θα φανεί από τα αποτελέσματα που προέκυψαν.

Αναλυτικότερα για το simpletest.aspx προέκυψαν τα παρακάτω:

Πίνακας 23. Αποτελέσματα για simpletest.aspx σε περιβάλλον Windows/IIS

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	20 – 50 ms
20	30 - 44 ms
50	35 - 95 ms
100	185 – 191 ms
200	365 - 380 ms
500	894 - 897 ms
1000	1785 - 1789 ms
2000	4775 - 5053 ms

Μέτρηση απόδοσης Web Server

Από τον παραπάνω πίνακα βλέπουμε ότι οι χρήστες που κατάφεραν να εξυπηρετηθούν ήταν 2000. Και εδώ όμως στους 2000 χρήστες παρουσιάστηκαν σφάλματα τα οποία έφτασαν στο 90% των αιτημάτων, με τα αποτελέσματα των μέσων όρων να δείχνουν πιο αληθοφανή σε σχέση με τις μετρήσεις των PHP αρχείων. Σε 60 δευτερόλεπτα στάλθηκαν ~16000 αιτήσεις και σε 120 ~32000.

Για `hardtest.aspx` τα αποτελέσματα είναι:

Πίνακας 24. Αποτελέσματα για `hardtest.aspx` σε περιβάλλον Windows/IIS

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	19 - 49 ms
20	26 - 96 ms
50	234 - 247 ms
100	468 - 471 ms
200	868 - 871 ms
500	2329 - 2333 ms
1000	4658 - 4700 ms
2000	22212 - 23450 ms

Και εδώ οι ιδιότητες των αποτελεσμάτων είναι οι ίδιες με προηγουμένως με μόνη εξαίρεση φυσικά τους μέσους όρους. Για άλλη μια φορά τα σφάλματα εμφανίστηκαν στους 2000 χρήστες σε ποσοστό 90% Σε 60 δευτερόλεπτα στάλθηκαν ~12700 αιτήσεις και σε 120 ~25000.

4.3 Συμπεράσματα

I. Εγκατάσταση / Παραμετροποίηση

Αρχικά, ο IIS δεν χρειάστηκε ξεχωριστή εγκατάσταση, καθώς ήταν ήδη ενσωματωμένος μέσα στο λειτουργικό σύστημα (αυτό ισχύει από την έκδοση των Windows XP και έπειτα). Το μόνο που χρειάστηκε ήταν μία απλή διαδικασία ενεργοποίησης, χωρίς κάποια ιδιαίτερη δυσκολία ρύθμισης. Επίσης, και από τη στιγμή που όλα γίνονται μέσω γραφικού περιβάλλοντος, είναι αρκετά προσιτός προς έναν άπειρο χρήστη. Η μόνη δυσκολία ήταν στην ρύθμιση της PHP η οποία είναι μία όχι και τόσο εύκολη διαδικασία. Με την ASP.Net φυσικά δεν υπήρξε κανένα απολύτως πρόβλημα.

II. Σύγκριση μεταξύ των περιβαλλόντων Linux και Windows

Όπως προαναφέρθηκε, σύγκριση μεταξύ των δυο λειτουργικών συστημάτων δεν ήταν δυνατή να γίνει, καθώς υλοποίηση του IIS σε Linux δεν υπάρχει. Επομένως τα αποτελέσματα χρησιμεύουν για σύγκριση με τους υπόλοιπους web server.

Ως κάποια συμπεράσματα, μπορούμε να αναγνωρίσουμε στον IIS την σταθερότητά του, καθώς σε καμία περίπτωση δεν σταμάτησε τη λειτουργία του κάτω από συνθήκες μετρήσεων (δηλ. δεν χρειάστηκε επανεκκίνηση του λόγω σταματήματος της υπηρεσίας του). Άλλο θετικό στοιχείο του, είναι ότι κατάφερε να εξυπηρετήσει το μεγαλύτερο αριθμό χρηστών (2000) σε σχέση με τους άλλους web servers. Επίσης, είναι αρκετά εύκολος στη ρύθμισή του, γεγονός που βοηθάει και η χρήση του γραφικού του περιβάλλοντος, ενώ η επανασχεδίαση της έκδοσης 7.0 έδωσε τη δυνατότητα της ταυτόχρονης υποστήριξης περισσότερων ιστοσελίδων των μία ιστοσελίδων. Αντίθετα, στα αρνητικά μπορούμε να αναφέρουμε ότι εκτός από την ASP.Net και την PHP δεν φαίνεται να μπορεί να υποστηρίξει κάποια άλλη γλώσσα. Παράλληλα, το γεγονός ότι δεν είναι cross-platform web server τον κάνει να χάνει ένα μεγάλο μερίδιο του συνόλου των σελίδων του διαδικτύου.

Κεφάλαιο 5 - Nginx

5.1 Ιστορικά στοιχεία

Ο Nginx (προφέρεται engine-x), δημιουργήθηκε από τον Igor Sysoen με την ανάπτυξη του χρονολογείτε από το 2002. Η πρώτη επίσημη έκδοση του δόθηκε στο κοινό το 2004 και χρησιμοποιήθηκε αρχικά από πολλές ρώσικης προέλευσης σελίδες όπως η Yandex και Mail.ru, η οποίες χαρακτηρίζονται από υψηλή επισκεψιμότητα. Σήμερα το 11% των ιστοσελίδων του διαδικτύου τον χρησιμοποιούν με πιο γνωστές τις σελίδες Wordpress.org, SourceForge.com και TorrentReactor.com, ενώ σύμφωνα με προβλέψεις θα ξεπεράσει σύντομα τον ISS ως ο δεύτερος πιο δημοφιλής web server μετά τον Apache.

Ο Nginx διαφημίζεται ως ένας σταθερός, υψηλής απόδοσης web server, εύκολος στη ρύθμιση και με χαμηλή κατανάλωση ισχύος. Η σχεδίαση του διαφέρει από τους υπόλοιπους web servers, στο ότι δεν χρησιμοποιεί διαφορετικές διεργασίες για την εξυπηρέτηση των αιτημάτων, αλλά βασίζεται σε μία ασύγχρονη, οδηγούμενη από γεγονότα αρχιτεκτονική. Σύμφωνα με αυτή, ένα μικρό μέρος της μνήμης δεσμεύεται κάτω από συνθήκες φόρτου, το μέγεθος της οποίας προκύπτει με τη χρήση ενός αλγόριθμου πρόβλεψης. Έτσι εξασφαλίζεται η λειτουργία του σε περιπτώσεις εξυπηρέτησης μεγάλου αριθμού αιτημάτων.

Οι μετρήσεις που ακολουθούν ήταν εφικτές μόνο σε περιβάλλον Linux. Αν και υπάρχει επίσημη έκδοση για Windows, η μέτρηση της λειτουργίας του Nginx με την PHP, ASP.Net και Perl ήταν αδύνατη. Χαρακτηριστικό παράδειγμα του παραπάνω είναι ότι στην επίσημη σελίδα του web server, αναφέρεται ότι είναι δοκιμασμένος μόνο σε Windows XP και Windows Server 2003. Με την ASP.Net το πρόβλημα ήταν το ίδιο όπως και στους υπόλοιπους web servers. Με τις PHP και Perl, ενώ η ρύθμισή τους έγινε κανονικά, δεν καταγράφηκε καμία μέτρηση. Αυτό έγινε γιατί από την πρώτη κιόλας δοκιμή η υπηρεσία της PHP (ή της Perl) σταματούσε τη λειτουργία της μη κάνοντας δυνατή την παραγωγή οποιουδήποτε αποτελέσματος. Φυσικά η μόνη λύση ήταν η επανεκκίνηση τόσο της υπηρεσίας όσο και του web server.

5.2 Αποτελέσματα μετρήσεων

Στη συνέχεια παρουσιάζονται τα συγκεντρωτικά αποτελέσματα που προέκυψαν από την διενέργεια των μετρήσεων.

5.2.1 Στατικές ιστοσελίδες

Στις μετρήσεις που διεξήχθησαν με τα τρία διαφορετικά εργαλεία, το μέγιστο των χρηστών που εξυπηρετήθηκαν ταυτόχρονα ήταν 1000 και στα 2 αρχεία. Σφάλματα εμφανίστηκαν μόνο στην μέτρηση των 1000 του αρχείου complextest.htm.

Πιο συγκεκριμένα, από τις μετρήσεις του simpletest.html προέκυψαν τα παρακάτω:

Πίνακας 25. Αποτελέσματα για simpletest.html σε περιβάλλον Linux/Nginx

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	1 – 3 ms
20	2 ms
50	5 – 6 ms
100	10 – 19 ms
200	22 - 30 ms
500	66 - 68 ms
1000	120 - 132 ms

Όπως βλέπουμε από τον παραπάνω πίνακα, ο web server ανταποκρίνεται γρήγορα στα αιτήματα των χρηστών. Φυσικά δεν πρέπει να ξεχνάμε ότι πρόκειται για ένα απλό αρχείο χωρίς ιδιαιτερότητες. Ο μέσος όρος των αιτημάτων που εξυπηρετήθηκαν σε 60 δευτερόλεπτα είναι ~15500 ενώ των 120 δευτερολέπτων είναι τα ~32000.

Μέτρηση απόδοσης Web Server

Στην συνέχεια ακολουθούν τα αποτελέσματα του hardtest.htm:

Πίνακας 26. Αποτελέσματα για hardtest.htm σε περιβάλλον Linux/Nginx

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	42 - 45 ms
20	86 - 92 ms
50	172 - 225 ms
100	311 - 380 ms
200	583 - 864 ms
500	2012 - 2211 ms
1000	3744- 4155 ms

Εδώ οι χρόνοι όπως είναι φυσικό οι χρόνοι αυξήθηκαν λόγω του μεγέθους του αρχείου. Ελάχιστα σφάλματα παρουσιάστηκαν στους 1000 χρήστες, περίπου στο 3% των αιτημάτων. Οι μέσοι όροι αιτημάτων που προέκυψαν είναι ~12000 (60 δευτερόλεπτα) και ~24000 αιτήματα (για 120 δευτερόλεπτα).

5.2.2 Δυναμικές ιστοσελίδες

I. Μετρήσεις σε Linux

a. PHP

Στις μετρήσεις των 3 PHP αρχείων το μέγιστο των χρηστών που επιτεύχθηκε ήταν 500. Σφάλματα παρουσιάστηκαν σε μετρήσεις και των 3 αρχείων. Αναλυτικότερα για το simpletest.php τα αποτελέσματα είναι:

Μέτρηση απόδοσης Web Server

Πίνακας 27. Αποτελέσματα για simpletest.php σε περιβάλλον Linux/Nginx

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	11 - 16 ms
20	22 - 32 ms
50	38 - 86 ms
100	56 - 164 ms
200	72 - 565 ms
500	269 - 678 ms

Όπως βλέπουμε από τον πίνακα, η μέσοι όροι κρατήθηκαν σε χαμηλά επίπεδα, ενώ οι αποκλίσεις προήλθαν για άλλη μία φορά από τα αποτελέσματα των διαφορετικών εργαλείων. Τα μόνα σφάλματα που παρουσιάστηκαν εδώ ήταν στη μέτρηση των 500 χρηστών και σε ποσοστό λίγο πάνω από 1%. Τα αιτήματα που εξυπηρετήθηκαν για 60 δευτερόλεπτα ήταν ~15000 και για 120 δευτερόλεπτα ~31000.

Ακολουθούν τα αποτελέσματα του mediumtest.php

Πίνακας 28. Αποτελέσματα για mediumtest.php σε περιβάλλον Linux/Nginx

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	44 - 70 ms
20	98 - 140 ms
50	146 - 351 ms
100	202 - 701 ms
200	580 - 1732 ms
500	2326 - 4418 ms

Μέτρηση απόδοσης Web Server

Όπως και την προηγούμενη μέτρηση, έχουμε αποκλίσεις στα αποτελέσματα οι οποίες μεγαλώνουν με την αύξηση των χρηστών. Σφάλματα εμφανιστήκαν από τους 200 χρήστες (περίπου στο 1%) ενώ στους 500 έφτασαν το 10% των αιτημάτων. Σε 60 δευτερόλεπτα στάλθηκαν ~8500 αιτήσεις και σε 120 ~16700.

Τέλος τα αποτελέσματα του `hardtest.php` είναι:

Πίνακας 29. Αποτελέσματα για `hardtest.php` σε περιβάλλον Linux/Nginx

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	206 - 303 ms
20	461 - 607 ms
50	997 - 1520 ms
100	1713 – 3038 ms
200	3975 - 7099 ms
500	6811 - 10916 ms

Και εδώ η μέτρηση έχει τα ίδια χαρακτηριστικά με τις προηγούμενες 2. Τα σφάλματα ξεκίνησαν και σε αυτή την περίπτωση από τους 200 χρήστες (με μικρά ποσοστά γύρω στο 1%) ενώ στους 500 έφτασαν και το 20% των αιτημάτων. Σε 60 δευτερόλεπτα στάλθηκαν ~2000 αιτήματα και σε 120 λίγο πάνω από 4000.

Γενικά από τις μετρήσεις και των 3 αρχείων βλέπουμε ότι ο Nginx ανταποκρίνεται ικανοποιητικά στην εξυπηρέτηση των PHP αιτημάτων.

b. ASP.net

Η ομάδα του `mono-project` δημιούργησε και για τον Nginx μία υλοποίηση ώστε η ASP.net να μπορεί να «τρέξει» στο web server. Τα αποτελέσματα όμως είναι παρόμοια με όλες τις άλλες περιπτώσεις αυτής της υλοποίησης. Κατ' αρχήν, ο βαθμός δυσκολίας για τη ρύθμιση του είναι μεγάλος. Οι οδηγίες που παρέχονται από την επίσημη σελίδα του `mono-project` δεν είναι επαρκής και σαφής, και η λύση βρέθηκε μέσω ανεπίσημων forum. Πάλι όμως τα

Μέτρηση απόδοσης Web Server

προβλήματα δεν σταμάτησαν εδώ καθώς η υπηρεσία του mono-project σταματούσε τη λειτουργία της συχνά απαιτώντας την συνολική επανεκκίνηση του server και όχι απλώς της υπηρεσίας. Φυσικά ούτε ο Nginx σε τέτοια περίπτωση μπορούσε να λειτουργήσει. Γενικά, ούτε σε αυτή την περίπτωση μπορούμε να πούμε ότι το mono-project ικανοποίησε με την επίδοσή του.

Πίνακας 30. Αποτελέσματα για simpletest.aspx σε περιβάλλον Linux/Nginx

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	50 - 130 ms
20	70 - 245 ms
50	119 - 617 ms
100	127 - 1183 ms
200	375 - 2495 ms

Όπως βλέπουμε στον πίνακα, μέχρι 200 χρήστες κατάφεραν να εξυπηρετηθούν, με τις αποκλίσεις να είναι και εδώ μεγάλες. Σφάλματα, εκτός από τα προαναφερθέντα προβλήματα, καταγράφηκαν στη μέτρηση των 200 χρηστών σε ποσοστό 1%. Σε 60 δευτερόλεπτα σταλήθηκαν ~6000 αιτήματα και σε 120 ~12500.

Ακολουθούν τα αποτελέσματα του hardtest.aspx:

Πίνακας 31. Αποτελέσματα για hardtest.aspx σε περιβάλλον Linux/Nginx

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	53 - 173 ms
20	79 - 316 ms
50	504 - 1918 ms
100	239 - 1242 ms

Μέτρηση απόδοσης Web Server

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
200	2155 - 2295 ms

Εδώ παρατηρούμε την ίδια συμπεριφορά με προηγουμένως. Τα σφάλματα εμφανιστήκαν για πρώτη φορά από τους 50 χρήστες. Γενικότερα, η μέτρηση είχε πολλές ιδιαιτερότητες λόγω του γνωστού προβλήματος που αντιμετωπίστηκε. Σε 60 δευτερόλεπτα σταλθήκαν ~3300 αιτήματα και σε 120 ~6300.

c. Perl

Η Perl μπορεί να υποστηριχτεί από τον Nginx μετά από κατάλληλες ρυθμίσεις, η οποίες δεν παρουσίασαν μεγάλο βαθμό δυσκολίας. Και στις 2 περιπτώσεις το μέγιστο των χρηστών που επιτεύχθηκε ήταν οι 200. Αναλυτικότερα για το simpletest.pl προέκυψαν τα παρακάτω:

Πίνακας 32. Αποτελέσματα για simpletest.pl σε περιβάλλον Linux/Nginx

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	44 - 122 ms
20	88 - 214 ms
50	221 - 756 ms
100	445 - 895 ms
200	707 - 1545 ms

Σφάλματα εμφανίστηκαν στους 200 χρήστες τα οποία έφτασαν γύρω στο 10%. Σε 60 δευτερόλεπτα σταλθήκαν ~13000 αιτήματα και σε 120 ~24000.

Μέτρηση απόδοσης Web Server

Για hardtest.pl τα αποτελέσματα είναι:

Πίνακας 33. Αποτελέσματα για hardtest.pl σε περιβάλλον Linux/Nginx

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	264 - 564 ms
20	577 - 1136 ms
50	1470 - 2893 ms
100	3087 - 5724 ms
200	16663 - 25321 ms

Εδώ σφάλματα είχαμε πάλι στους 200 χρήστες αλλά πιο εκτεταμένα (έφτασαν ακόμα και το 85%). Σε 60 δευτερόλεπτα σταλήκαν ~1150 αιτήματα και σε 120 ~2300.

Γενικά και στις 2 μετρήσεις των Perl αρχείων παρουσιάστηκε παρόμοια συμπεριφορά με τις μετρήσεις των PHP και ASP.Net.

5.3 Συμπεράσματα

I. Εγκατάσταση / Παραμετροποίηση

Η εγκατάσταση του Nginx και στα δυο περιβάλλοντα δεν εμφάνισε καμία δυσκολία. Η ρύθμιση του από την άλλη εμφάνισε προβλήματα, με εξαίρεση τη λειτουργία της PHP σε Linux. Από τη μία η ρύθμιση (της ανεπίσημα υποστηριζόμενης) ASP.Net και Perl δεν ήταν απλή υπόθεση, ενώ σε Windows η ρύθμιση της PHP και της Perl έγινε, αλλά από εκεί και πέρα ουσιαστικά δεν λειτούργησαν ποτέ. Η ουσία είναι πάντως ότι ο Nginx δεν έχει τόσο απλή και εύκολη ρύθμιση όπως ισχυρίζεται.

II. Σύγκριση μεταξύ των περιβαλλόντων Linux και Windows

Από τα παραπάνω, μόνο για τις μετρήσεις σε Linux μπορούμε να μιλήσουμε αφού σε Windows αυτές δεν ήταν εφικτό να γίνει. Η μόνη που έγιναν ήταν για τα αρχεία HTML, οι οποίες δεν παρουσίασαν σημαντικές διαφορές με αυτές σε Linux και δεν καταγράφηκαν εδώ καθώς μεγαλύτερη σημασία έχει πως ανταποκρίνεται ένας web server σε δυναμικό περιεχόμενο.

Στα αποτελέσματα τώρα, οι αποδόσεις σε HTML και PHP κρίνονται ικανοποιητικές, δικαιολογώντας τη φήμη του web server. Με την ASP.Net το μεγαλύτερο πρόβλημα είναι το ότι σταματάει τη λειτουργία της μετά από τυχαίου τύπου μετρήσεις. Επίσης, ο αριθμός των χρηστών που κατάφεραν να εξυπηρετηθούν δεν μπορεί να θεωρηθεί ιδιαίτερα ικανοποιητικός, ειδικά για ένα web server ο οποίος ισχυρίζεται ότι μπορεί να εξυπηρετήσει μέχρι 10000 χρήστες την ίδια χρονική στιγμή. Γενικά, τα εν λόγω αποτελέσματα είναι χρήσιμα μόνο για σύγκριση με τους υπόλοιπους web servers.

Κεφάλαιο 6 - Lighttpd

6.1 Ιστορικά στοιχεία

Ο Lighttpd είναι ένας web server ανοιχτού κώδικα γραμμένος σε γλώσσα C. Δημιουργός του είναι ο Jan Kneschike και η πρώτη έκδοση του βγήκε στο διαδίκτυο το Μάρτιο του 2003 για λειτουργικά συστήματα Linux και Windows. Διαφημίζεται ως ένας γρήγορος, «ελαφρύς» web server σύμφωνα με όλες τις σύγχρονες προδιαγραφές, παραμένοντας ταυτόχρονα ασφαλής και λειτουργικός. Βασικό του πλεονέκτημα είναι η μικρή χρήση του επεξεργαστή και της μνήμης του server, κάνοντας τον κατάλληλο για μηχανήματα που παρουσιάζουν μεγάλο φόρτο. Έχει τη δυνατότητα να «τρέχει» διαφορετικές γλώσσες με την χρήση των κατάλληλων προγραμμάτων, με περισσότερη προσοχή να έχει δοθεί στην PHP ως η πιο δημοφιλή γλώσσα δικτυακού προγραμματισμού. Στον αντίποδα δεν υποστηρίζει την ASP.net.

Ο Lighttpd χρησιμοποιείτε σήμερα σε αρκετές ιστοσελίδες που παρουσιάζουν μεγάλο φόρτο, με πιο δημοφιλείς τις σελίδες Youtube, Meebo, Wikipedia και SourceForge. Επίσης, εμφανίζεται σε 3 δημοφιλείς torrent σελίδες (The Pirate Bay, Mininova, IsoHunt), οι οποίες έχουν πάνω από 1000 επισκέψεις το δευτερόλεπτο. Παρόλα αυτά το ποσοστό του στο χώρο των web servers παραμένει ακόμα πολύ χαμηλό (κάτω από 1%).

6.2 Αποτελέσματα μετρήσεων

Στη συνέχεια παρουσιάζονται τα συγκεντρωτικά αποτελέσματα που προέκυψαν από την διενέργεια των μετρήσεων.

6.2.1 Στατικές ιστοσελίδες

Ι. Μετρήσεις σε Linux

Στις μετρήσεις που διεξήχθησαν, ο Lighttpd κατάφερε να εξυπηρετήσει μέχρι 1000 χρήστες (μόνο με το Jmeter. Το ab tool έφτασε μέχρι τους 500 και το Pylot μέχρι 200). Παραπάνω χρήστες οδηγούσαν ή σε μεγάλο αριθμό σφαλμάτων ή έκλεινε η σύνδεση από το server. Στα καταγεγραμμένα αποτελέσματα μικρά σφάλματα μόνο στις δοκιμές με 1000 χρήστες στο `complextest.htm` εμφανίστηκαν.

Στα κυρίως αποτελέσματα, μέχρι τους 200 χρήστες ο αριθμός των συνολικών αιτημάτων δεν επηρέασε το μέσο χρόνο εξυπηρέτησης τους. Στους 500 και στους 1000 χρήστες, όπου ο αριθμός των αιτημάτων ήταν και μεγαλύτερος, ο χρόνος αυτός μειωνόταν. Επίσης όμως, η αύξηση των χρηστών είχε και ως αποτέλεσμα τη μείωση των αιτημάτων. Πιο συγκεκριμένα, από τις μετρήσεις του `simpletest.html` προέκυψαν τα παρακάτω:

Πίνακας 34. Αποτελέσματα για `simpletest.html` σε περιβάλλον Linux/Lighttpd

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	2 ms
20	4 ms
50	6 - 11ms
100	16 – 21 ms
200	10 - 55 ms
500	35 - 139 ms
1000	82 - 174 ms

Μέτρηση απόδοσης Web Server

Από τα αποτελέσματα βλέπουμε ότι εκτός από την άνοδο των χρόνων όσο αυξάνονται οι χρήστες, οι διαφορές παρουσιάζουν μεγαλύτερες αποκλίσεις με την αύξηση αυτών. Οι αποκλίσεις αυτές προέκυψαν μεταξύ των διαφορετικών εργαλείων (μετρήσεις με το ίδιο εργαλείο δεν έδιναν σημαντικές διαφορές). Σε 60 δευτερόλεπτα ο μέσος αριθμός αιτημάτων που εξυπηρετήθηκαν είναι ~16500 και σε 120 ~34000 (με πτωτική τάση όσο οι χρήστες αυξάνονταν).

Στην συνέχεια ακολουθούν τα αποτελέσματα του `hardtest.htm`:

Πίνακας 35. Αποτελέσματα για `hardtest.htm` σε περιβάλλον Linux/Lighttpd

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	42 - 46 ms
20	72 - 92 ms
50	115 - 222ms
100	191 - 380 ms
200	547 - 860 ms
500	1549 - 1978 ms
1000	3661- 4294 ms

Όπως και στη προηγούμενη μέτρηση, έτσι και εδώ παρατηρούμε την αύξηση των αποκλίσεων. Μικρά σφάλματα εμφανίστηκαν στους 1000 χρήστες (0,3%). Σε 60 δευτερόλεπτα εξυπηρετήθηκαν περίπου ~12500 αιτήματα και σε 120 ~25000.

II. Μετρήσεις σε Windows

Στις μετρήσεις σε Windows ο αριθμός των χρηστών που εξυπηρετήθηκαν στην περίπτωση του `simpletest.html` ήταν και εδώ 1000 ενώ στην περίπτωση του `complextest.htm` δεν ξεπέρασε τους 500. Τα αποτελέσματα του `simpletest.html` είναι τα εξής.

Μέτρηση απόδοσης Web Server

Πίνακας 36. Αποτελέσματα για simpletest.html σε περιβάλλον Windows/Lighttpd

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	8 - 9 ms
20	17 -18 ms
50	42 - 54ms
100	88 – 103 ms
200	102 - 177 ms
500	449 - 454 ms
1000	908 - 956 ms

Το στοιχείο που πρέπει να αναφερθεί εδώ είναι ότι σφάλματα δεν υπήρξαν στα καταγεγραμμένα αποτελέσματα. Σε 60 δευτερόλεπτα εξυπηρετήθηκαν περίπου ~15500 αιτήματα και σε 120 ~30000.

Στην συνέχεια ακολουθούν τα αποτελέσματα του hardtest.htm:

Πίνακας 37. Αποτελέσματα για hardtest.htm σε περιβάλλον Windows/Lighttpd

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	45 - 48 ms
20	94 ms
50	186 - 235ms
100	363 - 470 ms
200	352 - 943 ms
500	2372 - 2387 ms

Όπως βλέπουμε για άλλη μία φορά, η αύξηση των χρηστών προκάλεσε και την αύξηση των αποκλίσεων μεταξύ των εργαλείων μέτρησης. Σφάλματα ούτε εδώ καταγράφηκαν. Σε 60 δευτερόλεπτα εξυπηρετήθηκαν περίπου ~12500 αιτήματα και σε 120 ~25000.

6.2.2 Δυναμικές ιστοσελίδες

Πριν προχωρήσουμε στις μετρήσεις, θα αναφερθούμε στην περίπτωση της ASP.Net. Ο Lighttpd δεν υποστηρίζει επίσημα τη γλώσσα. Η ομάδα του monoproject, όπως και στην περίπτωση του Apache, δημιούργησε μία υλοποίηση η οποία υπόσχεται τη συνεργασία του web server με την ASP.Net. Παρόλα αυτά, στην πρώτη περίπτωση σε περιβάλλον Linux η γλώσσα λειτούργησε αλλά από την πρώτη κιάλας μέτρηση η υπηρεσία «κατέρρευσε» και οποιαδήποτε άλλη προσπάθεια είχε ακριβώς το ίδιο αποτέλεσμα. Σε περιβάλλον Windows, ο Lighttpd με την ASP.Net δεν συνεργάστηκαν ποτέ. Επομένως δεν ήταν εφικτό να παραχθούν αποτελέσματα.

I. Μετρήσεις σε Linux

a. PHP

Στις μετρήσεις των 3 PHP αρχείων το μέγιστο των χρηστών που επιτεύχθηκε ήταν οι 1000. Σφάλματα παρουσιάστηκαν από τους 500 χρήστες, όπου στην καλύτερη των περιπτώσεων έφτασαν το 45%. Για τη μέτρηση του hardest.php, ο server χρησιμοποίησε το 100% των δυνατοτήτων του. Αναλυτικότερα για το simplestest.php τα αποτελέσματα είναι:

Πίνακας 38. Αποτελέσματα για simplestest.php σε περιβάλλον Linux/Lighttpd

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	12 - 24 ms
20	16 - 29 ms
50	27 - 72 ms
100	27 - 144 ms
200	24 - 290 ms

Μέτρηση απόδοσης Web Server

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
500	150 - 235 ms
1000	257 - 613 ms

Σε αυτή τη μέτρηση το ενδιαφέρον στοιχείο που προέκυψε ήταν οι μετρήσεις που πραγματοποιήθηκαν με το Pylot. Στα ίδια χρονικά διαστήματα (60, 120 δευτερόλεπτα), παρόλο που ο αριθμός των χρηστών αυξανόταν, δεν παρατηρήθηκε καμία σημαντική αλλαγή στους μέσους όρους (κυμάνθηκαν μεταξύ 23 και 28 ms). Στα άλλα δύο εργαλεία οι χρόνοι ανέβαιναν ανάλογα τους χρήστες, παρουσίαζαν όμως και αυτά αποκλίσεις μεταξύ τους σε ορισμένες ομάδες χρηστών. Σφάλματα παρουσιάστηκαν μόνο στη μέτρηση των 500 χρηστών με το ab tool. Σε 60 δευτερόλεπτα εξυπηρετήθηκαν ~15500 αιτήματα και σε 120 ~31000.

Ακολουθούν τα αποτελέσματα του `mediumtest.php`

Πίνακας 39. Αποτελέσματα για `mediumtest.php` σε περιβάλλον Linux/Lighttpd

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	60 - 77 ms
20	93 - 156 ms
50	166 - 394 ms
100	281 - 788 ms
200	588 - 1569 ms
500	371 - 648 ms
1000	296 - 379 ms

Όπως φαίνεται από τον πίνακα, εμφανίζονται και εδώ μεγαλύτερες αποκλίσεις όσο αυξάνονται οι χρήστες. Κύριος υπεύθυνος για αυτό εδώ είναι το

Μέτρηση απόδοσης Web Server

Jmeter καθώς τα άλλα δύο εργαλεία εμφανίζουν να συμφωνούν μεταξύ τους. Επίσης, τα αποτελέσματα των 500 και 1000 χρηστών δεν μπορούν να θεωρηθούν αξιόπιστα λόγω του μεγάλου αριθμού των σφαλμάτων που κυμάνθηκαν μεταξύ 60% και 90%. Σε 60 δευτερόλεπτα εξυπηρετήθηκαν ~7600 αιτήματα και σε 120 ~15200.

Τέλος τα αποτελέσματα του `hardtest.php` είναι:

Πίνακας 40. Αποτελέσματα για `hardtest.php` σε περιβάλλον Linux/Lighttpd

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	269 - 344 ms
20	476 - 686 ms
50	1080 - 1740 ms
100	2307 - 3472 ms
200	4271 - 6785 ms
500	634 - 15855 ms
1000	1583 - 8586 ms

Η αυξημένη επεξεργαστική ισχύ που χρειάστηκε, δεν επηρέασε με σφάλματα τις μετρήσεις μέχρι του 200 χρήστες. Από εκεί και πέρα, τα σφάλματα στους 500 και 1000 χρήστες κυμάνθηκαν μεταξύ 60% και 93 %. Στο Jmeter οφείλονται και εδώ οι αποκλίσεις που παρουσιάστηκαν. Τέλος, σε 60 δευτερόλεπτα στάλθηκαν ~1800 αιτήματα και σε 120 ~3500.

b. Perl

Η Perl, όπως και στους άλλους web servers, θέλει τις ανάλογες ρυθμίσεις για να λειτουργήσει στον Lighttpd. Οι μετρήσεις έφτασαν μέχρι του 1000 χρήστες, ενώ σφάλματα εμφανίστηκαν ακόμα και στους 10 χρήστες (μόνο με τις μετρήσεις του Jmeter και στο αρχείο `simpletest.pl`). Επίσης, το Pylot εμφάνισε προβλήματα με τις

Μέτρηση απόδοσης Web Server

μετρήσεις της Perl, καθώς δεν μπόρεσε να καταγράψει οποιοδήποτε αποτέλεσμα. Αναλυτικότερα για το simpletest.pl προέκυψαν τα παρακάτω:

Πίνακας 41. Αποτελέσματα για simpletest.pl σε περιβάλλον Linux/Lighttpd

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	60 - 215 ms
20	148 - 549 ms
50	286 - 488 ms
100	684 - 760 ms
200	851 - 1449 ms
500	3703 - 5001 ms
1000	15050 - 31515 ms

Τα σφάλματα έπαιξαν το ρόλο τους εδώ, δίνοντας αποτελέσματα τα οποία παρουσίαζαν μεγάλες διαφορές μεταξύ των μετρήσεων. Ως μόνα αξιόπιστα μπορούν να θεωρηθούν τα αποτελέσματα του ab tool.

Για hardtest.pl τα αποτελέσματα είναι:

Πίνακας 42. Αποτελέσματα για hardtest.pl σε περιβάλλον Linux/Lighttpd

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	293 - 604 ms
20	351 - 1156 ms
50	1726 - 2936 ms
100	2510 - 6068 ms
200	7746 - 12559 ms

Μέτρηση απόδοσης Web Server

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
500	11037 - 11172 ms
1000	70612 - 78724 ms

Σε αυτά τα αποτελέσματα η Perl εμφάνισε πολύ μικρά σφάλματα (!!!) στους 1000 χρήστες, αποδεικνύοντας για άλλη μία φορά την αστάθειά της. Επίσης οι αποκλίσεις της έχουν να κάνουν για άλλη μία φορά με το εργαλείο μέτρησης που χρησιμοποιήθηκε.

II. Μετρήσεις σε Windows

a. PHP

Στις μετρήσεις των PHP αρχείων, οι χρήστες που κατάφεραν να εξυπηρετηθούν ταυτόχρονα ήταν μόλις 200. Σφάλματα εμφανίστηκαν μόνο στην περίπτωση της μέτρησης των 200 χρηστών στο αρχείο `hardtest.php`, που στην μία των περιπτώσεων ήταν 100%. Να σημειωθεί επίσης ότι αρκετές φορές η υπηρεσία της PHP «κόλλησε», προκαλώντας και τη μη λειτουργία του web server. Αναλυτικότερα για το `simpletest.php` τα αποτελέσματα είναι:

Πίνακας 43. Αποτελέσματα για `simpletest.php` σε περιβάλλον Windows/Lighttpd

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	1269 - 1282 ms
20	2570 ms
50	5837 - 6520 ms
100	13031 - 13074 ms
200	23383 - 26017 ms

Από τα παραπάνω το πρώτο που διαπιστώνουμε είναι η αργή λειτουργία του web server, καθώς μόνο για 10 χρήστες χρειάζεται μέσο χρόνο εξυπηρέτησης

Μέτρηση απόδοσης Web Server

κάτι παραπάνω από 1 δευτερόλεπτο. Επίσης και ο συνολικός αριθμός των χρηστών που εξυπηρετήσε δεν μπορεί να θεωρηθεί ικανοποιητικός. Σε 60 δευτερόλεπτα εξυπηρετήθηκαν μόλις ~420 αιτήματα και σε 120 ~800.

Ακολουθούν τα αποτελέσματα του `mediumtest.php`

Πίνακας 44. Αποτελέσματα για `mediumtest.php` σε περιβάλλον Windows/Lighttpd

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	1249 - 1331 ms
20	2662 - 2672 ms
50	6085 - 6830 ms
100	13520 - 13720 ms
200	25675 - 27238 ms

Εδώ ισχύει ότι ακριβώς και στην περίπτωση του `simplestest.php`, με τα αποτελέσματα να εμφανίζουν τις ίδιες ιδιαιτερότητες. Επίσης και το Jmeter είχε δυσκολία στη μέτρηση εδώ, καθώς τα αποτελέσματα των 20 και 100 χρηστών παρουσίασαν μεγαλύτερους μέσους όρους από τα αποτελέσματα των 50 και 200 χρηστών αντίστοιχα. Σε 60 δευτερόλεπτα εξυπηρετήθηκαν ~450 αιτήματα και σε 120 ~850.

Τέλος τα αποτελέσματα του `hardtest.php` είναι:

Πίνακας 45. Αποτελέσματα για `hardtest.php` σε περιβάλλον Windows/Lighttpd

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	1610 - 1740 ms
20	3367 - 3517 ms
50	8784 - 9717 ms
100	17820 - 18576 ms

Μέτρηση απόδοσης Web Server

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
200	35339 - 45062 ms

Και εδώ η μέτρηση δεν διέφερε σε συμπεριφορά από τις προηγούμενες που προηγήθηκαν. Μόνη διαφορά είναι τα σφάλματα που εμφανίστηκαν στους 200 χρήστες. Τέλος, σε 60 δευτερόλεπτα στάλθηκαν ~350 αιτήματα και σε 120 ~620.

b. Perl

Η ρύθμιση της Perl ήταν παρόμοια με της προηγούμενες περιπτώσεις. Όπως και με την PHP, έτσι και εδώ αρκετές φορές η υπηρεσία της Perl σταμάτησε να λειτουργεί. Οι μετρήσεις έφτασαν μέχρι τους 200 χρήστες για την περίπτωση του simpletest.pl και τους 100 για το complextest.pl. Σφάλματα εμφανίστηκαν στη μέτρηση του complextest.pl από τους 20 χρήστες. Αναλυτικότερα για το simpletest.pl προέκυψαν τα παρακάτω:

Πίνακας 46. Αποτελέσματα για simpletest.pl σε περιβάλλον Windows/Lighttpd

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	457 - 904 ms
20	918 - 923 ms
50	2347 - 5172 ms
100	4784 - 4815 ms
200	9655 - 9883 ms

Για hardtest.pl τα αποτελέσματα είναι:

Πίνακας 47. Αποτελέσματα για hardtest.pl σε περιβάλλον Linux/Lighttpd

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	1492 - 2564 ms
20	2995 - 3545 ms
50	41074 - 97593 ms
100	455210 – 517713 ms

Τα αποτελέσματα και στις δυο περιπτώσεις παρουσίασαν ενδιαφέρον ως προς τη μέτρηση τους, καθώς στη μεν μέτρηση του simpletest.pl το jmeter εμφάνισε συμπεριφορά παρόμοια με αυτή της PHP, στη δε μέτρηση του complextest.pl το ab tool ήταν αυτό που δεν κατάφερε να μετρήσει πάνω από τους 20 χρήστες. Επίσης σε αντίθεση με την μέτρηση σε περιβάλλον Linux, το Pylot εδώ κατάφερε να βγάλει αποτελέσματα από τα οποία προέκυψε ότι σε 60 δευτερόλεπτα και για το simpletest.pl τα αιτήματα που εξυπηρετήθηκαν ήταν ~1300 ενώ για 120 δευτερόλεπτα ήταν ~2500. Για το complextest.pl οι αντίστοιχες τιμές ήταν ~350 και ~600.

6.3 Συμπεράσματα

I. Εγκατάσταση / Παραμετροποίηση

Η εγκατάσταση του Lighttpd έγινε χωρίς προβλήματα και το μόνο που χρειαζόταν ήταν η κατάλληλη ρύθμισή του ώστε να λειτουργήσουν οι PHP, Perl και ASP.Net.

II. Σύγκριση μεταξύ των περιβαλλόντων Linux και Windows

Από τα αποτελέσματα είναι σαφές ότι ο Lighttpd λειτουργεί καλύτερα σε περιβάλλον Linux. Στις στατικές ιστοσελίδες και σε μικρές ομάδες χρηστών οι διαφορές ήταν μικρές. Όμως στην αύξηση των χρηστών, οι εξυπηρέτηση από περιβάλλον windows γίνεται όλο και πιο αργή. Στις δυναμικές ιστοσελίδες, οι διαφορές από την αρχή είναι μεγάλες (η υλοποίηση σε περιβάλλον Linux κερδίζει κατά κράτος).

Ένα άλλο ενδιαφέρον αποτέλεσμα που προέκυψε είναι ο αριθμός των χρηστών που μπορεί να εξυπηρετηθεί από κάθε περιβάλλον. Σε όλες τις περιπτώσεις ο Lighttpd μέσω windows δεν κατάφερε να φτάσει τα επίπεδα της έκδοσης για Linux. Μόνο στις μετρήσεις του simpletest.html έφτασε των ίδιο αριθμό χρηστών.

Όσον αφορά τις δυναμικές ιστοσελίδες, εκτός από την ASP.Net που αποδείχθηκε προβληματική και στα 2 περιβάλλοντα, η PHP και η Perl αποδείχτηκαν πιο σταθερές και λειτουργικές στο Linux. Αντίθετα σε Windows, εκτός από πιο αργές, αντιμετώπιζαν συχνά προβλήματα που είχαν σαν αποτέλεσμα την κατάρρευση της υπηρεσίας του Lighttpd, μην μπορώντας να εξυπηρετήσει οποιοδήποτε αίτημα (ακόμα και στατικών σελίδων). Γενικά, η χρήση του Lighttpd σε Windows δεν είναι κάτι το οποίο προτείνεται.

Κεφάλαιο 7 - Συγκρίσεις - Ανάλυση αποτελεσμάτων - Συμπεράσματα

Σε αυτό το κεφάλαιο θα επιχειρηθεί η σύγκριση των 4 web servers που εξετάστηκαν προηγουμένως. Διευκρινίζεται ότι κανένας δεν μπορεί να θεωρηθεί ως ο τέλειος web server, καθώς όλοι παρουσιάζουν θετική και αρνητική συμπεριφορά. Επομένως εδώ θα αναφερθούμε στα γενικά συμπεράσματα που καταλήξαμε από τις μετρήσεις.

7.1 Εργαλεία μέτρησης

Πριν προχωρήσουμε στα αποτελέσματα, θα γίνει μία αναφορά για τα 3 εργαλεία μέτρησης που χρησιμοποιήθηκαν, το AB Tool, το Jmeter και το Pylot. Όπως και για τους web servers, έτσι και εδώ δεν μπορούμε να πούμε ότι υπάρχει ένα τέλειο εργαλείο. Το καθένα παρουσιάζει μία ξεχωριστή συμπεριφορά.

Ξεκινώντας από το AB tool, στα θετικά του μπορούμε να καταλογίσουμε των τρόπο εμφάνισης των αποτελεσμάτων, με κατανοητή παρουσίαση αν συμπεριλάβουμε και το γεγονός ότι πρόκειται για ένα πρόγραμμα που εκτελείται μέσω κονσόλας. Είναι το μόνο από τα 3 εργαλεία το οποίο εκτός από τα αποτελέσματα, εμφάνιζε και τον τύπο του web server στον οποίο κάναμε της μετρήσεις. Αντίθετα, το πρόβλημά του εστιαζόταν κυρίως στην καταγραφή των αποτελεσμάτων σε .csv αρχεία, καθώς οι πληροφορίες που κατέγραφε ήταν ουσιαστικά μη χρήσιμες, ενώ ο τρόπος καταγραφής τους δεν ήταν και ο πιο κατανοητός.

Για το Jmeter, το πρώτο που το κάνει να διαφέρει είναι η χρήση του γραφικού περιβάλλοντος. Δίνει τη δυνατότητα δημιουργίας διαφόρων σεναρίων για την πραγματοποίηση των μετρήσεων, όπως επίσης και διαφορετικούς τρόπους καταγραφής των αποτελεσμάτων. Τα αποτελέσματα μπορούν να καταγραφούν και σε εξωτερικά αρχεία .csv ή εικόνων αν πρόκειται για γραφήματα. Τα γραφήματά του, όμως υπήρξαν αρκετά προβληματικά και ασαφή χωρίς δυνατότητα χρησιμοποίησής τους. Μεγάλο πρόβλημα ήταν επίσης και η αστάθεια του ίδιου του προγράμματος, με συχνά «κολλήματα» και απροειδοποίητες στιγμές όπου το

πρόγραμμα έκλεινε μόνο του χωρίς πριν να έχει αποθηκευτεί κάποιο αποτέλεσμα η σενάριο.

Το Pylot, ουσιαστικά είναι ένα νέο εργαλείο, χωρίς την εμπειρία των άλλων 2, γραμμένο σε μία σχετικά καινούργια γλώσσα και με λίγο διαφορετική φιλοσοφία. Από τις δοκιμές όμως έγινε φανερό ότι θέλει αρκετή βελτίωση ακόμα, καθώς φάνηκε προβληματικό στην καταγραφή των αποτελεσμάτων. Αν και τα αποτελέσματα, μπορούν να χαρακτηριστούν πλήρη (από την άποψη ότι καταγράφονται), η ίδια η καταγραφή εμφανίζει πρόβλημα. Για παράδειγμα, το πιο συχνό πρόβλημα ήταν η μη σωστή συνολική καταγραφή των αιτημάτων που εξυπηρετήθηκαν, με μόνη λύση την χειροκίνητη μέτρηση τους από τον αναλυτικό πίνακα με το πόσο αιτήματα εξυπηρετήθηκαν για κάθε χρήστη.

Βγάζοντας, επομένως ένα συνολικό συμπέρασμα, βλέπουμε ότι το κάθε εργαλείο έχει τις δικές του ιδιαιτερότητες. Η πιο σημαντική βέβαια είναι οι ίδιες οι μετρήσεις. Και αναφερόμαστε σε αυτό γιατί ίδιες μετρήσεις, με διαφορετικά εργαλεία έδωσαν διαφορετικά αποτελέσματα. Επομένως, ίσως να θεωρείται προτιμότερη η επιλογή και χρησιμοποίηση ενός και μόνο εργαλείου. Για να γίνει πιο κατανοητό αυτό, θα μπορούσαμε να αναφερθούμε σε ένα παράδειγμα από την πραγματική ζωή, όπου κάθε άνθρωπος είναι προτιμότερο να μετράει το βάρος του σε μία συγκεκριμένη ζυγαριά. Αν χρησιμοποιεί διαφορετικές, το πιθανότερο είναι να πάρει και διαφορετικά αποτελέσματα, καθώς κάθε κατασκευαστής έχει διαφορετικό τρόπο κατασκευής της ζυγαριάς. Κάτι ανάλογο συμβαίνει και στην περίπτωση των εργαλείων μέτρησης.

7.2 Συγκρίσεις – Ανάλυση αποτελεσμάτων - Συμπεράσματα

I. Εγκατάσταση / Παραμετροποίηση

Η εγκατάσταση των web servers δεν εμφάνισε καμία ιδιαίτερη δυσκολία. Εκεί που παρουσιάστηκε μεγαλύτερος βαθμός δυσκολίας είναι στις ρυθμίσεις τους. Στην περίπτωση του Apache, τα πράγματα ήταν απλά καθώς το διαδίκτυο είναι κατακλεισμένο από οδηγίες για το πώς θα μπορέσει να συνεργαστεί με τις γλώσσες διαδικτυακού προγραμματισμού. Από εκεί και πέρα, ο IIS κερδίζει έδαφος στο ότι παρέχει γραφικό περιβάλλον, μη μπλέκοντας με ρυθμίσεις ξεχωριστών αρχείων. Παρόλα αυτά, ρύθμιση μιας γλώσσας εκτός από την ενσωματωμένη ASP.Net δεν είναι και τόσο εύκολη. Για τους Nginx και Lighttpd ισχύουν πάνω-κάτω τα ίδια σε αυτό τον τομέα. Στα συνολικά μείον τώρα, μπορούμε να αναφέρουμε την γενική μη επίσημη υποστήριξη της ASP.Net (εκτός φυσικά του IIS) και τις διαφορές που υπήρξαν ανάλογα με το λειτουργικό σύστημα (μη ύπαρξη έκδοσης του IIS για Linux, προβληματική λειτουργία του Nginx σε Windows).

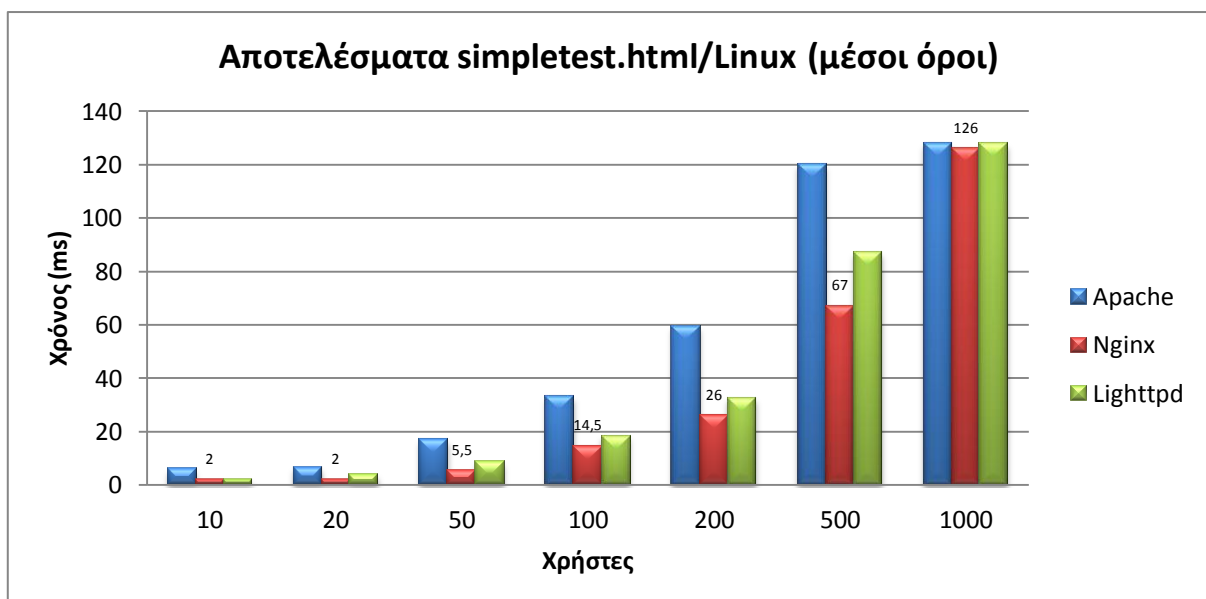
II. Ανάλυση των αποτελεσμάτων

Στην συνέχεια ακολουθούν τα συγκριτικά αποτελέσματα από τις μετρήσεις.

- **Simpletest.html**

- Μέσοι όροι

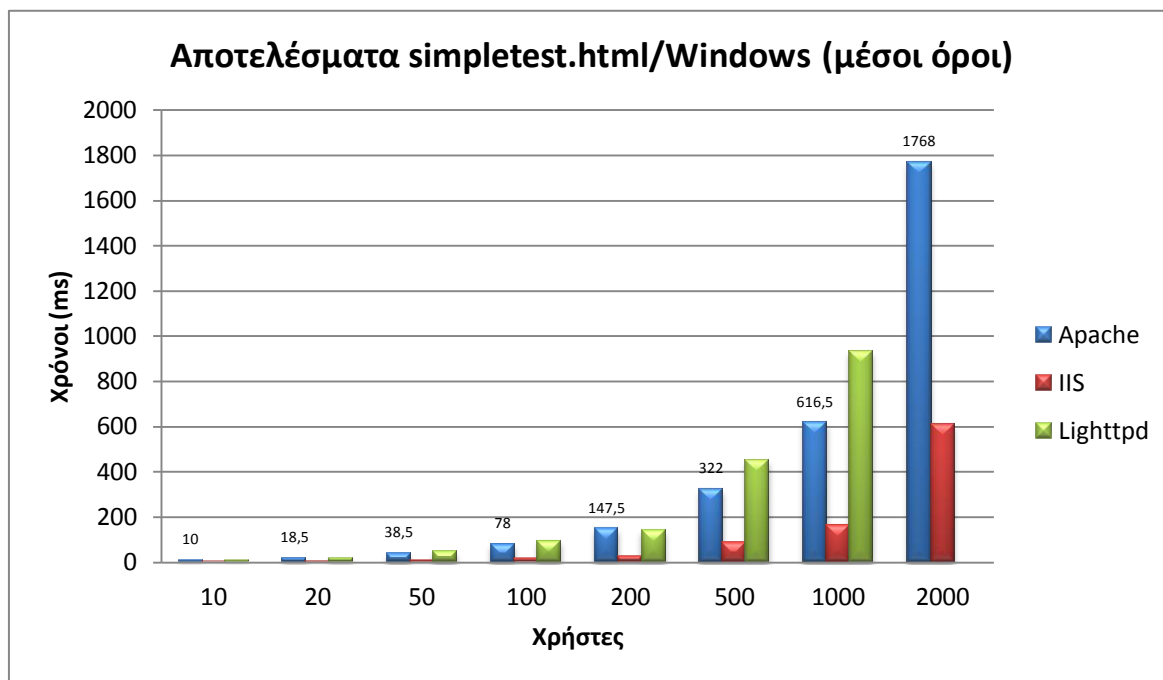
Ξεκινάμε με τις μετρήσεις σε περιβάλλον Linux.



Μέτρηση απόδοσης Web Server

Όπως βλέπουμε στο διάγραμμα, οι διαφορές μεταξύ των web servers είναι μικρές. Προβάδισμα εμφανίζει να έχει ο Nginx, με τον Lighttpd να ακολουθεί και τελευταίο τον Apache. Πάντως στους 1000 χρήστες οι διαφορές εξισορροπήθηκαν.

Ακολουθεί η αντίστοιχη μέτρηση σε Windows:



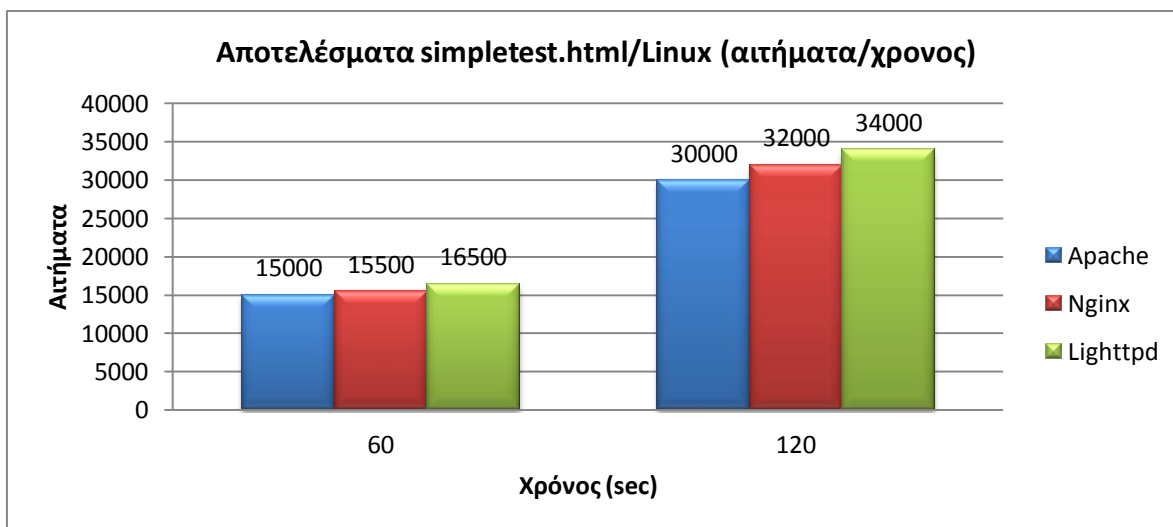
Σε περιβάλλον Windows βλέπουμε ότι ο IIS έχει το προβάδισμα έναντι των άλλων 2 από τις πρώτες μετρήσεις, με την ψαλίδα να ανοίγει σταδιακά. Apache και Lighttpd κινήθηκαν στα ίδια επίπεδα μέχρι τους 200 χρήστες, με τον δεύτερο να αποδίδει λιγότερο από εκεί και έπειτα. Επίσης IIS και Apache εξυπηρέτησαν διπλάσια αιτήματα από τον Lighttpd.

Συγκρίνοντας τώρα τις μετρήσεις και στα δύο λειτουργικά συστήματα, παρατηρούμε ότι οι επιδόσεις των υλοποιήσεων των web server σε περιβάλλον Linux και του IIS κινούνται περίπου στα ίδια επίπεδα. Αντίθετα, Apache και Lighttpd αποδίδουν μεγαλύτερους μέσους όρους σε Windows. Ο Lighttpd κατάφερε να εξυπηρετήσει λιγότερους χρήστες σε Windows, ενώ ο Apache τους διπλάσιους σε σχέση με την υλοποίηση σε Linux.

Μέτρηση απόδοσης Web Server

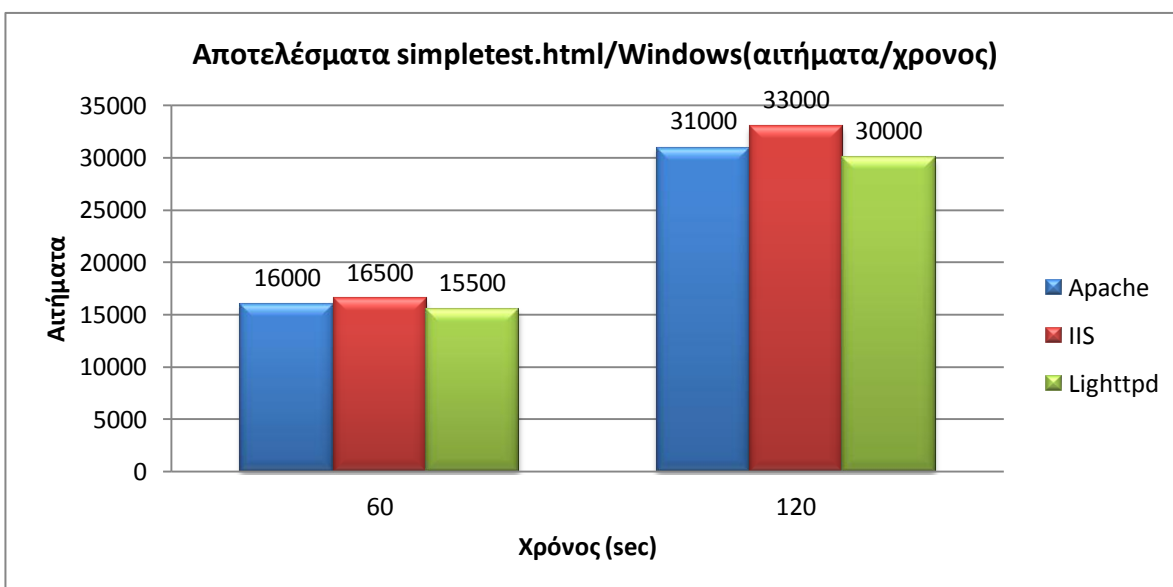
- Εξυπηρέτηση αιτημάτων σε 60 και 120 δευτερόλεπτα

Ακολουθούν οι μετρήσεις που έγιναν σε περιβάλλον Linux



Από το διάγραμμα είναι φανερό ότι ο Apache υστέρησε, με τους Nginx και Lighttpd να έχουν μικρή διαφορά μεταξύ τους και το δεύτερο να κατέχει το προβάδισμα.

Η μέτρηση σε Windows κατέγραψε τα παρακάτω:



Εδώ βλέπουμε ότι ο IIS έχει ένα μικρό προβάδισμα, με τον Apache να ακολουθεί και ο Lighttpd έρχεται 3^{ος} με μικρή διαφορά.

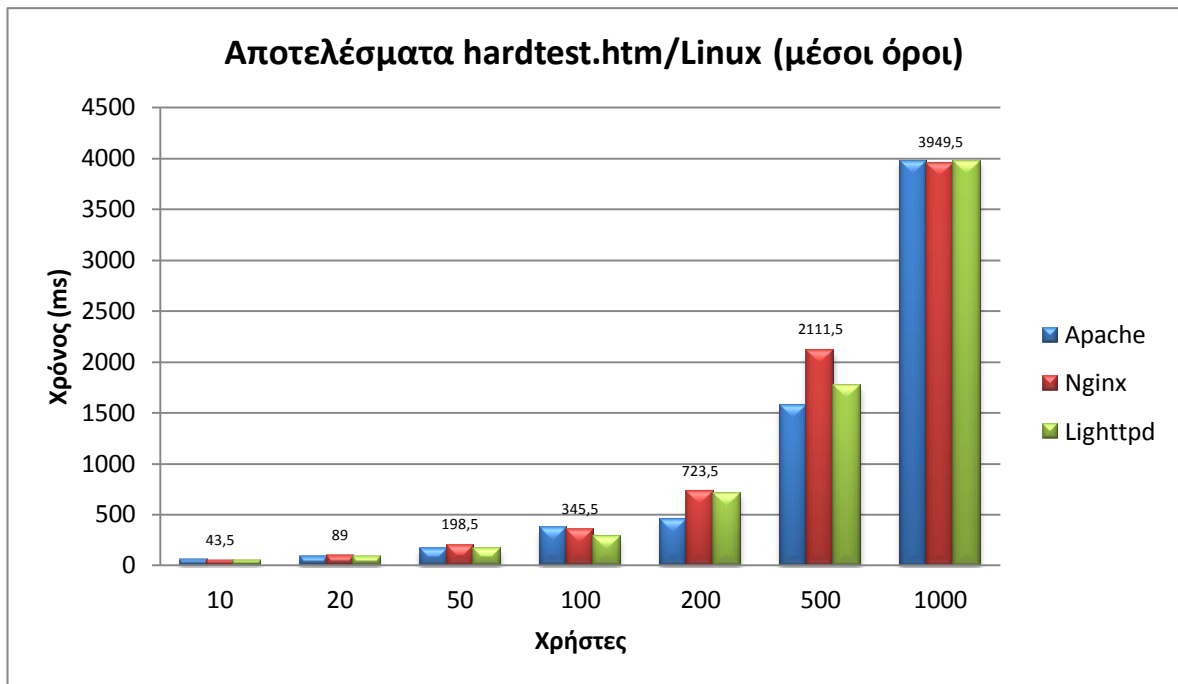
Μέτρηση απόδοσης Web Server

Βλέποντας τώρα συνολικά τα αποτελέσματα, δεν υπήρξαν τεράστιες διαφοροποιήσεις μεταξύ των web server.

- **Hardtest.htm**

- Μέσοι όροι

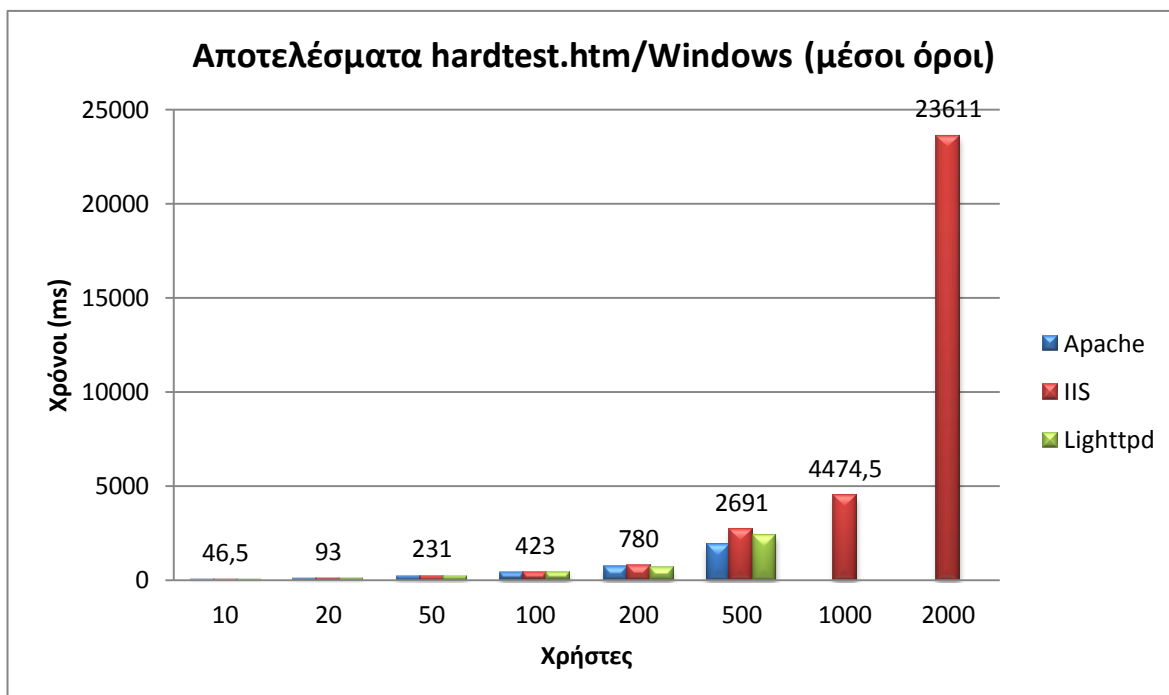
Οι μετρήσεις σε περιβάλλον Linux ήταν οι ακόλουθες:



Όπως βλέπουμε στο διάγραμμα, είχαμε διάφορες αυξομειώσεις στις μετρήσεις μέχρι τους 100 χρήστες. Στους 200 και 500 χρήστες γρηγορότερος ήταν ο Apache, ενώ στους 1000 η κατάσταση εξισορροπήθηκε.

Μέτρηση απόδοσης Web Server

Ακολουθεί η αντίστοιχη μέτρηση σε Windows:



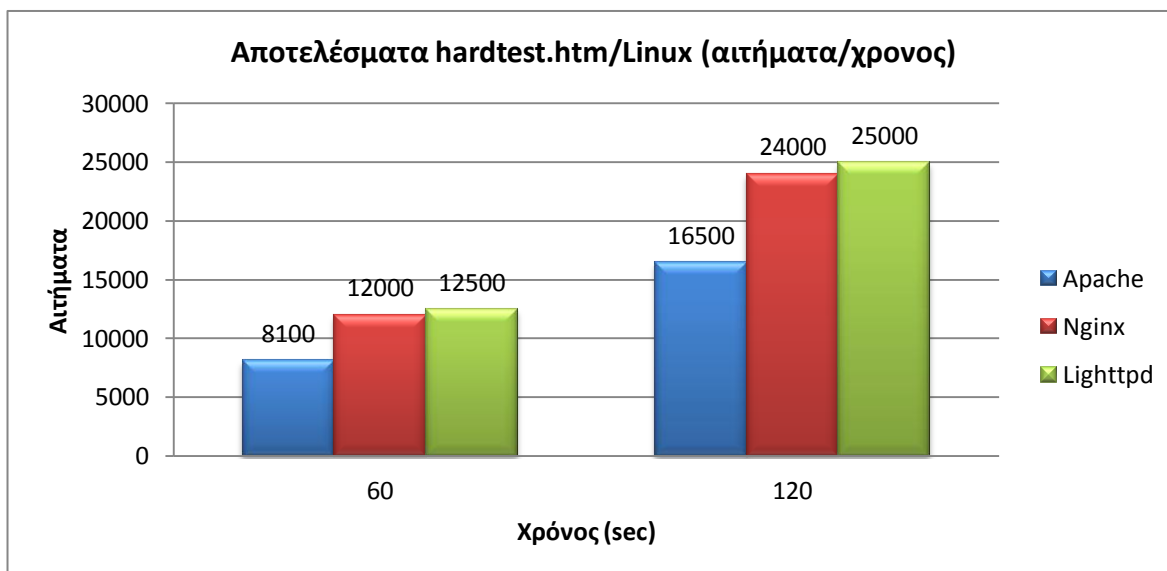
Εδώ μέχρι τους 200 χρήστες, η απόδοση και των 3 web server είναι παρόμοια, ενώ στους 500 έχει ένα ελαφρύ προβάδισμα ο Apache. Πάντως ο τελευταίος μαζί με τον Lighttpd δεν κατάφεραν να ξεπεράσουν τους 500 χρήστες, με τον IIS να εμφανίζεται αποδοτικότερος σε αυτό τον τομέα φτάνοντας τους 2000.

Συγκρίνοντας τώρα τα δύο αποτελέσματα, βλέπουμε ότι από άποψη μέσων όρων, οι διανομές σε Linux απέδωσαν καλύτερα. Από την μεριά των χρηστών που εξυπηρετήθηκαν ο IIS κατάφερε να εξυπηρετήσει περισσότερους, με όλες τις υλοποιήσεις σε Linux να φτάνουν στο μισό αυτής της επίδοσης. Οι υλοποιήσεις σε Windows που μετρήθηκαν, κινήθηκαν χαμηλότερα από αυτές.

Μέτρηση απόδοσης Web Server

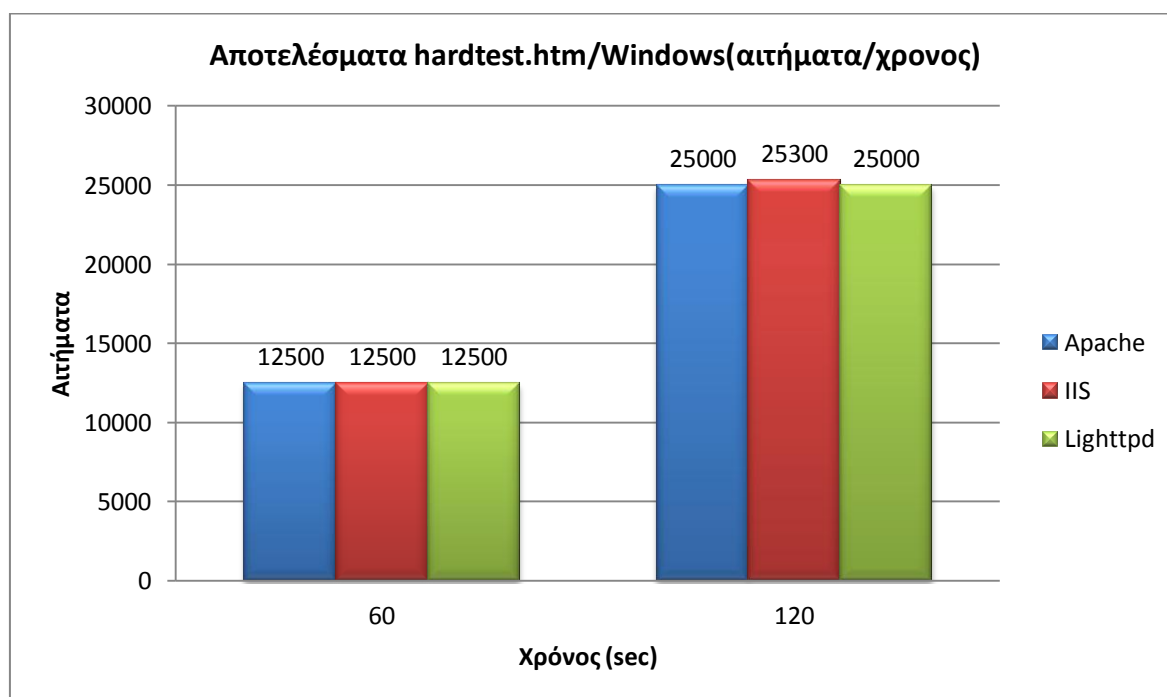
- Εξυπηρέτηση αιτημάτων σε 60 και 120 δευτερόλεπτα

Ακολουθούν οι μετρήσεις που έγιναν σε περιβάλλον Linux



Όπως βλέπουμε, Lighttpd και Nginx απέδωσαν σχετικά το ίδιο, με τον πρώτο να αποδίδει ελαφρώς καλύτερα. Ο Apache αντίθετα έμεινε αρκετά πίσω.

Η μέτρηση σε Windows κατέγραψε τα παρακάτω:



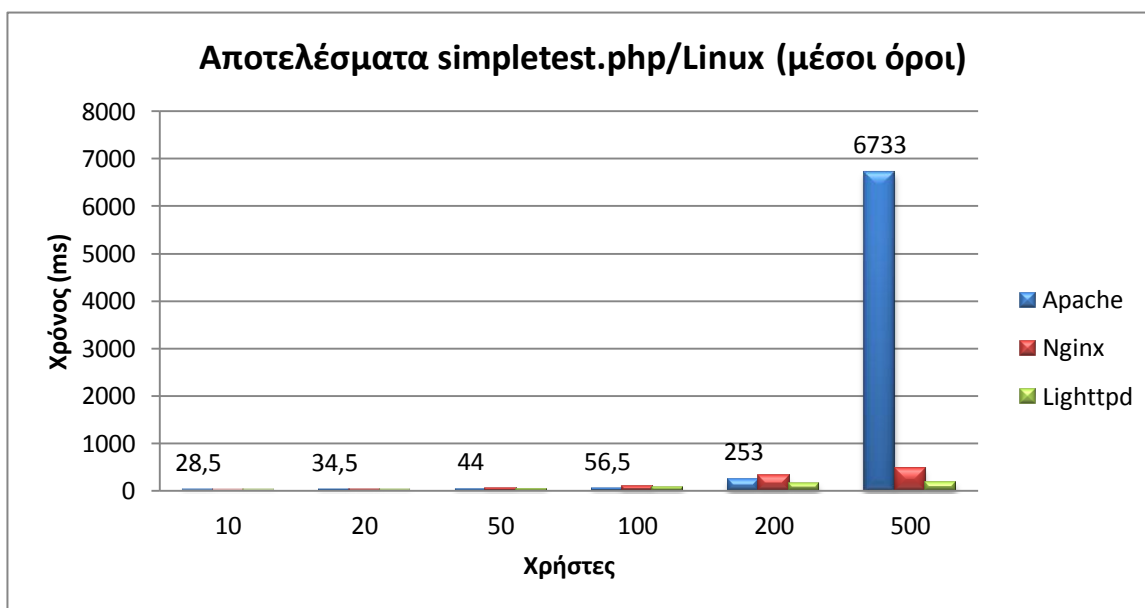
Εδώ βλέπουμε μία σχεδόν πλήρη ισορροπία μεταξύ των 3 web servers, με μόνη (μικρή) υπεροχή του IIS στα 120 δευτερόλεπτα.

Βλέποντας τώρα και τα δύο παραπάνω αποτελέσματα, παρατηρούμε ότι εκτός από την περίπτωση Apache/Linux, σε όλες τις άλλες μετρήσεις δεν υπήρξε κάποιο ισχυρό προβάδισμα από κάποιον web server, με τις διαφορές τους να είναι μικρές ή μηδενικές.

- **Simpletest.php**

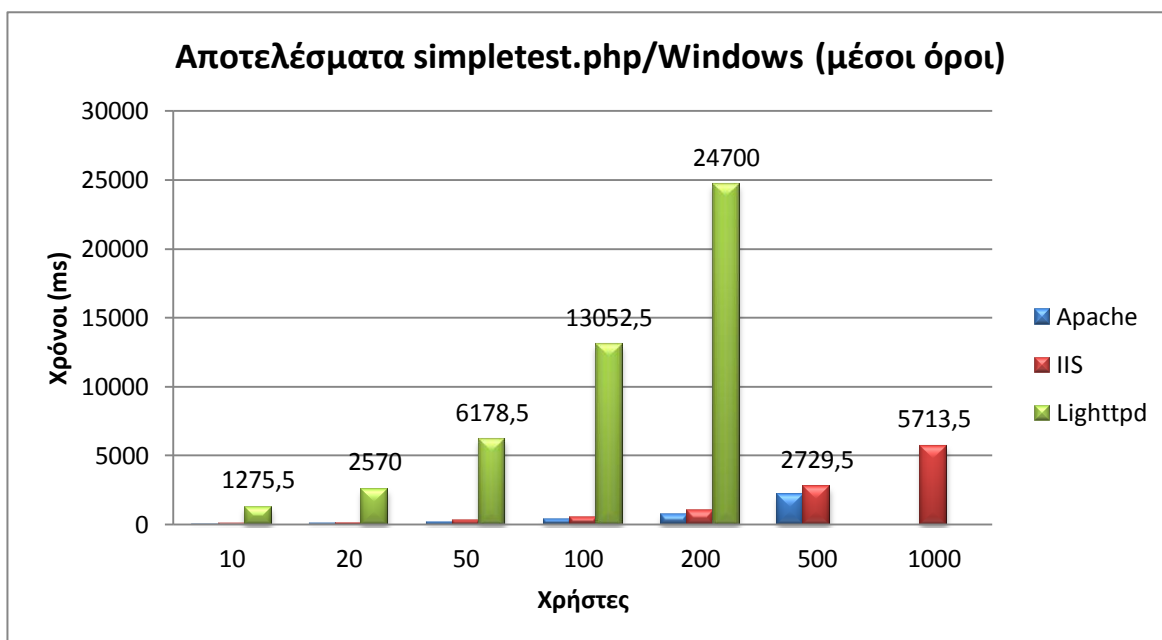
- Μέσοι όροι

Οι μετρήσεις σε περιβάλλον Linux ήταν οι ακόλουθες:



Από το παραπάνω διάγραμμα παρατηρούμε ότι μέχρι τους 200 χρήστες, οι 3 web server είχαν σχεδόν παρόμοια απόδοση με μικρές εναλλαγές μεταξύ τους. Σους 500 χρήστες όμως, ο Apache δεν συνέχισε να λειτουργεί σε αυτό το ρυθμό και έμεινε πάρα πολύ πίσω, ενώ και η απόσταση Lighttpd – Nginx διευρύνθηκε.

Ακολουθεί η αντίστοιχη μέτρηση σε Windows:



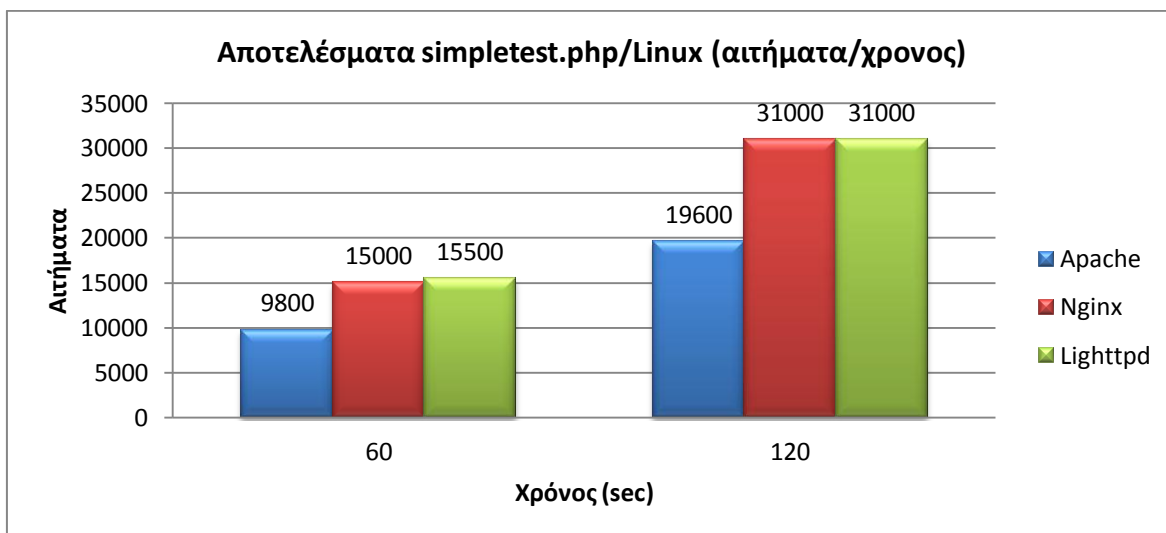
Σε αυτή τη μέτρηση, ανταγωνισμός υπήρξε μεταξύ του Apache και του IIS, με τον Lighttpd να μην μπορεί να αποδώσει όπως αυτοί. Στην μάχη των 2, πιο γρήγορος αποδείχτηκε ο Apache αλλά ο IIS εξυπηρέτησε τους διπλάσιους χρήστες.

Παίρνοντας τώρα και τα δύο αποτελέσματα, βλέπουμε καταρχήν ότι οι χρόνοι εξυπηρέτησης είναι μικρότεροι σε περιβάλλον Linux. Μόνη παραφωνία η περίπτωση του Apache/Linux στη μέτρηση των 500 χρηστών, όπου ο web server απέδωσε πολύ λιγότερο του αναμενόμενου. Αυτό μπορεί να μην οφείλεται και στον ίδιο αλλά στο γεγονός ότι το αρχείο χρησιμοποιεί τη Mysql. Από την άποψη των χρηστών που εξυπηρετήθηκαν, όλοι με εξαίρεση τον IIS δεν κατάφεραν να ξεπεράσουν τους 500 χρήστες.

Μέτρηση απόδοσης Web Server

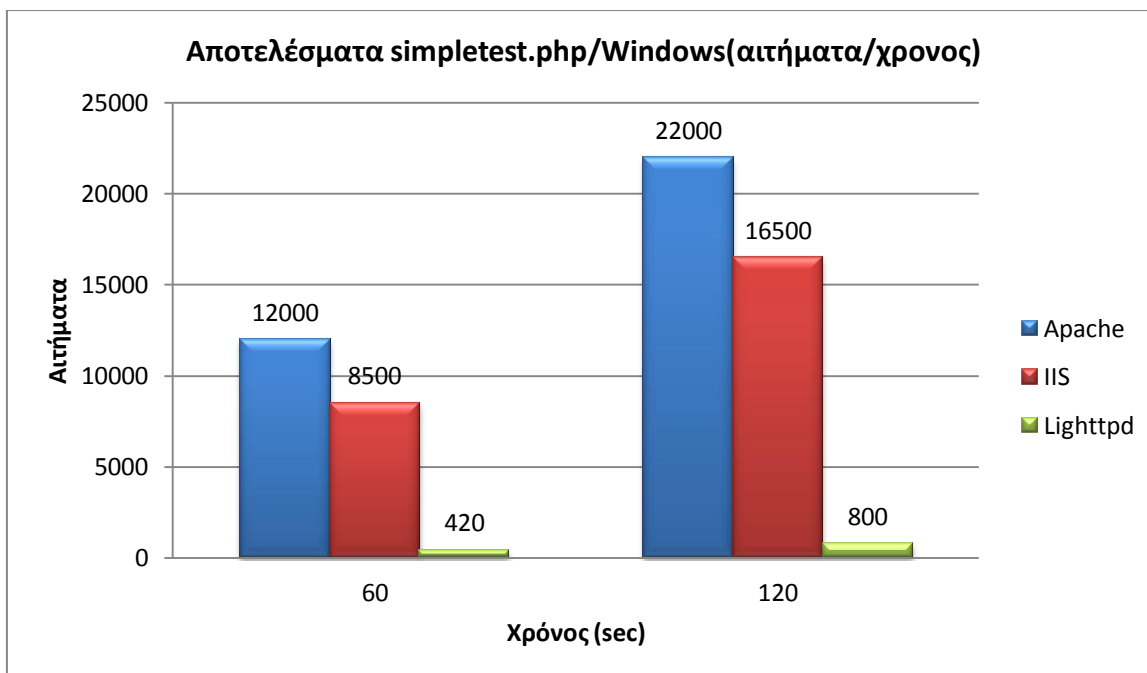
- Εξυπηρέτηση αιτημάτων σε 60 και 120 δευτερόλεπτα

Ακολουθούν οι μετρήσεις που έγιναν σε περιβάλλον Linux



Από το διάγραμμα φαίνεται ότι για άλλη μία φορά Lighttpd και Nginx απέδωσαν σχεδόν το ίδιο, με τον Apache να μένει πίσω.

Η μέτρηση σε Windows κατέγραψε τα παρακάτω:



Μέτρηση απόδοσης Web Server

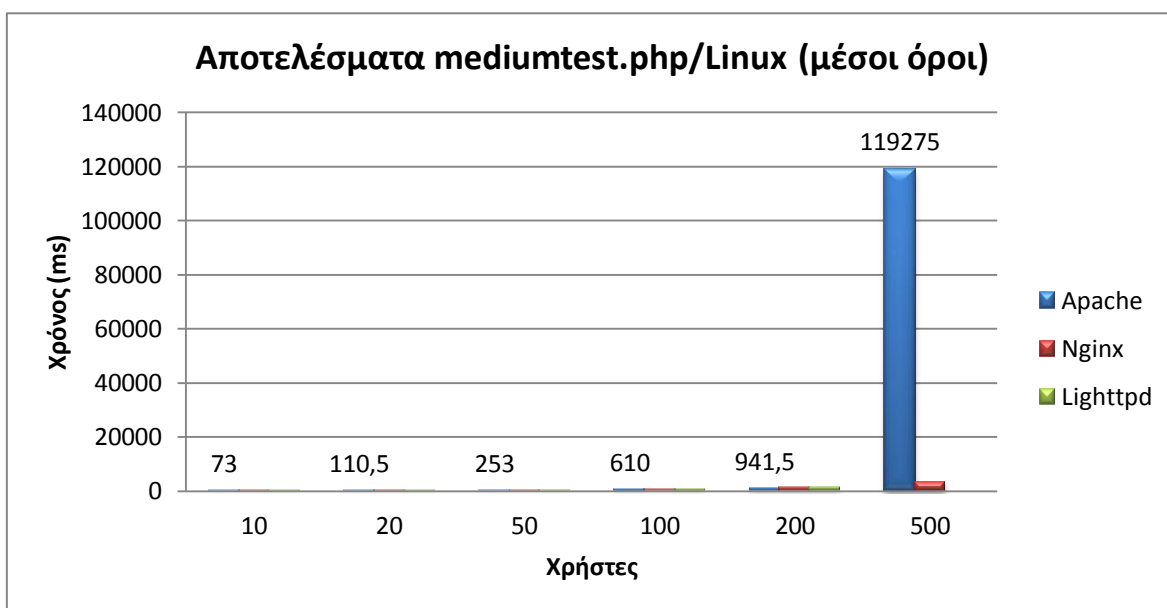
Εδώ βλέπουμε την υπεροχή του Apache έναντι των άλλων 2, και με μία σχετική διαφορά. Ο IIS έρχεται δεύτερος, με τον Lighttpd να έχει πάρα πολύ χαμηλές επιδόσεις.

Από τα 2 παραπάνω διαγράμματα βλέπουμε ότι οι Nginx και Lighttpd(Linux) έχουν προβάδισμα. Ο Apache, εδώ παρουσίασε καλύτερη απόδοση σε περιβάλλον Windows, ενώ ο IIS έμεινε πιο πίσω. Η έκδοση του Lighttpd σε windows φαίνεται αρκετά προβληματική, καθώς κυμαίνεται σε πολύ χαμηλά επίπεδα.

- **mediumtest.php**

- Μέσοι όροι

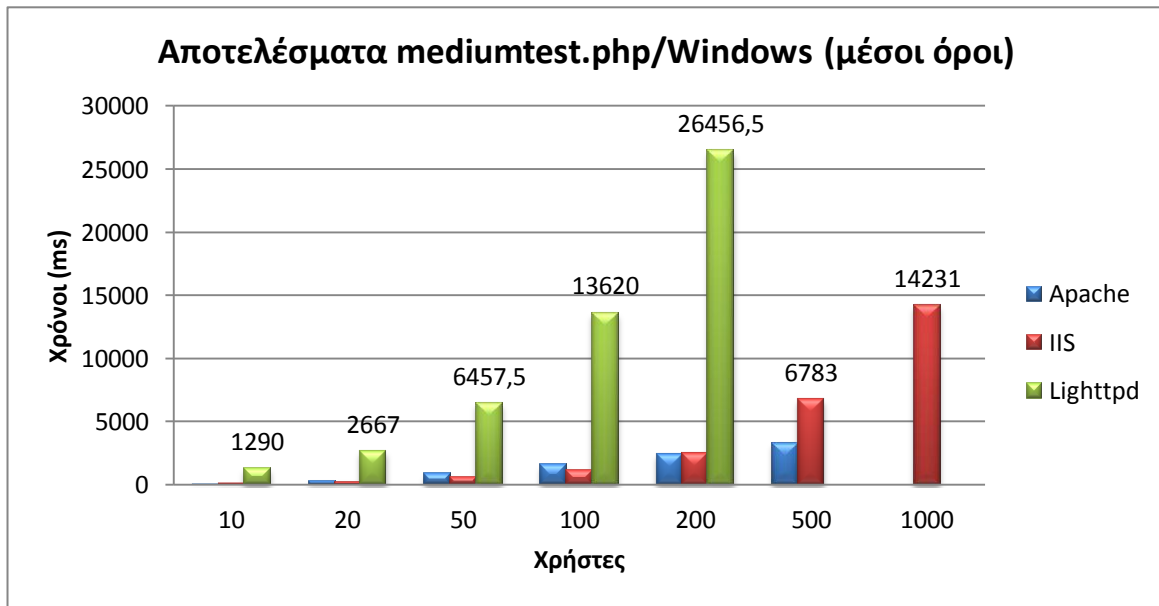
Οι μετρήσεις σε περιβάλλον Linux ήταν οι ακόλουθες:



. Σε αυτή τη μέτρηση, οι εναλλαγές μέχρι τους 200 χρήστες ήταν συνεχείς, χωρίς κάποιος να παίρνει ένα μεγάλο προβάδισμα. Στους 500 όμως ο Apache εμφάνισε συμπεριφορά ανάλογη αυτής της μέτρησης του simpletest.php ενώ για τον Lighttpd δεν ήταν εφικτό να παραχθούν αποτελέσματα.

Μέτρηση απόδοσης Web Server

Ακολουθεί η αντίστοιχη μέτρηση σε Windows:



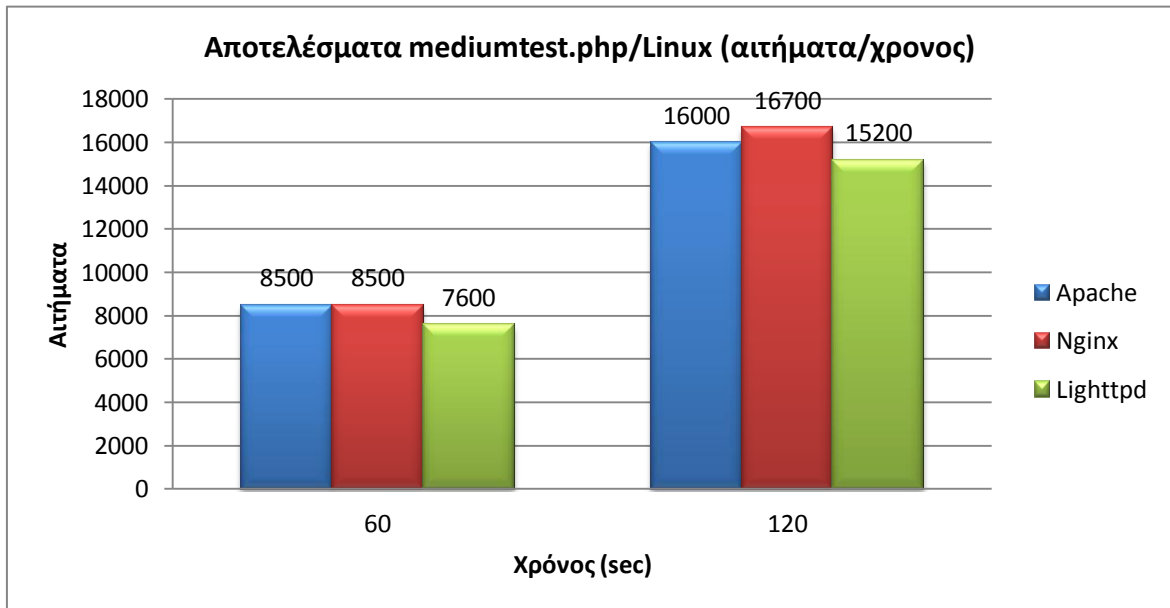
Και σε αυτή την περίπτωση ο Lighttpd έμεινε αρκετά πίσω, με τους Apache και IIS να συναγωνίζονται. Ο IIS μέχρι τους 100 χρήστες είχε ένα ελαφρύ προβάδισμα, αλλά έπειτα οι χρόνοι του Apache ήταν μικρότεροι. Ο IIS πάντως και εδώ εξυπηρέτησε περισσότερους χρήστες.

Συγκρίνοντας τώρα τα δύο αποτελέσματα, οι γενικότερες επιδόσεις των web server είχαν παρόμοια συμπεριφορά με αυτές του simpletest.php.

Μέτρηση απόδοσης Web Server

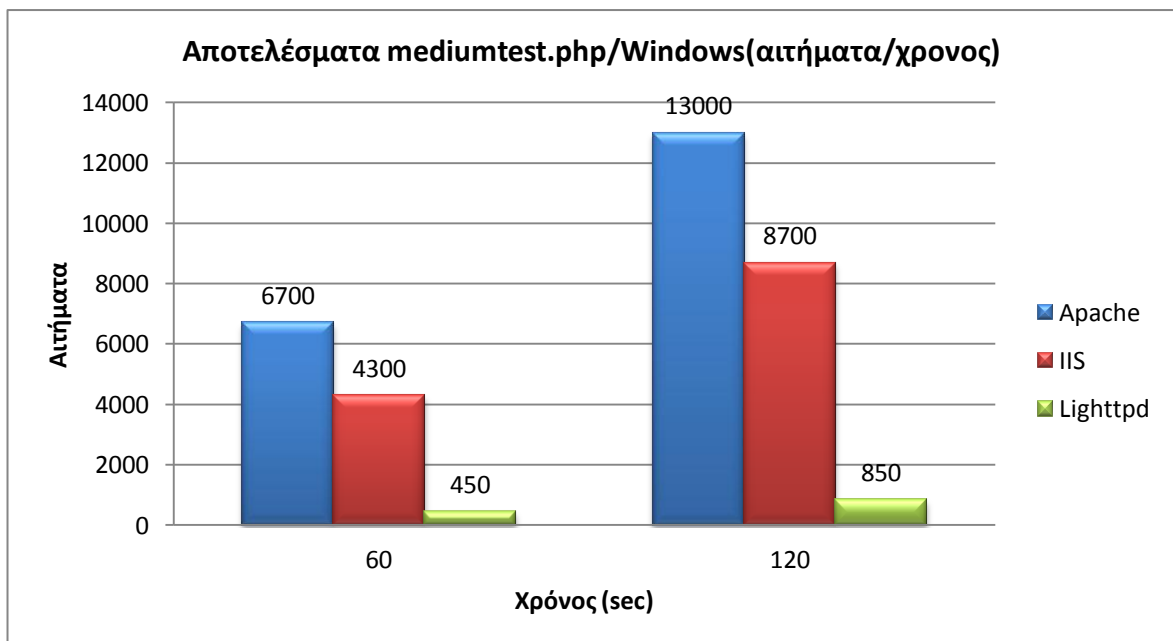
- Εξυπηρέτηση αιτημάτων σε 60 και 120 δευτερόλεπτα

Ακολουθούν οι μετρήσεις που έγιναν σε περιβάλλον Linux



Σε αυτό το διάγραμμα βλέπουμε ότι Nginx και Apache απέδωσαν σχεδόν το ίδιο, με τον πρώτο να αποκτά ένα ελαφρύ προβάδισμα στα 120 δευτερόλεπτα. Ο Lighttpd έμεινε πίσω με μικρή όμως διαφορά.

Η μέτρηση σε Windows κατέγραψε τα παρακάτω:



Μέτρηση απόδοσης Web Server

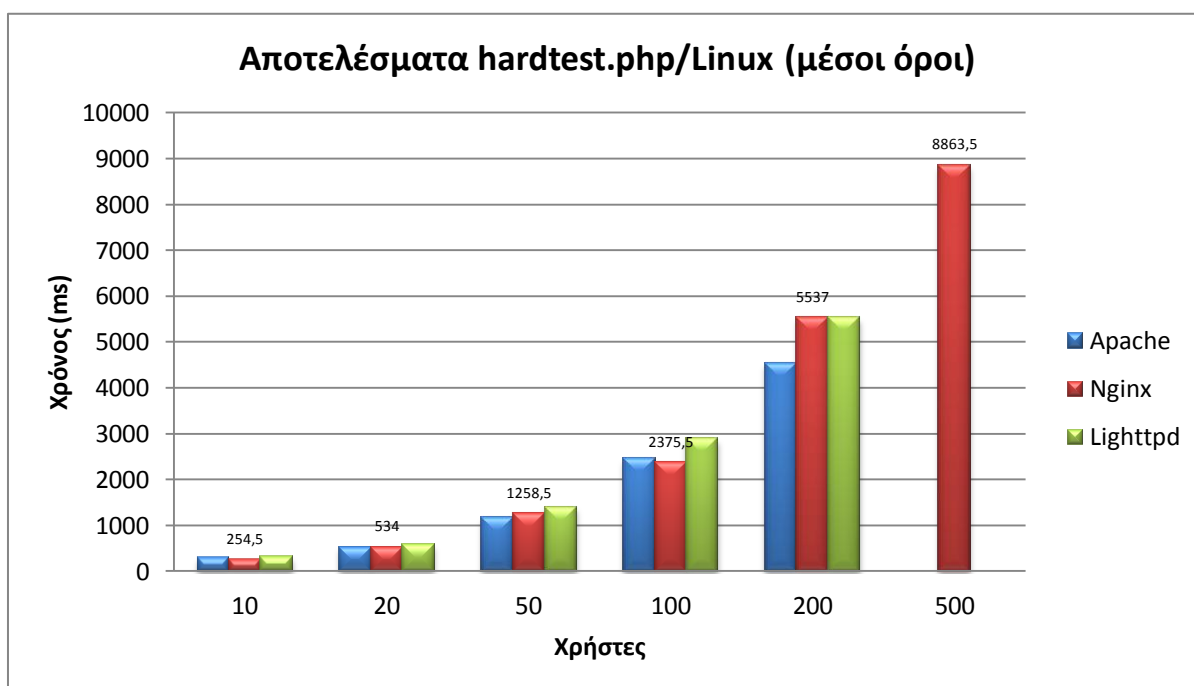
Σε αυτό το διάγραμμα βλέπουμε τη σαφή υπεροχή του Apache έναντι του IIS. Ο Lighttpd, όπως και προηγουμένως απέδωσε απογοητευτικά.

Βλέποντας τώρα και τα δύο διαγράμματα, παρατηρούμε ότι οι υλοποιήσεις σε Linux έχουν την υπεροχή έναντι αυτών των Windows, με τον Nginx να είναι αυτός που είχε το προβάδισμα.

- **hardtest.php**

- Μέσοι όροι

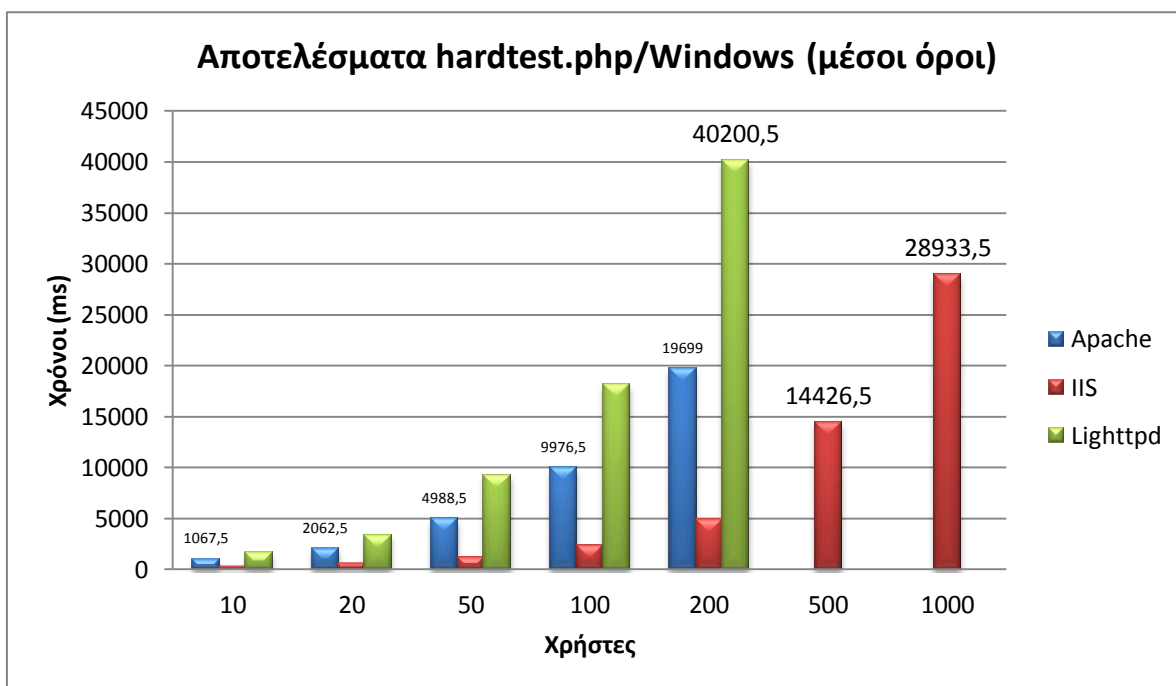
Οι μετρήσεις σε περιβάλλον Linux ήταν οι ακόλουθες:



Όπως και στις προηγούμενες μετρήσεις σε Linux, έτσι και εδώ υπήρξαν εναλλαγές στις μετρήσεις με κανέναν web server να μην καταφέρνει να παίρνει μεγάλο προβάδισμα. Οι μόνες διαφοροποιήσεις ήταν στους 200 χρήστες, όπου ο Apache απέκτησε ένα μεγαλύτερο προβάδισμα, και στους 500 χρήστες όπου μόνο ο Nginx κατάφερε να περατώσει αυτή τη μέτρηση.

Μέτρηση απόδοσης Web Server

Ακολουθεί η αντίστοιχη μέτρηση σε Windows:



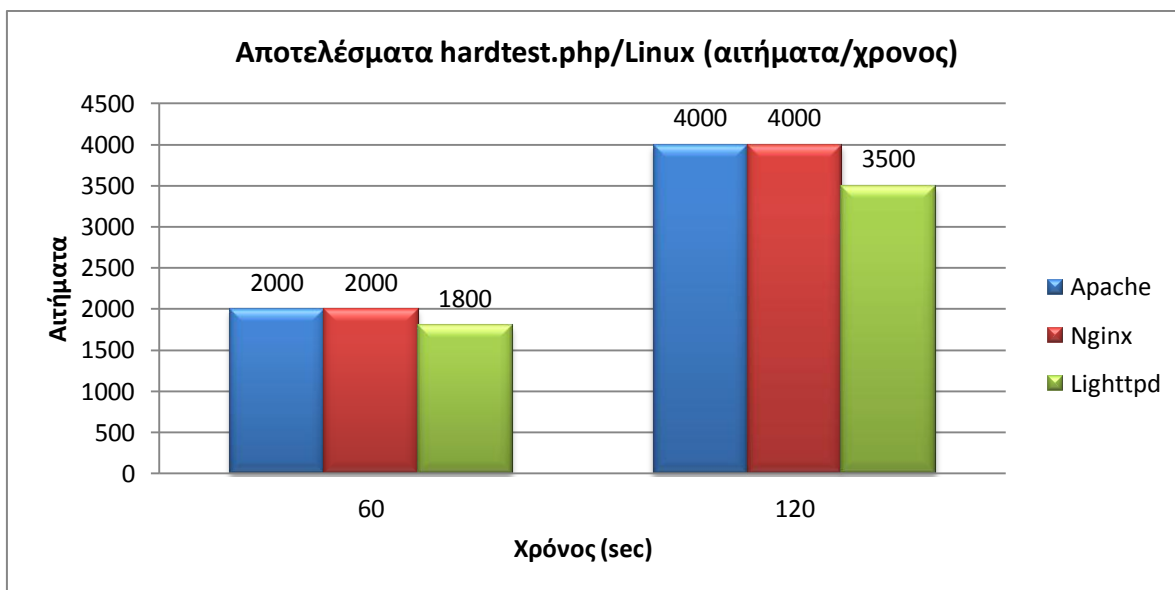
Σε αυτή μέτρηση ο IIS απέδωσε καλύτερα και στις 2 μονάδες μέτρησης. Ο Apache δεν κατάφερε να φτάσει στα επίπεδά του εξυπηρετώντας μόλις 200 χρήστες, ενώ ο Lighttpd έμεινε για άλλη μια φορά αρκετά πίσω.

Παρατηρώντας τα διαγράμματα, βλέπουμε ότι ο IIS κατάφερε σε αυτές τις μετρήσεις να πιάσει σχεδόν τα ίδια επίπεδα απόδοσης με τους web server κάτω από περιβάλλον Linux, δείχνοντας ότι μπορεί να φέρει σε πέρας, με ικανοποιητικό τρόπο, πολύπλοκες υλοποιήσεις αρχείων.

Μέτρηση απόδοσης Web Server

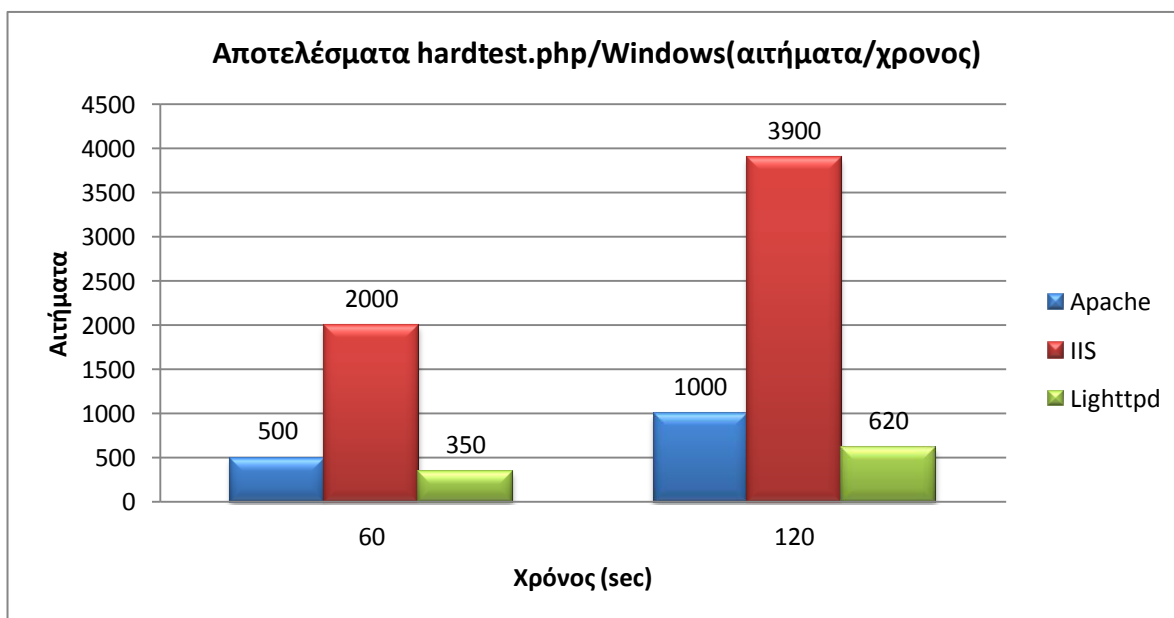
- Εξυπηρέτηση αιτημάτων σε 60 και 120 δευτερόλεπτα

Ακολουθούν οι μετρήσεις που έγιναν σε περιβάλλον Linux



Όπως βλέπουμε, Nginx και Apache παρουσίασαν ακριβώς την ίδια συμπεριφορά. Ο Lighttpd έμεινε λίγο πιο πίσω.

Η μέτρηση σε Windows κατέγραψε τα παρακάτω:



Εδώ μπορούμε να δούμε καθαρά ότι ο IIS κέρδισε κατά κράτος, παρουσιάζοντας 3-πλάσια και 4-πλάσια απόδοση. Ο Apache σε σχέση με τις

Μέτρηση απόδοσης Web Server

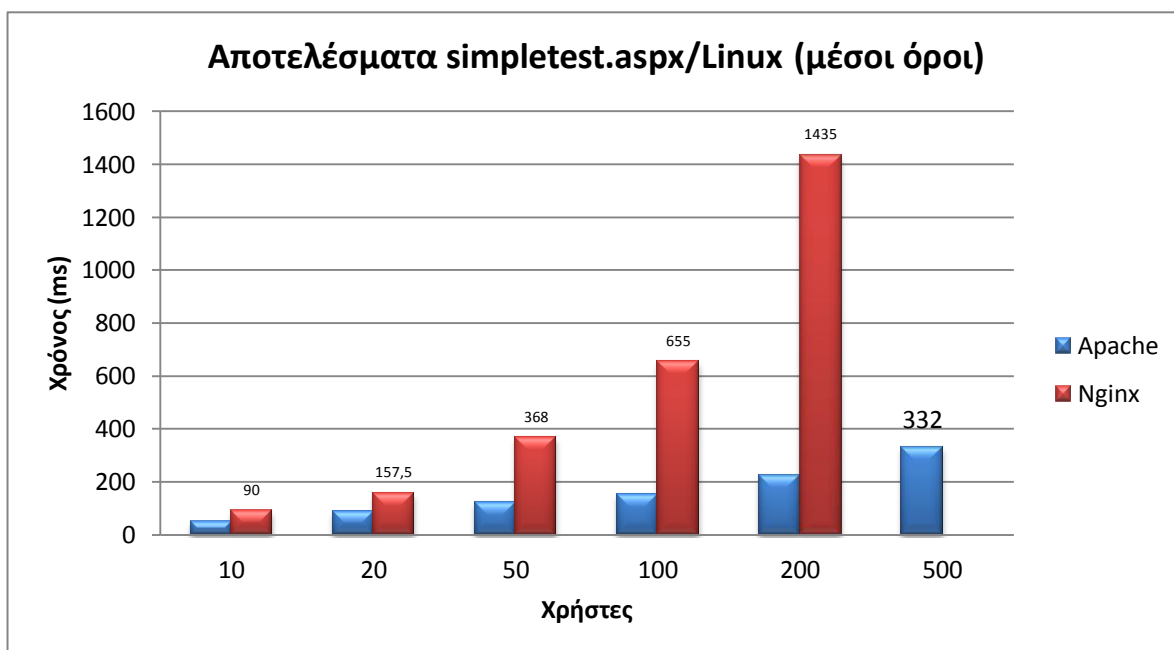
προηγούμενες μετρήσεις αρχείων, απέδωσε πολύ λιγότερο, ενώ ο Lighttpd έμεινε σταθερός στα χαμηλά επίπεδα.

Από τα διαγράμματα βλέπουμε ότι Apache/Linux, Nginx/Linux και IIS παρουσίασαν παρόμοια αποτελέσματα, δείχνοντας ότι μπορούν να αποδώσουν ικανοποιητικά κάτω από συνθήκες που απαιτείται μεγαλύτερη επεξεργαστική ισχύ. Ο Lighttpd/Linux βρέθηκε λίγο πιο πίσω δείχνοντας και αυτός το μπορεί να αντεπεξέλθει. Ο Apache/Windows αντιθέτως δεν κατάφερε να πιάσει αυτά τα νούμερα, ενώ ο Lighttpd/Windows δεν κατάφερε κάτι καλύτερο.

- **simpletest.aspx**

- Μέσοι όροι

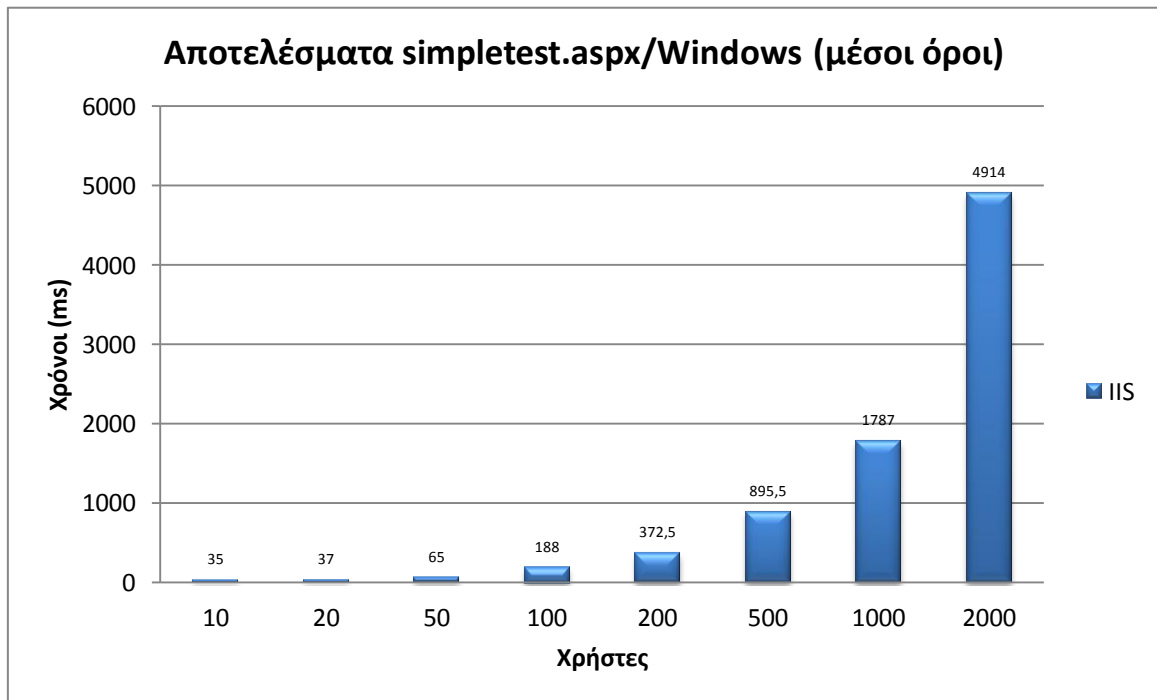
Οι μετρήσεις σε περιβάλλον Linux ήταν οι ακόλουθες:



Όπως βλέπουμε στο διάγραμμα, ο Apache υπερτερεί του Nginx και στους 2 δείκτες.

Μέτρηση απόδοσης Web Server

Ακολουθεί η αντίστοιχη μέτρηση σε Windows:



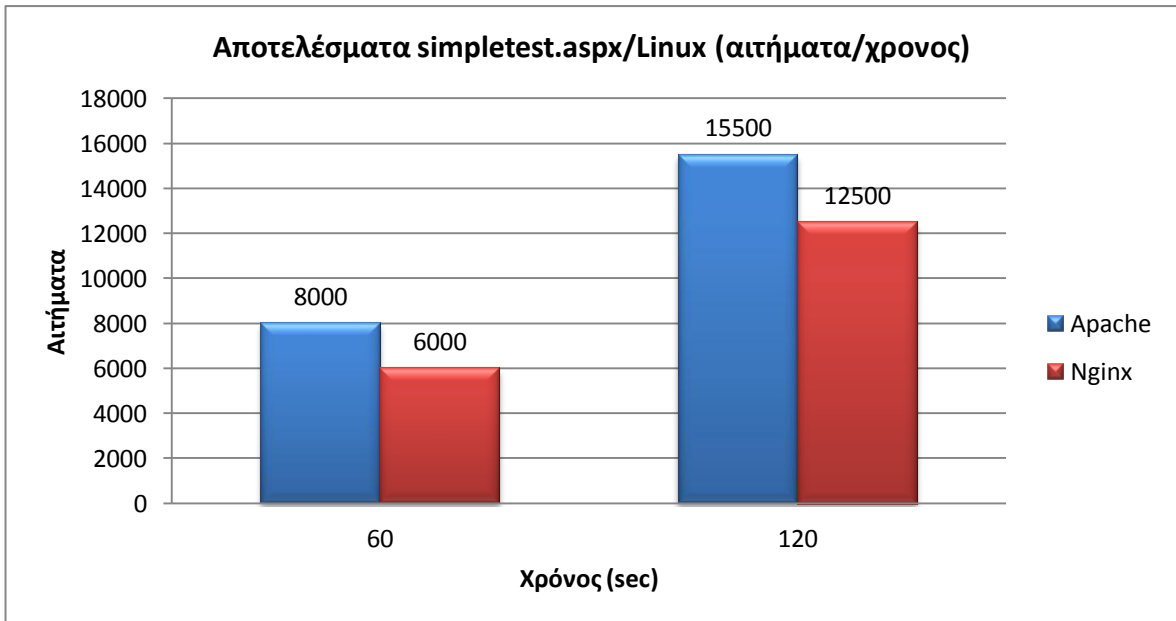
Σε αυτή τη μέτρηση τα μόνα αποτελέσματα που μπορούσαμε να πάρουμε ήταν από τον IIS, λόγω της μη δυνατής συνεργασίας της ASP.Net με τους άλλους web server.

Βλέποντας τώρα και τα 2 αποτελέσματα, παρατηρούμε ότι μέχρι τους 50 χρήστες ο IIS έχει το προβάδισμα, αλλά έπειτα ο Apache παρουσιάζεται αποδοτικότερος σε αυτό τον τομέα. Ο IIS όμως κατάφερε να εξυπηρετήσει τους 4-πλάσιους χρήστες από τον Apache. Ο Nginx από τη μεριά του έμεινε αρκετά πίσω και στους 2 δείκτες.

Μέτρηση απόδοσης Web Server

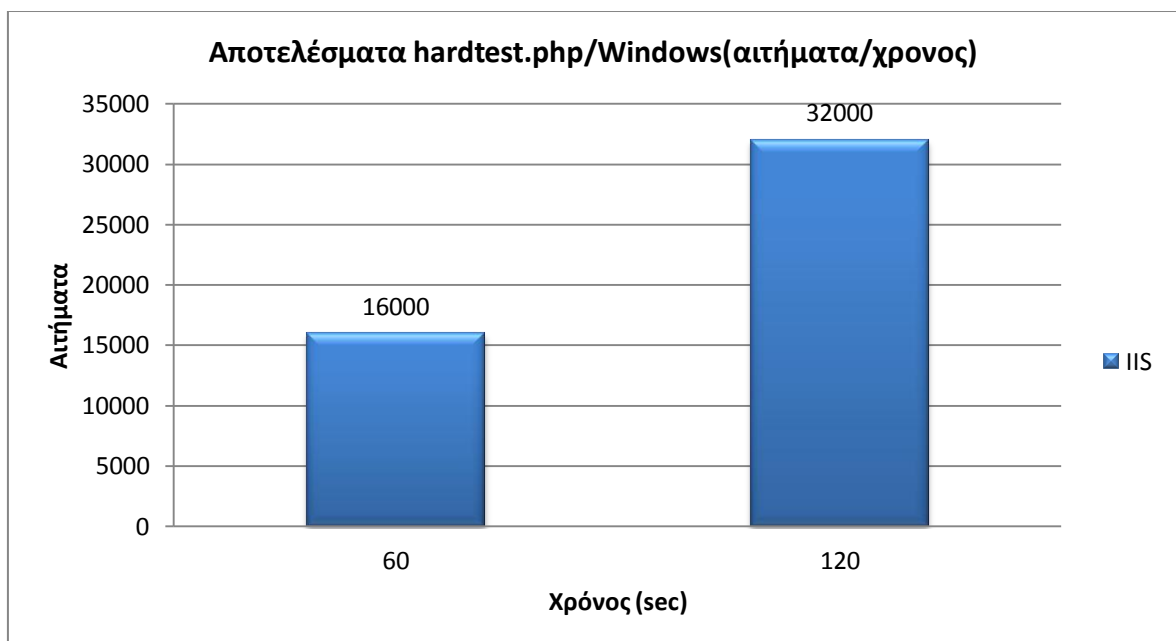
- Εξυπηρέτηση αιτημάτων σε 60 και 120 δευτερόλεπτα

Ακολουθούν οι μετρήσεις που έγιναν σε περιβάλλον Linux



Από το παραπάνω διάγραμμα είναι φανερή η υπεροχή του Apache, ο οποίος παρουσιάζει αποδοτικότερη συμπεριφορά από τον Nginx.

Η μέτρηση σε Windows κατέγραψε τα παρακάτω:

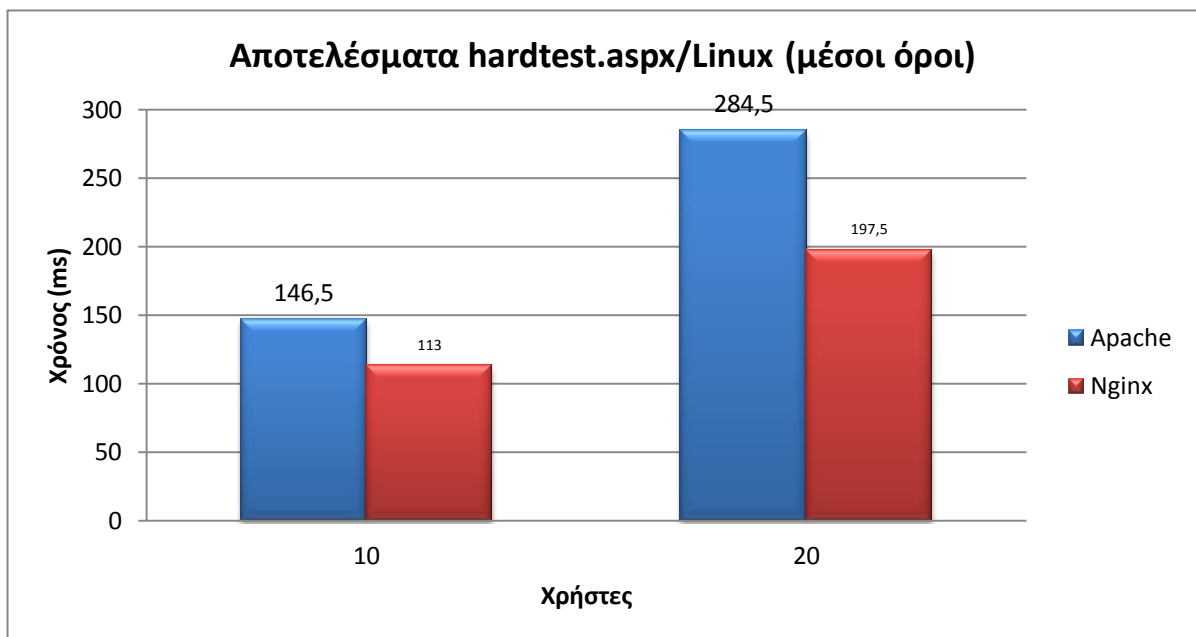


Μέτρηση απόδοσης Web Server

Συγκρίνοντας τα 2 διαγράμματα βλέπουμε ότι ο IIS είναι αποδοτικότερος από τον Apache, καθώς στους ίδιους χρόνους εξυπηρέτησε τα 2-πλάσια αιτήματα, καθώς και από τον Nginx που έμεινε αρκετά πιο πίσω.

- **hardtest.aspx**
 - Μέσοι όροι

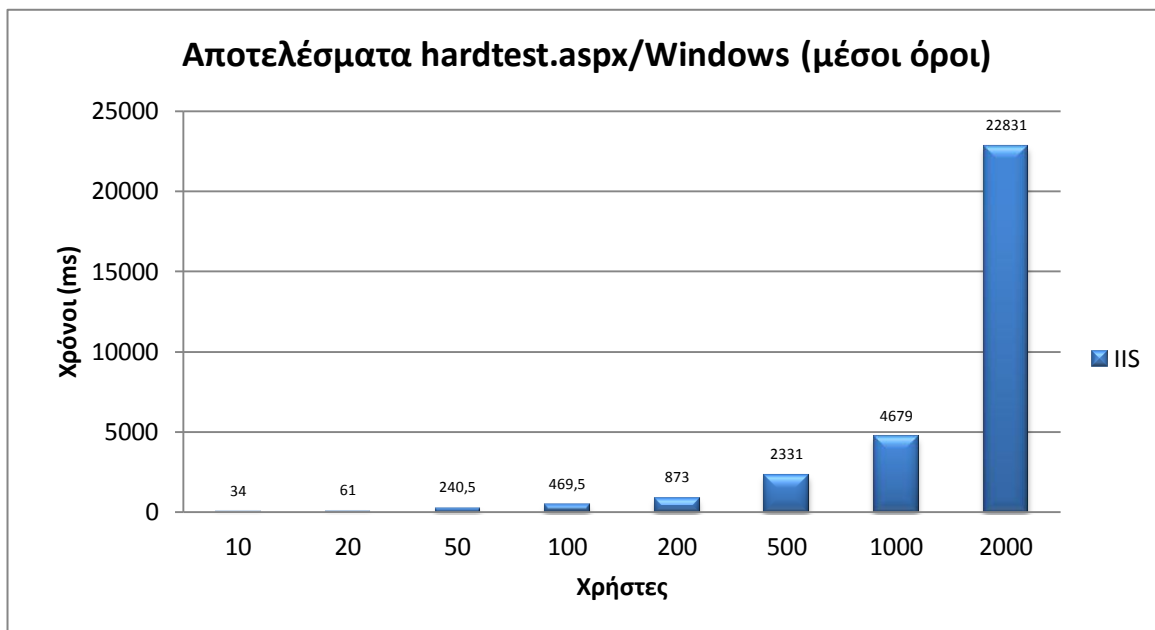
Οι μετρήσεις σε περιβάλλον Linux ήταν οι ακόλουθες:



Σε αυτό το διάγραμμα εμφανίζονται οι μετρήσεις μόνο για 10 και 20 χρήστες καθώς τα υπόλοιπα αποτελέσματα λόγω των πολλών σφαλμάτων δεν μπορούν να θεωρηθούν αξιόπιστα.

Μέτρηση απόδοσης Web Server

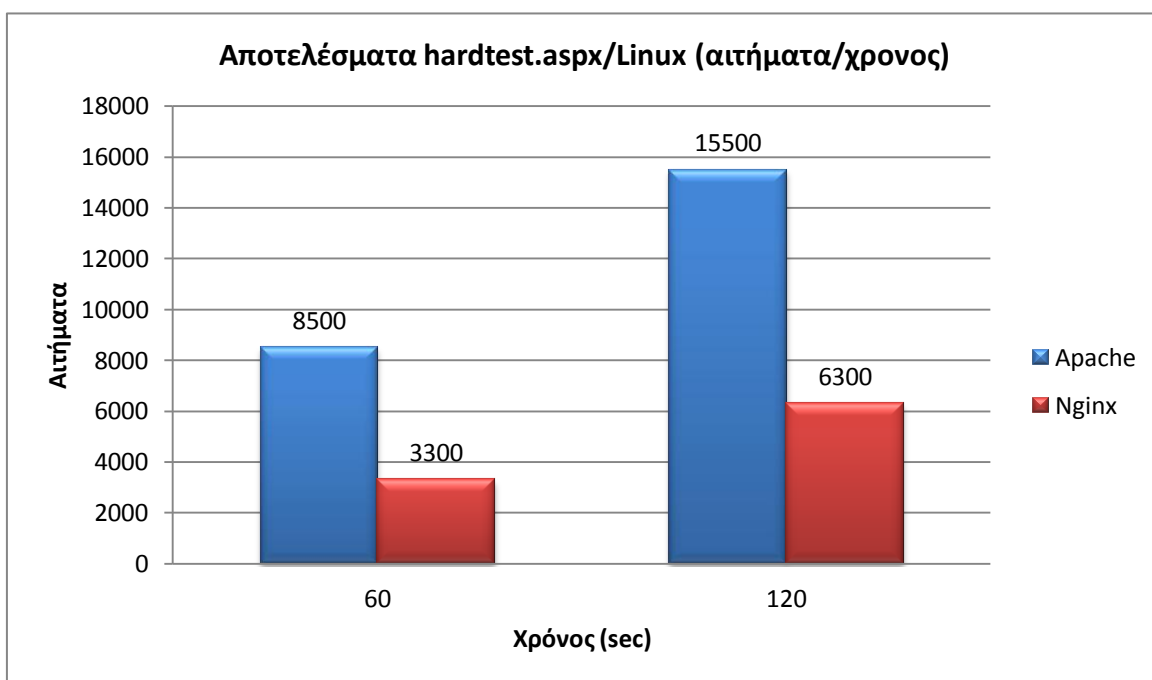
Ακολουθεί η αντίστοιχη μέτρηση σε Windows:



Εδώ δεν υπάρχει αμφιβολία για το ποιος υπερτερεί καθώς ο IIS υπερτερεί και στους 2 δείκτες.

- Εξυπηρέτηση αιτημάτων σε 60 και 120 δευτερόλεπτα

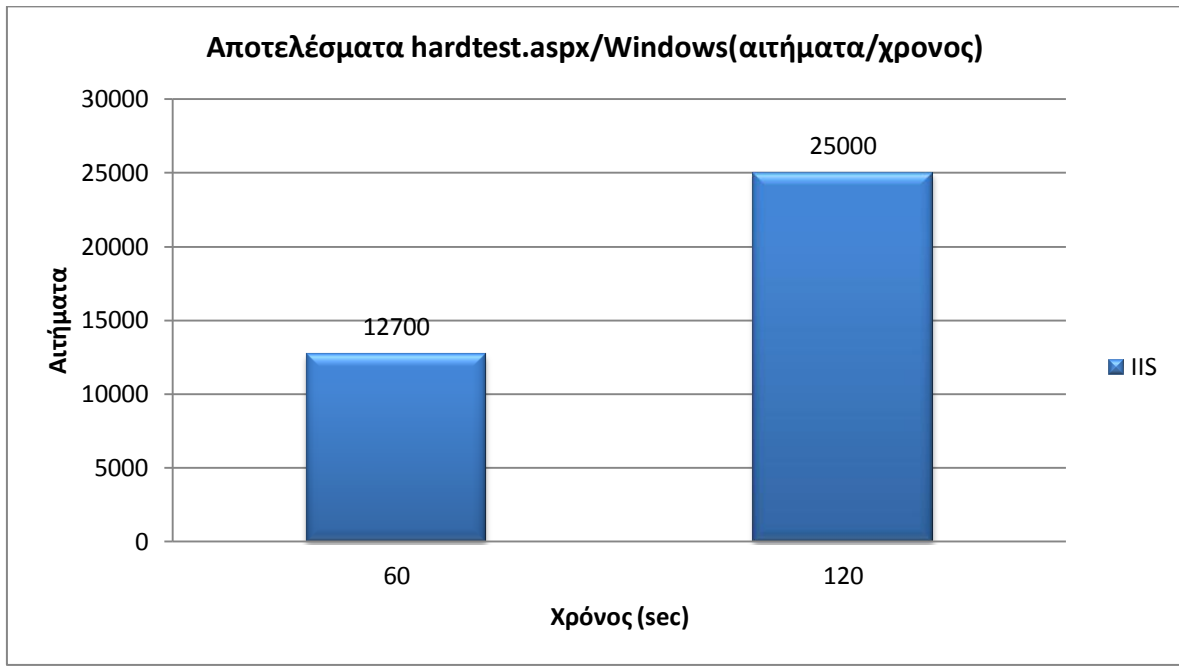
Ακολουθούν οι μετρήσεις που έγιναν σε περιβάλλον Linux



Μέτρηση απόδοσης Web Server

Και σε αυτή τη μέτρηση η Apache απέδωσε πολύ καλύτερα από τον Nginx. Βέβαια δεν πρέπει να ξεχνάμε ότι κανείς από τους 2 δεν κατάφερε να εξυπηρετήσει πάνω από 20 χρήστες.

Η μέτρηση σε Windows κατέγραψε τα παρακάτω:

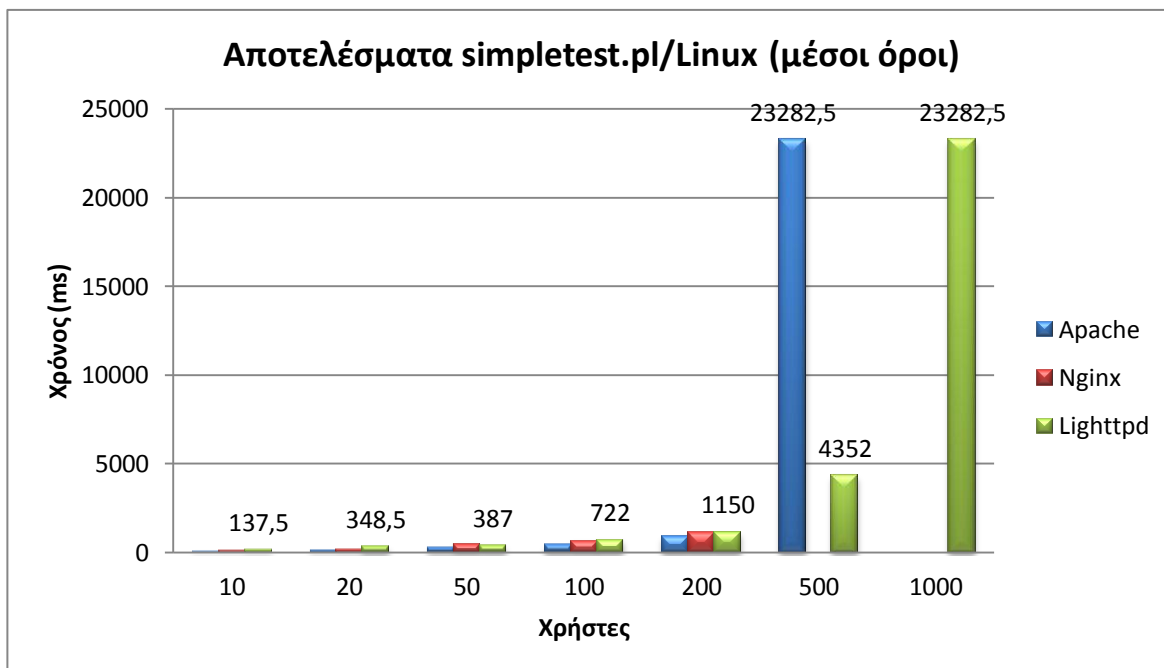


Είναι φανερό από τα παραπάνω ότι ο IIS υπερέχει και μπορεί να ανταποκριθεί καλύτερα στην αυξημένη επεξεργαστική ισχύ της ASP.Net.

- **simpletest.pl**

- Μέσοι όροι

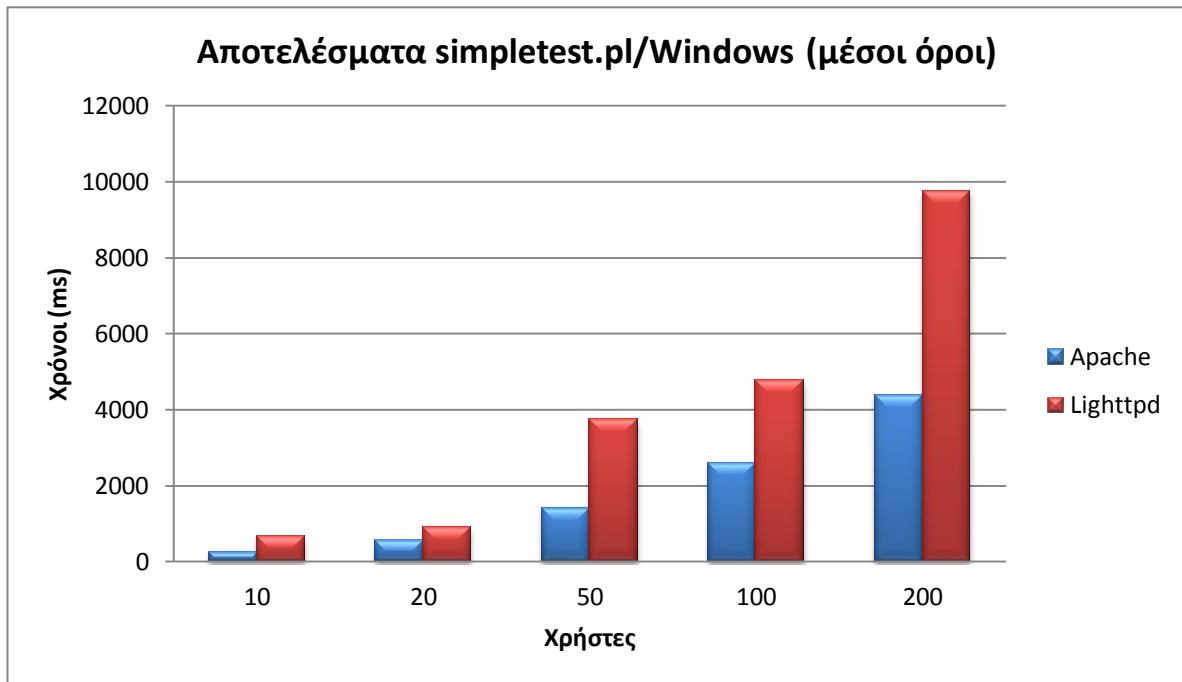
Οι μετρήσεις σε περιβάλλον Linux ήταν οι ακόλουθες:



Παρατηρώντας το παραπάνω, βλέπουμε τον Apache να έχει το προβάδισμα, όντας αποδοτικότερος μέχρι τους 200 χρήστες. Έπειτα όμως, και στους 500 χρήστες παρουσιάζει πολύ μεγάλη αύξηση των μέσων όρων του. Αντίθετα ο Lighttpd συνεχίζει να κρατιέται σε χαμηλά επίπεδα και εκτός αυτού φτάνει και τους 1000 χρήστες. Ο Nginx από τη μεριά του δεν κατάφερε να ξεπεράσει τους 200.

Μέτρηση απόδοσης Web Server

Ακολουθεί η αντίστοιχη μέτρηση σε Windows:



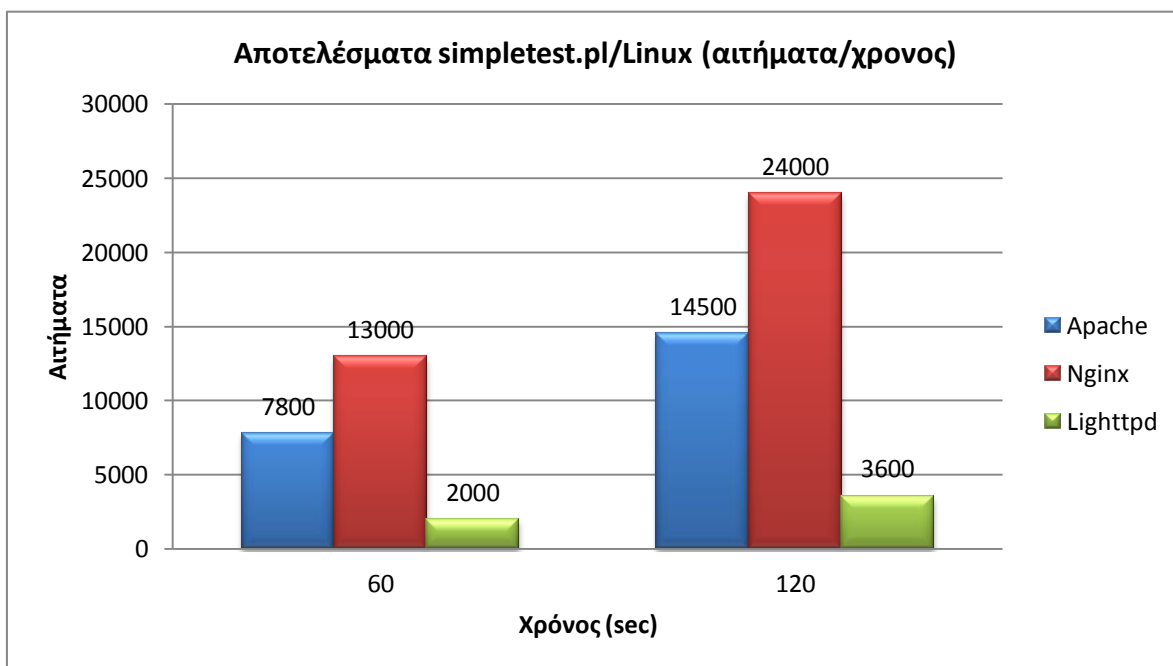
Σε αυτό το διάγραμμα βλέπουμε τον Apache να αποδίδει καλύτερα σε όλες τις ομάδες χρηστών, με τη διαφορά του με τον Lighttpd να ανοίγει όσο αυτοί αυξάνονται.

Αν συγκρίνουμε τώρα τα 2 διαγράμματα θα δούμε ότι όλοι οι web server σε Linux ανταποκρίνονται καλύτερα από τις υλοποιήσεις σε Windows.

Μέτρηση απόδοσης Web Server

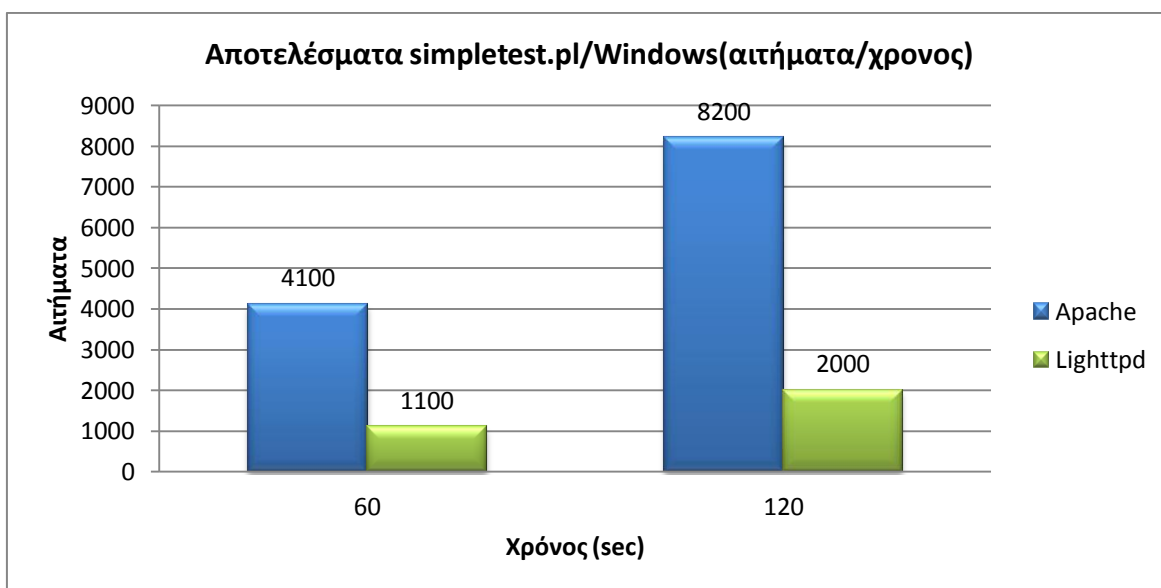
- Εξυπηρέτηση αιτημάτων σε 60 και 120 δευτερόλεπτα

Ακολουθούν οι μετρήσεις που έγιναν σε περιβάλλον Linux



Εδώ βλέπουμε, τον Nginx να έχει εξυπηρετήσει περισσότερα αιτήματα. Βέβαια δεν ξεχνάμε ότι σύμφωνα με τα παραπάνω ο αριθμός των χρηστών που εξυπηρέτησε ήταν μικρότερος έναντι των άλλων 2. Ο Apache έρχεται δεύτερος με τον Lighttpd να κινείται σε χαμηλά επίπεδα.

Η μέτρηση σε Windows κατέγραψε τα παρακάτω:



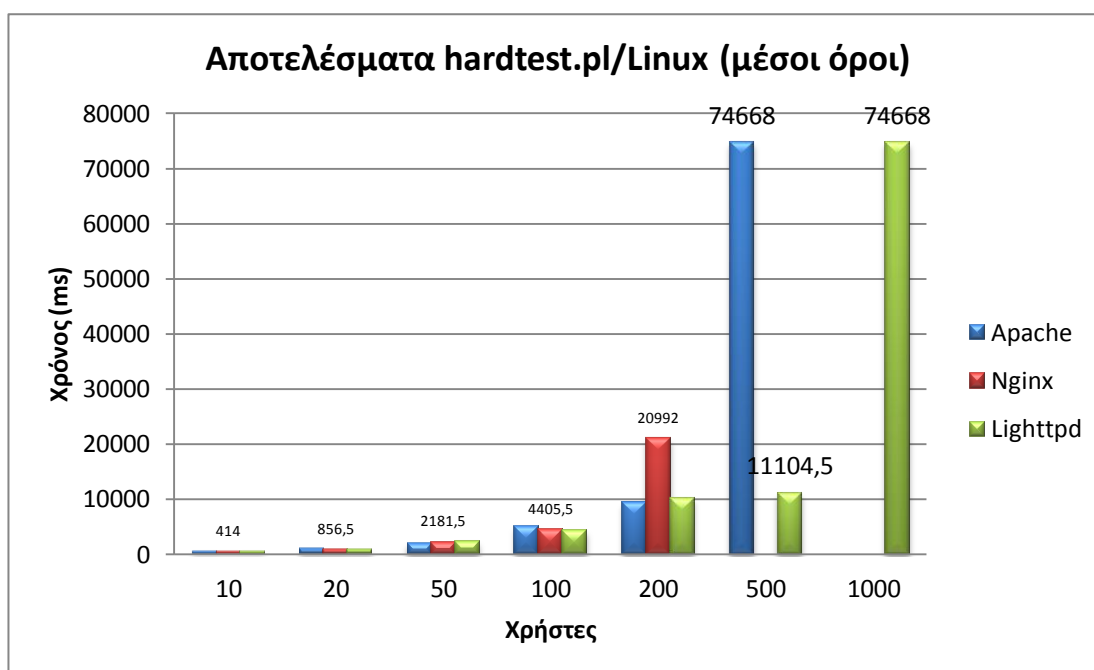
Όπως βλέπουμε στο διάγραμμα, ο Apache κατέγραψε 4-πλάσια ποσοστά από τον Lighttpd.

Παίρνοντας και τα 2 διαγράμματα, θα διαπιστώσουμε και εδώ ότι σε περιβάλλον Linux τα αιτήματα που εξυπηρετούνται είναι αρκετά περισσότερα από ότι σε Windows.

- **hardtest.pl**

- Μέσοι όροι

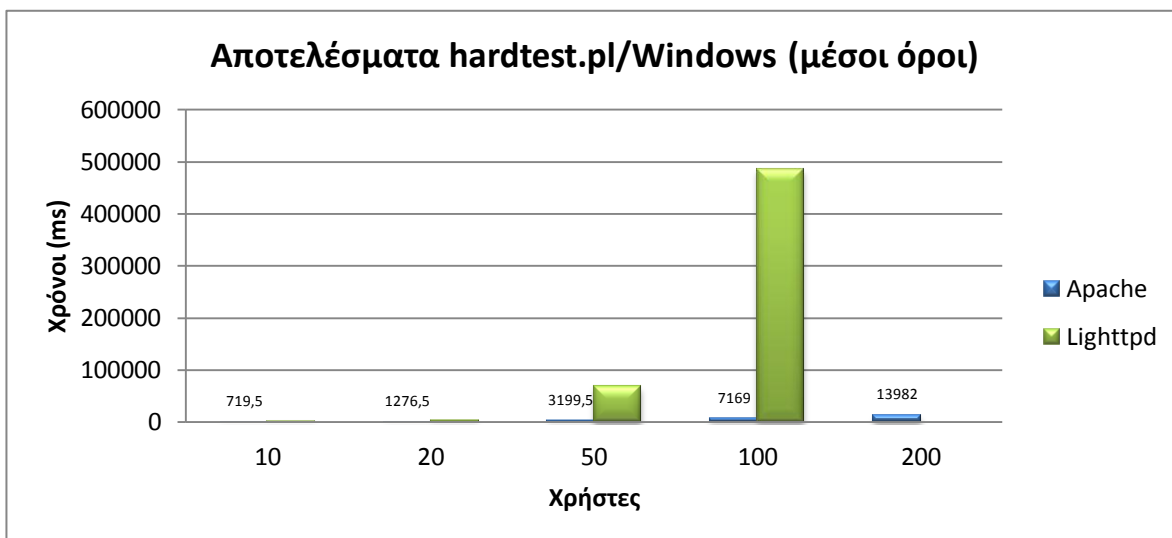
Οι μετρήσεις σε περιβάλλον Linux ήταν οι ακόλουθες:



Βλέποντας το παραπάνω παρατηρούμε να ότι κανείς web server δεν επικρατεί του άλλου σημαντικά, μέχρι τους 200 χρήστες. Εκεί, ο Apache με τον Lighttpd εμφανίζουν παρόμοια αποτελέσματα, ενώ ο Nginx φαίνεται να μην μπορεί να τους ακολουθήσει. Επίσης ο Nginx δεν κατάφερε να ξεπεράσει αυτό τον αριθμό των χρηστών. Από εκεί και πέρα, ο Apache και ο Lighttpd εμφάνισαν συμπεριφορά παρόμοια με αυτή της μέτρησης του simpletest.pl

Μέτρηση απόδοσης Web Server

Ακολουθεί η αντίστοιχη μέτρηση σε Windows:

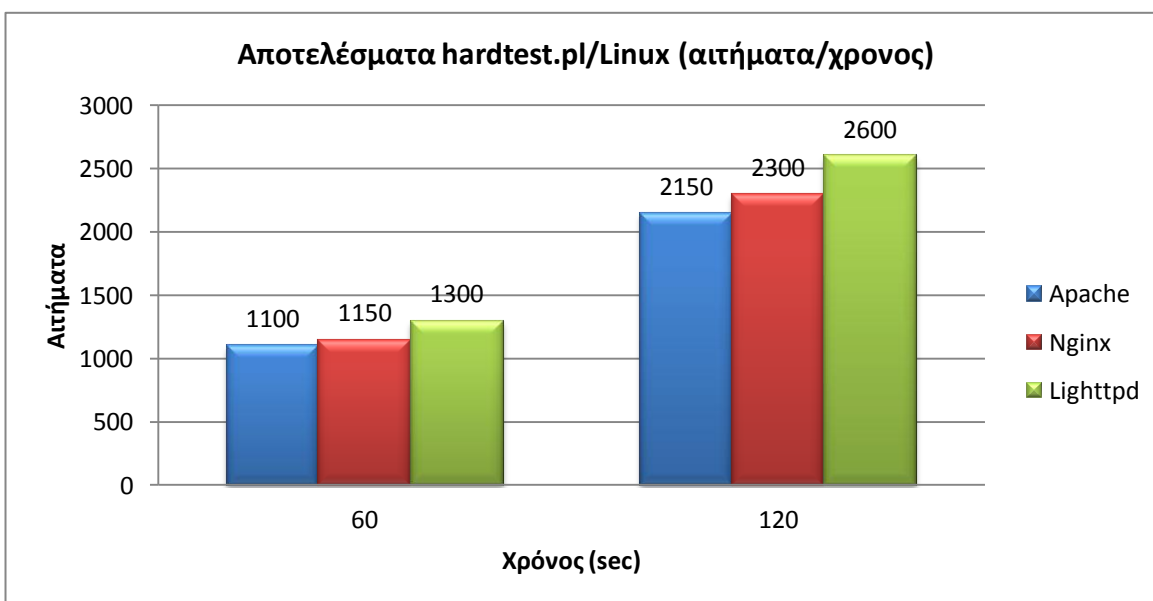


Σε αυτό το διάγραμμα είναι εμφανές ότι ο Apache έχει πολύ καλύτερη απόδοση από τον Lighttpd

Συγκριτικά ανάμεσα στις 2 μετρήσεις αυτό που βλέπουμε είναι ότι και εδώ οι επιδόσεις των web server σε περιβάλλον Linux είναι αποδοτικότερες σε σχέση με αυτές των Windows.

- Εξυπηρέτηση αιτημάτων σε 60 και 120 δευτερόλεπτα

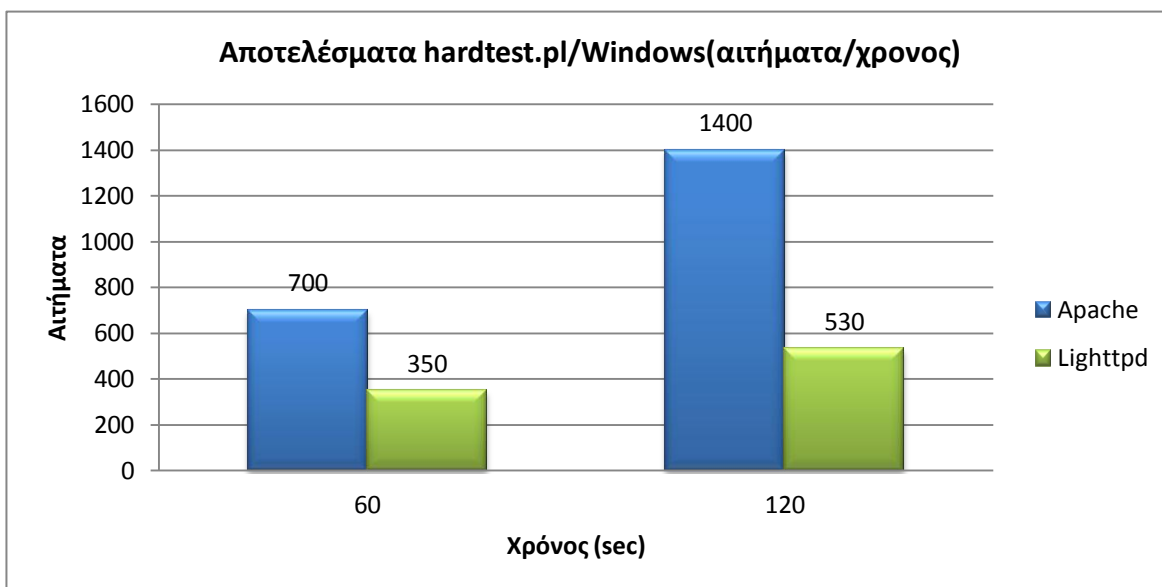
Ακολουθούν οι μετρήσεις που έγιναν σε περιβάλλον Linux



Μέτρηση απόδοσης Web Server

Σε αυτή τη μέτρηση τα πράγματα αντιστράφηκαν αν λάβουμε υπόψη την αντίστοιχη μέτρηση του simpletest.pl. Αυτό δεν οφείλεται στην άνοδο του Lighttpd αλλά στη μεγάλη πτώση των άλλων 2 web server.

Η μέτρηση σε Windows κατέγραψε τα παρακάτω:



Για άλλη μία φορά, ο Apache κατέγραψε μεγαλύτερη αποδοτικότητα από τον Lighttpd.

Και σε αυτές τις περιπτώσεις, οι web server κατέγραψαν καλύτερα αποτελέσματα μέσω Linux.

III. Συμπεράσματα

Έχοντας καταγράψει όλα τα παραπάνω θα προσπαθήσουμε να βγάλουμε μερικά συμπεράσματα για όλα αυτά που καταγράψαμε και είδαμε.

Καταρχάς, διαπιστώνουμε ότι ο ίδιος web server μπορεί να έχει διαφορετική συμπεριφορά ανάλογα σε ποιο λειτουργικό σύστημα τον χρησιμοποιούμε. Στις περισσότερες των περιπτώσεων, οι υλοποιήσεις σε Linux φάνηκαν να έχουν το προβάδισμα έναντι αυτόν των Windows. Φυσικά μιλάμε για τους web server ανοιχτού κώδικα και όχι για τον IIS ο οποίος δεν διατίθεται σε έκδοση για λειτουργικό σύστημα διαφορετικό των windows.

Από άποψη λειτουργίας, οι Apache και IIS αποδείχτηκαν οι πιο σταθεροί, καθώς, εκτός από ελάχιστες εξαιρέσεις, δεν προκλήθηκε κατάρρευση της υπηρεσίας τους. Αντίθετα Nginx και Lighttpd θέλουν βελτίωση σε αυτό τον τομέα. Επίσης Apache και IIS είναι ευκολότερη στη ρύθμιση, βοηθώντας έναν άπειρο χρήστη. Όσον αφορά τον Apache, στο διαδίκτυο μπορεί κανείς να βρει τη μεγαλύτερη ποικιλία σε εγχειρίδια χρήσης.

Στις επιδόσεις τώρα, και ξεκινώντας από τα στατικά αρχεία, βλέπουμε ότι υπάρχει μία σχετική ισορροπία μεταξύ των Apache, Nginx, Lighttpd σε Linux και IIS. Έπειτα ακολουθούσε ο Apache σε windows και ο Lighttpd σε windows. Στα δυναμικά αρχεία, ανάλογα με τον τύπο του αρχείου υπήρξαν και διαφορετικές επιδόσεις. Στα αρχεία PHP, η τριάδα Apache, Nginx, Lighttpd σε Linux συναγωνίστηκαν χωρίς κάποιος να πάρει ένα σαφή προβάδισμα., παρουσιάζοντας και οι 3 πολύ καλή συμπεριφορά. Οι Apache σε windows και IIS ακολούθησαν, με τον Apache να έχει προβάδισμα στις περιπτώσεις όπου δεν απαιτούνταν μεγάλη επεξεργαστική ισχύ από τον server. Ο Lighttpd από τη μεριά του φαίνεται ότι θέλει πολύ δουλειά ακόμα σε περιβάλλον windows, καθώς δεν απέδωσε καθόλου ικανοποιητικά. Για την ASP.Net δεν τίθεται ζήτημα για το ποιος ήταν ο καλύτερος, αφού ο IIS παρουσίασε καλύτερα αποτελέσματα, με ίσως μόνο τον Apache σε Linux να προσπάθησε να τον ανταγωνιστεί και αυτό μόνο στην περίπτωση του απλού αρχείου. Στις μετρήσεις τις Perl, οι επιδόσεις των Apache, Nginx και Lighttpd σε Linux ως προς τους μέσους όρους είχαν συμπεριφορά παρόμοια με αυτήν της PHP. Από εκεί και πέρα ο Lighttpd κατάφερε να εξυπηρετήσει τους περισσότερους χρήστες. Εκείνος πάντως που μπορεί να θεωρηθεί πρώτος στην εξυπηρέτηση των χρηστών είναι ο IIS καθώς σε όποιες μετρήσεις χρησιμοποιήθηκε είχε το μέγιστο των χρηστών.

Τέλος, αν συμπεριλάβουμε υπόψη όλα τα παραπάνω μπορούμε να βγάλουμε ένα συμπέρασμα για το ποιος web server είναι ο καλύτερος για τις ανάγκες μας. Πιο σταθεροί έδειξαν οι Apache και IIS, δικαιολογώντας το γεγονός ότι μαζί κατέχουν το 70% των σελίδων του διαδικτύου. Ο Nginx και Lighttpd δεν υστερούν τόσο σε επιδόσεις, αλλά στην ίδια την λειτουργία τους καθώς και στο γεγονός ότι δεν παρέχουν την ίδια υποστήριξη και για τα δύο λειτουργικά συστήματα.

Κεφάλαιο 8 - Java Servlets – Αρχεία JSP

8.1 Εισαγωγή στα Servlets

Όταν αναφερόμαστε σε servlets, αναφερόμαστε στη δυνατότητα της χρησιμοποίησης της γλώσσας προγραμματισμού Java σε εφαρμογές του διαδικτύου. Χρησιμοποιούνται κυρίως για την επέκταση των δυνατοτήτων ενός web server, παρόλο που ουσιαστικά μπορούν να εξυπηρετήσουν οποιοδήποτε αίτημα. Είναι σχεδιασμένα με τέτοιο τρόπο ώστε να μπορούν να λειτουργούν ανεξάρτητα χωρίς την μεσολάβηση κάποιου άλλου προγράμματος.

Ένα servlet είναι στην πράξη μία κλάση της Java της πλατφόρμας Java EE (Enterprise Edition), έχοντας πρόσβαση σε όλο το Java API, με σκοπό την δημιουργία δυναμικών εφαρμογών διαδικτύου χρησιμοποιώντας τις δυνατότητες της Java. Χρησιμοποιεί το πρωτόκολλο Java Servlet API, το οποίο διεκπεραιώνει τα αιτήματα των χρηστών προς μία Java κλάση. Για την επικοινωνία του server με τον client χρησιμοποιείται το πρωτόκολλο HTTP.

Η πρώτη ολοκληρωμένη έκδοση τους δημιουργήθηκε από την εταιρία Sun Microsystems, με την 1.0 να τελειοποιείτο το καλοκαίρι του 1997, ενώ μέχρι τώρα έχει φτάσει στην έκδοση 3.0 (Μάρτιος 2010). Σκέψεις για την δημιουργία των servlet υπήρξε από την αρχή της ανάπτυξης της Java, με κύριο εκφραστή τον προγραμματιστή James Gosling. Η ιδέα πάντως δεν προχώρησε μέχρι η εταιρεία να επεκταθεί στο χώρο των διαδικτυακών εφαρμογών, με την γνωστή σήμερα Java EE.

Το παραγόμενο αποτέλεσμα των servlets είναι συνήθως HTML κώδικας, μπορώντας να εξάγει και άλλο περιεχόμενο όπως κώδικα XML. Σήμερα, η απευθείας χρήση τους έχει περιοριστεί. Αντί αυτού, χρησιμοποιείται η τεχνολογία JavaServer Pages (JSP), η οποία έχει τη δυνατότητα της αυτόματης παραγωγής του περιεχομένου ενός Servlet. Η κύρια διαφορά μεταξύ ενός Servlet και ενός JSP είναι ότι ένας Servlet ενσωματώνει κώδικα HTML σε κώδικα Java, ενώ τα JSP αρχεία ενσωματώνουν τον κώδικα της Java μέσα σε αυτόν της HTML.

Για να λειτουργήσει ένα Servlet ή ένα JSP αρχείο χρησιμοποιείται ένας κατάλληλος web server (στην παρούσα περίπτωση ονομάζονται Web Containers), που θα μπορεί να τα διαχειριστεί. Τα JSP αρχεία μεταγλωττίζονται από τον

JavaServer Pages compiler. Οι web containers που εξετάζονται εδώ είναι ο Apache Tomcat της Apache Foundation, ο Glassfish της Sun Microsystems και ο Jetty από την Eclipse Foundation.

8.2 Ιστορικά στοιχεία

- Apache Tomcat

Ο Apache Tomcat (γνωστός και ως Jakarta Tomcat ή απλώς Tomcat) είναι ένας web container, ανοιχτού κώδικα, δημιούργημα της Apache Software Foundation. Σκοπός του η πλήρη υποστήριξη των Java Servlets και των JavaServer Pages και η παροχή ενός HTTP web server που θα είναι σε θέση να μεταγλωττίσει κώδικα Java. Δεν πρέπει να συγχέεται με τον Apache web server, καθώς πρόκειται για μία τελείως διαφορετική υλοποίηση. Παρόλα αυτά, οι δύο αυτοί web servers χρησιμοποιούνται συχνά μαζί για την παροχή μίας πλήρους λύσης web υπηρεσιών, αν και δεν υπάρχουν σαν ένα συνολικό πακέτο. Ο Tomcat διαθέτει δικά του εργαλεία ρύθμισης και παραμετροποίησης, μπορώντας να τροποποιηθεί και μέσω XML αρχείων.

Δημιουργός του Tomcat είναι ο James Duncan Davidson, προγραμματιστής της Sun Microsystems, ο οποίος χρησιμοποίησε ήδη υπάρχουσες υλοποιήσεις για servlets προκειμένου να προχωρήσει στην ανάπτυξή του. Συνέβαλε επίσης στο να γίνει ένας ανοιχτού κώδικα web container, καθώς και στην δωρεά του όλου project στην Apache Software Foundation. Η πρώτη έκδοση του 1.0 χρονολογείται το 1999, ενώ η νεότερη έκδοσή του είναι η 7.0.22.

- GlassFish

Ο GlassFish (επίσημη ονομασία Oracle GlassFish Server) είναι ένας ανοιχτού κώδικα web container για την υποστήριξη της πλατφόρμας Java EE. Η ανάπτυξή του ξεκίνησε από τη Sun Microsystems, με την Oracle να συμμετέχει στην διαδικασία και να τον προωθήει στο ευρύ κοινό. Ουσιαστικά, χρησιμοποιήθηκε ένα μέρος της υπάρχουσας υλοποίησης του Tomcat (που επίσης ξεκίνησε από τη Sun) με την εισαγωγή νέων στοιχείων, όπως του Grizzly, για την χρήση του νέου I/O της Java, προσθέτοντας του περισσότερη λειτουργικότητα και ταχύτητα.

Η πρώτη έκδοσή του χρονολογείται τον Ιούνιο του 2005, ενώ ένα χρόνο αργότερα προχώρησε στην αναβάθμισή του ώστε να υποστηρίζει την Java EE 5. Η 3^η έκδοση του, που λανσαρίστηκε το Δεκέμβριο του 2009, έκανε τον Glassfish τον πρώτο που υποστήριξε την Java EE 6, ενώ παρείχε τη δυνατότητα της εύκολης μετάβασης από ένα Tomcat server σε Glassfish.

- Jetty

Ο Jetty είναι ένας web container ανοιχτού κώδικα γραμμένος εξ ολοκλήρου σε Java. Ξεκίνησε σαν ένα ανεξάρτητο project το 1995 από τον Greg Wilkins, ως ένα μία προσθήκη στον κρατικό server του Mort Bay (περιοχή του προαστίου Bal main, Sydney). Από το 2000 μέχρι το 2005, το project φιλοξενήθηκε από το sourceforge.net. Το 2005, εντάσσεται στην ομάδα ανοιχτού λογισμικού Codehaus, ενώ το 2009 μετακινείται για άλλη μια φορά για να γίνει μέρος του Eclipse Foundation, προκειμένου να εξελιχθεί και να διευρύνει το κοινό του μέσω του ιδρύματος. Η ομάδα του Codehaus συνεχίζει να του παρέχει υποστήριξη και επεκτάσεις. Σήμερα βρίσκεται στην έκδοση 7.4.5

Οι τρεις παραπάνω web containers είναι cross-platform. Στην περίπτωση του Jetty όμως μετρήσεις σε Windows δεν καταγράφηκαν, καθώς δεν κατάφερε να φέρει εις πέρας καμία μέτρηση. Ο λόγος ήταν ότι σε οποιαδήποτε προσπάθεια έγινε η υπηρεσία του σταματούσε τη λειτουργία της. Επομένως για αυτόν υπάρχουν μόνο αποτελέσματα σε περιβάλλον Linux.

8.3 Αποτελέσματα μετρήσεων

Στη συνέχεια παρουσιάζονται τα συγκεντρωτικά αποτελέσματα που προέκυψαν από την διενέργεια των μετρήσεων.

- Apache Tomcat

I. Μετρήσεις σε Linux

Στις μετρήσεις του GlassFish σε περιβάλλον Linux το μέγιστο των χρηστών που επιτεύχθηκε ήταν μόλις 200 για το αρχείο simpletest.jsp και μόλις 100 για το hardesttest.jsp, χωρίς όμως στα καταγεγραμμένα να υπάρξει κάποιο σφάλμα .

Μέτρηση απόδοσης Web Server

Αναλυτικότερα για το `simpletest.jsp` τα αποτελέσματα είναι:

Πίνακας 48. Αποτελέσματα για `simpletest.jsp` σε περιβάλλον Linux/Tomcat

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	16 - 22 ms
20	32 - 53 ms
50	102 - 121 ms
100	181 - 204 ms
200	374 - 541 ms

Όπως βλέπουμε από τον πίνακα, δεν κατάφεραν να εξυπηρετηθούν πάνω από 200 χρήστες. Οι μέσοι όροι θα χρησιμοποιηθούν για σύγκριση με τους υπόλοιπους web containers καθώς από μόνοι τους δεν μας δίνουν κάποιο ασφαλή στοιχείο. Σφάλματα δεν καταγράφηκαν στην μέτρηση αυτή. Σε 60 δευτερόλεπτα εξυπηρετήθηκαν ~14100 αιτήματα και σε 120 ~27500.

Ακολουθούν τα αποτελέσματα του `hardtest.jsp`

Πίνακας 49. Αποτελέσματα για `hardtest.jsp` σε περιβάλλον Linux/Tomcat

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	311 - 318 ms
20	731 - 803 ms
50	2330 - 3239 ms
100	5156 - 7536 ms

Η μέτρηση, εκτός από την φυσιολογική αύξηση των μέσων όρων, δεν διαφέρει σημαντικά από την προηγούμενη. Δεν μπορούμε φυσικά να παραβλέψουμε το γεγονός ότι ο αριθμός των μόλις 100 χρηστών που εξυπηρετήθηκαν θεωρείται πάρα πολύ μικρός. Όσον αφορά τα αιτήματα που εξυπηρετήθηκαν αυτά κατέγραψαν φθίνουσα πορεία όσο οι χρήστες αυξάνονταν. Ενδεικτικά αναφέρουμε ότι για 10 χρήστες σε 60 δευτερόλεπτα τα αιτήματα που εξυπηρετήθηκαν ήταν ~1500, ενώ για 100 χρήστες ήταν ~800.

II. Μετρήσεις σε Windows

Σε περιβάλλον Windows, ο Tomcat φάνηκε να λειτουργεί αποδοτικότερα όσον αφορά τον αριθμό των χρηστών, που έφτασε στις μετρήσεις και των 2 αρχείων τους 1000. Μικρά σφάλματα καταγράφηκαν μόνο στη μέτρηση των 1000 χρηστών του αρχείου `hardtest.jsp`. Αναλυτικότερα για το `simpletest.jsp` προέκυψαν τα παρακάτω:

Πίνακας 50. Αποτελέσματα για `simpletest.jsp` σε περιβάλλον Windows/Tomcat

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	6 - 17 ms
20	29 - 33 ms
50	33 - 86 ms
100	57 - 168 ms
200	88 - 363 ms
500	245 - 656 ms
1000	447 - 874 ms

Παρατηρώντας τον πίνακα διαπιστώνουμε, σε σύγκριση με την αντίστοιχη μέτρηση σε Linux, ότι εκτός από περισσότερους χρήστες, ο Tomcat σε Windows παρουσιάζει και καλύτερους μέσους όρους. Σφάλματα δεν καταγράφηκαν στην

Μέτρηση απόδοσης Web Server

μέτρηση. Σε 60 δευτερόλεπτα εξυπηρετήθηκαν ~14100 αιτήματα και σε 120 ~28300.

Για `hardtest.jsp` τα αποτελέσματα είναι:

Πίνακας 51. Αποτελέσματα για `hardtest.jsp` σε περιβάλλον Windows/Tomcat

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	416 - 742 ms
20	434 - 1524 ms
50	510 - 4001 ms
100	867 - 8126 ms
200	689 - 17529 ms
500	30606 - 45155 ms
1000	63098 - 81697 ms

Σε αυτή τη μέτρηση υπήρξαν μεγάλες αποκλίσεις η οποίες προκλήθηκαν από το Jmeter, δείχνοντας αστάθεια. Αν πάρουμε υπόψη μόνο τις μετρήσεις του `ab tool`, θα δούμε ότι οι μέσοι όροι είναι μεγαλύτεροι από την αντίστοιχη μέτρηση σε Linux. Από την άλλη μεριά, εδώ εξυπηρετήθηκαν περισσότεροι χρήστες, με τη μέτρηση των 1000 χρηστών να καταγράφει μικρά σφάλματα. Σε 60 δευτερόλεπτα εξυπηρετήθηκαν ~650 αιτήματα και σε 120 ~1300.

- GlassFish

I. Μετρήσεις σε Linux

Στις μετρήσεις που διεξήχθησαν για τον GlassFish το μέγιστο των χρηστών που επιτεύχθηκε ήταν οι 1000. Σφάλματα παρουσιάστηκαν μόνο στην περίπτωση των 1000 χρηστών (και στα δύο αρχεία). Αναλυτικότερα για το `simpletest.jsp` τα αποτελέσματα είναι:

Πίνακας 52. Αποτελέσματα για simpletest.jsp σε περιβάλλον Linux/Glassfish

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	36 - 97 ms
20	71 - 104 ms
50	180 - 247 ms
100	382 - 468 ms
200	796 - 1653 ms
500	2502 - 3199 ms
1000	5033 - 10572 ms

Στην παραπάνω μέτρηση το ενδιαφέρον στοιχείο που προέκυψε ήταν ότι όταν ο server εξυπηρετούσε περισσότερα αιτήματα τότε οι μέσοι όροι μειώνονταν (αναφερόμαστε πάντα για ίδιες ομάδες χρηστών). Οι αποκλίσεις που φαίνονται είναι λόγω της μέτρησης με διαφορετικά εργαλεία. Σφάλματα εμφανιστήκαν σε μικρό ποσοστό στη μέτρηση των 1000 χρηστών. Σε 60 δευτερόλεπτα εξυπηρετήθηκαν ~7300 αιτήματα και σε 120 ~17000.

Ακολουθούν τα αποτελέσματα του hardtest.jsp

Πίνακας 53. Αποτελέσματα για hardtest.jsp σε περιβάλλον Linux/GlassFish

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	313 - 347 ms
20	497 - 729 ms
50	1106 - 1919 ms

Μέτρηση απόδοσης Web Server

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
100	1997 - 3659 ms
200	3908 - 7137 ms
500	13836 - 17530 ms
1000	25366 - 27601 ms

Όπως και στην προηγούμενη μέτρηση έτσι και εδώ, η εξυπηρέτηση περισσότερων αιτημάτων απέφερε χαμηλότερους μέσους όρους. Σφάλματα είχαμε και εδώ στους 1000 χρήστες με το ποσοστό τους να φτάνει το 40%. Σε 60 δευτερόλεπτα εξυπηρετήθηκαν ~1500 αιτήματα και σε 120 ~3200.

II. Μετρήσεις σε Windows

Σε περιβάλλον Windows το μέγιστο των χρηστών που επιτεύχθηκε ήταν 500 και για τα δύο αρχεία. Μικρά σφάλματα υπήρξαν στις μετρήσεις των 500 χρηστών. Αναλυτικότερα για το `simpletest.jsp` προέκυψαν τα παρακάτω:

Πίνακας 54. Αποτελέσματα για `simpletest.jsp` σε περιβάλλον Windows/GlassFish

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	14 - 26 ms
20	28 - 55 ms
50	85 - 138 ms
100	197 - 275 ms
200	438 - 589 ms
500	1118 - 1328 ms

Μέτρηση απόδοσης Web Server

Όπως βλέπουμε, ο GlassFish στην περίπτωση αυτού του αρχείου εξυπηρετήσε γρηγορότερα τα αιτήματα σε σχέση με την αντίστοιχη μέτρηση που πραγματοποιήθηκε σε περιβάλλον Linux. Παράλληλα και εδώ είχαμε το φαινόμενο της ταχύτερης εξυπηρετήσης των αιτημάτων όσο αυτά αυξάνονταν. Παρόλα αυτά δεν ξεπέρασε το όριο των 500 χρηστών (σε αντίθεση με τους 1000 χρήστες της άλλης μέτρησης). Επίσης σε αυτή την ομάδα εμφανίστηκαν και μικρά σφάλματα. Σε 60 δευτερόλεπτα εξυπηρετήθηκαν ~13600 αιτήματα και σε 120 ~27600.

Για `hardtest.jsp` τα αποτελέσματα είναι:

Πίνακας 55. Αποτελέσματα για `hardtest.jsp` σε περιβάλλον Windows/GlassFish

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	681 - 810 ms
20	1331 - 1633 ms
50	3322 - 4108 ms
100	6697 - 7967 ms
200	12063 - 15939 ms
500	33632 - 35788 ms

Σε αντίθεση με την προηγούμενη μέτρηση, εδώ ο GlassFish φάνηκε πιο αργός σε σχέση με την αντίστοιχη μέτρηση σε Linux. Επίσης ούτε εδώ κατάφερε να ξεπεράσει τους 500 χρήστες, όπου εμφανίστηκαν και μικρά σφάλματα. Σε 60 δευτερόλεπτα εξυπηρετήθηκαν ~740 αιτήματα και σε 120 ~1500.

- Jetty

Όπως προαναφέρθηκε, μετρήσεις σε Windows δεν ήταν τελικά εφικτές, επομένως αποτελέσματα υπάρχουν μόνο για την υλοποίηση σε Linux.

Στις μετρήσεις τώρα, το μέγιστο των χρηστών που επιτεύχθηκε ήταν οι 1000 και για τα 2 αρχεία. Σφάλματα παρουσιάστηκαν στις μετρήσεις των 500 και 1000

Μέτρηση απόδοσης Web Server

χρηστών για το αρχείο `hardtest.jsp`. Αναλυτικότερα για το `simpletest.jsp` τα αποτελέσματα είναι:

Πίνακας 56. Αποτελέσματα για `simpletest.jsp` σε περιβάλλον Linux/Jetty

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	10- 36 ms
20	12 - 69 ms
50	23 - 171 ms
100	34 - 334 ms
200	29 - 681 ms
500	88 - 252 ms
1000	160 - 450 ms

Στην παραπάνω μέτρηση υπήρξαν και εδώ μεγάλες διαφορές μεταξύ των αποτελεσμάτων του `ab tool` και του `Jmeter`. Ακόμα το `ab tool` δεν μπόρεσε να μετρήσει πάνω από 200 χρήστες, καθώς ο `server` έκλεινε τη σύνδεση. Επίσης, όπως και στις μετρήσεις του `GlassFish`, περισσότερα αιτήματα είχαν σαν αποτέλεσμα πιο μικρούς μέσους όρους για τις ίδιες ομάδες χρηστών. Σφάλματα δεν καταγράφηκαν. Σε 60 δευτερόλεπτα εξυπηρετήθηκαν ~14000 αιτήματα και σε 120 ~28200.

Ακολουθούν τα αποτελέσματα του `hardtest.jsp`

Πίνακας 57. Αποτελέσματα για `hardtest.jsp` σε περιβάλλον Linux/Jetty

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
10	300 - 374 ms

Μέτρηση απόδοσης Web Server

Αριθμός χρηστών	Μέσος χρόνος απόκρισης αιτημάτων (ανάλογα με τον αριθμό των αιτημάτων)
20	201 - 653 ms
50	1106 - 1691 ms
100	3101 - 4451 ms
200	5193 - 9528 ms
500	17333 - 42990 ms
1000	29245 - 39581 ms

Και σε αυτή τη μέτρηση είχαμε παρόμοια προβλήματα όπως την προηγούμενη. Σφάλματα είχαμε εδώ στους 500 και στους 1000 χρήστες με το ποσοστό τους να φτάνει το 5% και 15% αντίστοιχα. Τα αιτήματα που εξυπηρετήθηκαν κατέγραψαν φθίνουσα πορεία όσο οι χρήστες αυξάνονταν. Ενδεικτικά αναφέρουμε ότι για 10 χρήστες σε 60 δευτερόλεπτα τα αιτήματα που εξυπηρετήθηκαν ήταν ~730, ενώ για 100 χρήστες ήταν ~590.

8.4 Συγκρίσεις – Ανάλυση αποτελεσμάτων - Συμπεράσματα

I. Εγκατάσταση / Παραμετροποίηση

Η εγκατάσταση των 3 web containers δεν παρουσίασε κάποια ιδιαίτερη δυσκολία. Μόνο στην περίπτωση των Tomcat και Jetty χρειάστηκε η ρύθμιση της πλήρους διαδρομής του JRE (με το Jetty να ζητάει το JDK και όχι απλώς το JRE). Από άποψη φιλικότητας προς το χρήστη, ο GlassFish παρουσιάζεται ανώτερος καθώς παρέχει ένα συνολικό και πλήρες περιβάλλον διαχείρισης μέσω browser, επιτρέποντας τη εφαρμογή απλών και προχωρημένων ρυθμίσεων, χωρίς τη παραμετροποίηση αρχείων μέσα στον server. Επίσης παρέχει και πλήρη εμφάνιση στατιστικών όσον αφορά τη λειτουργία του.

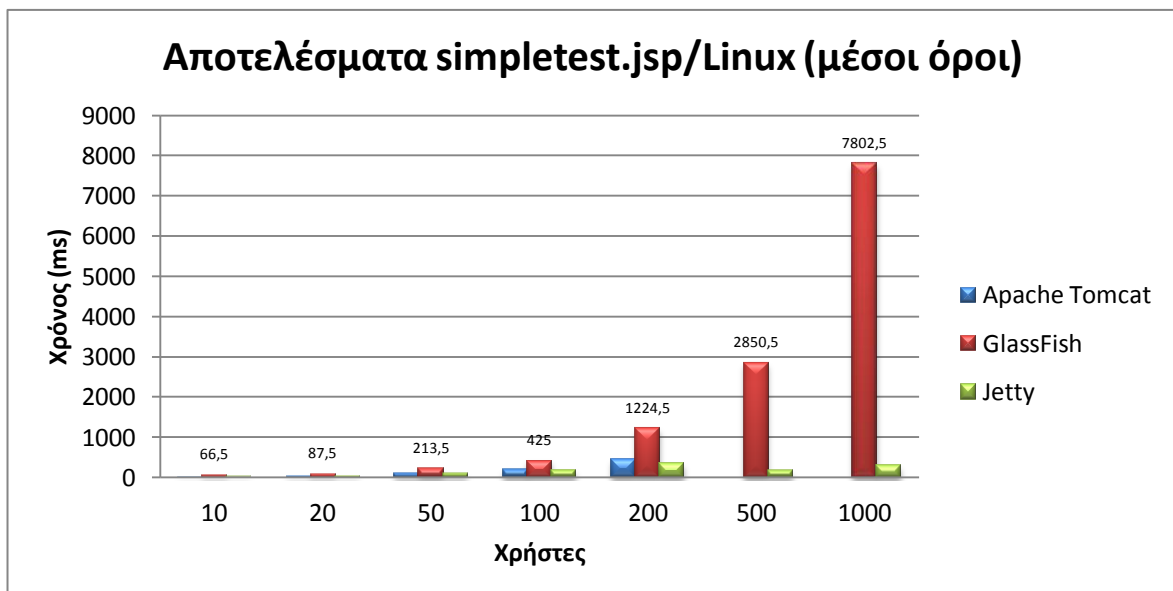
II. Ανάλυση των αποτελεσμάτων

Στην συνέχεια ακολουθούν τα συγκριτικά αποτελέσματα από τις μετρήσεις.

- **Simpletest.jsp**

- Μέσοι όροι

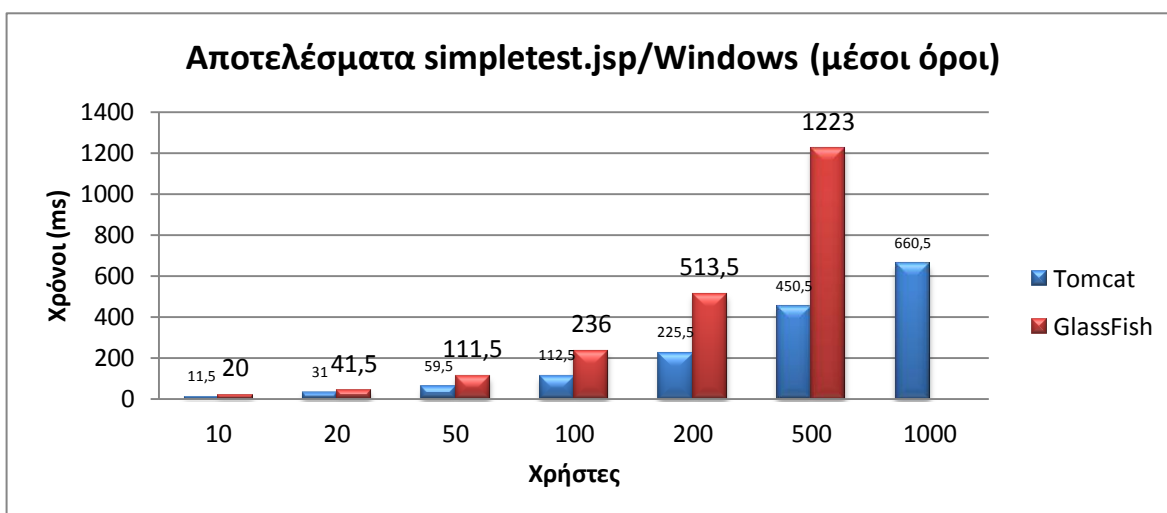
Ξεκινάμε με τις μετρήσεις σε περιβάλλον Linux.



Όπως βλέπουμε από το διάγραμμα πιο γρήγορος αναδείχτηκε ο Jetty, με μικρή διαφορά από τον δεύτερο Tomcat, ενώ ο GlassFish παρουσιάζεται πολύ πιο αργός, με την αύξηση των χρηστών να μεγαλώνει και την απόκλιση των μέσων όρων του από τους άλλους 2. Βέβαια δεν πρέπει να παραβλέψουμε το γεγονός ότι ο Tomcat εξυπηρέτησε λιγότερους χρήστες, προσθέτοντάς του ένα ισχυρό αρνητικό σημείο.

Μέτρηση απόδοσης Web Server

Ακολουθεί η αντίστοιχη μέτρηση σε Windows:



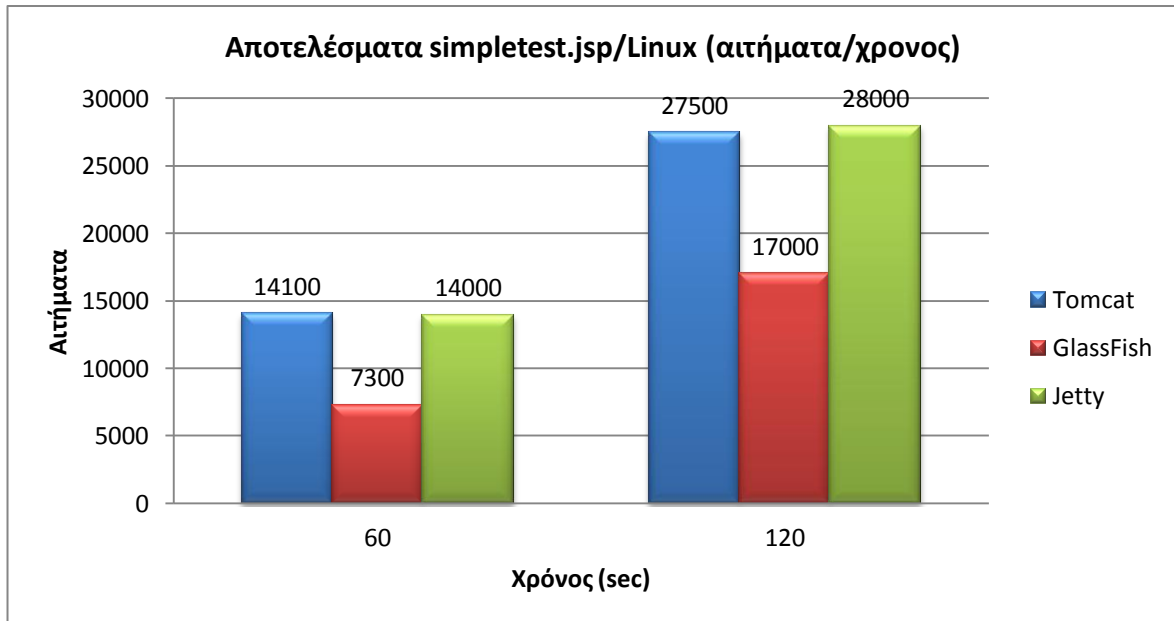
Το πρώτο που παρατηρούμε εδώ είναι η απουσία του Jetty, που όπως προαναφέρθηκε δεν ήταν εφικτό να καταγραφεί οποιοδήποτε αποτέλεσμα. Όσον αφορά τις μετρήσεις, ο Tomcat αποδείχτηκε και σε Windows αποδοτικότερος από τον GlassFish τόσο σε μέσους όρους όσο και ως προς του χρήστες που κατάφερε να εξυπηρετήσει.

Συγκρίνοντας τώρα τις μετρήσεις και στα δύο λειτουργικά συστήματα, βλέπουμε ότι οι μέσοι όροι είναι αρκετά μικρότεροι σε Windows και για τους 2 web containers. Επίσης στην περίπτωση του Tomcat εξυπηρετήθηκαν περισσότεροι χρήστες σε Windows. Αντίθετα, ο GlassFish ακολούθησε αντίστροφη πορεία σε αυτόν τον τομέα. Το άλλο μεγάλο αρνητικό των Windows είναι το πρόβλημα που δεν επέτρεψε τον Jetty να λειτουργήσει.

Μέτρηση απόδοσης Web Server

- Εξυπηρέτηση αιτημάτων σε 60 και 120 δευτερόλεπτα

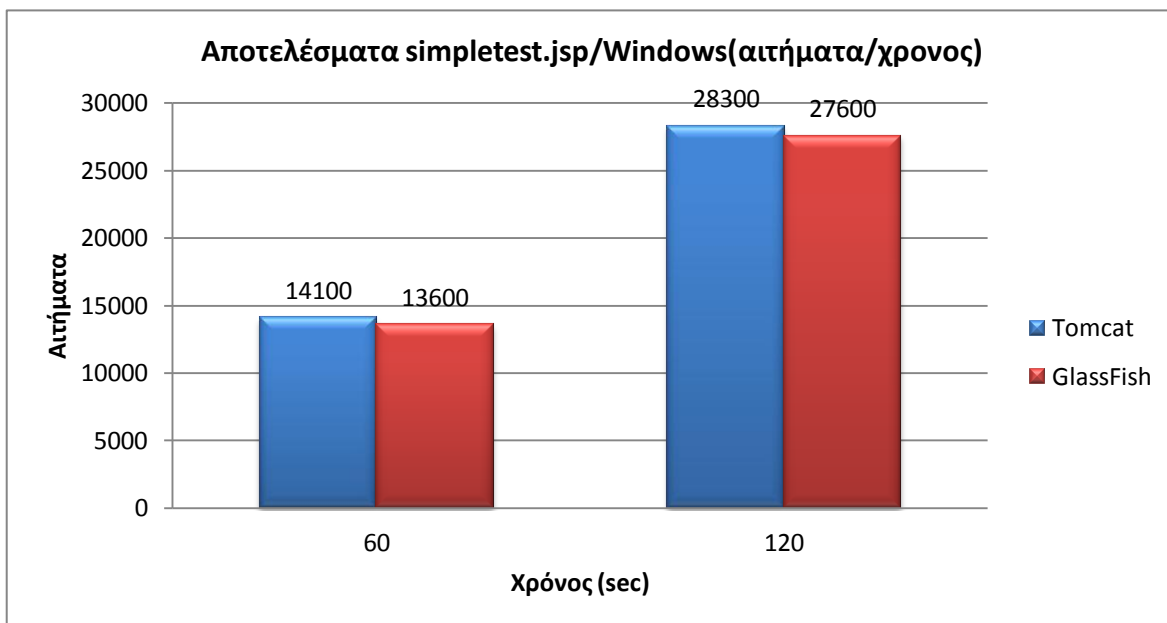
Ακολουθούν οι μετρήσεις που έγιναν σε περιβάλλον Linux



Όπως και στους μέσους όρους, έτσι και εδώ Tomcat και Jetty παρουσιάζουν παρόμοια αποτελέσματα, με μικρή διαφορά μεταξύ τους. Αντίθετα, ο GlassFish έμεινε αρκετά πίσω.

Η μέτρηση σε Windows κατέγραψε τα παρακάτω:

Μέτρηση απόδοσης Web Server



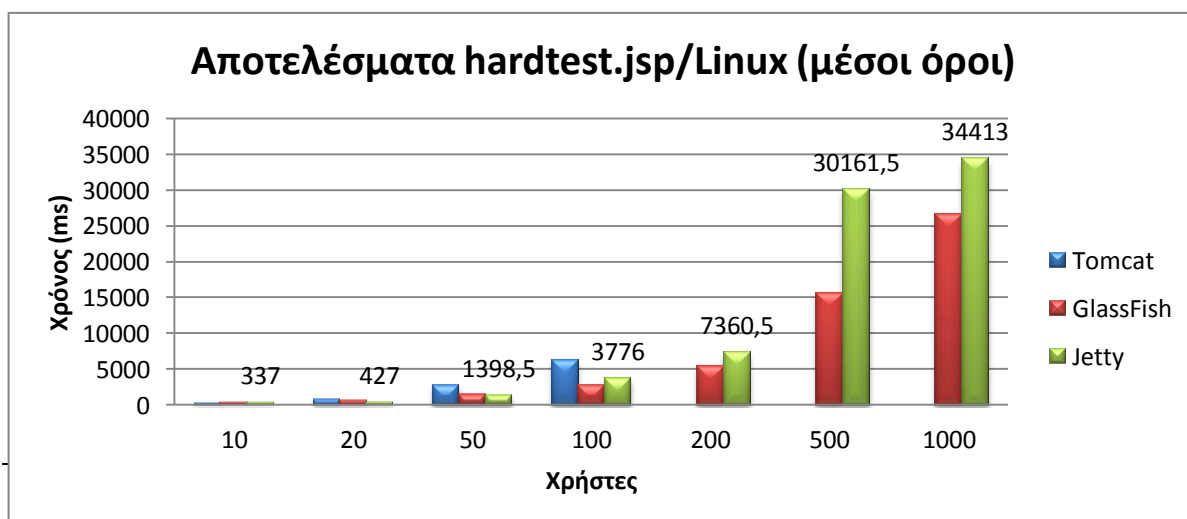
Εδώ βλέπουμε ότι ο Glassfish παρουσίασε καλύτερη απόδοση σε σχέση με την παραπάνω μέτρηση, χωρίς όμως να καταφέρει να ξεπεράσει τον Tomcat και μένοντας πίσω του με μικρή διαφορά

Παρατηρώντας τα αποτελέσματα, βλέπουμε ότι η μόνη μεγάλη διαφορά είναι η απόδοση του GlassFish η οποία σχεδόν διπλασιάστηκε σε περιβάλλον Windows. Ο Tomcat, από την άλλη δεν παρουσίασε σημαντικές διαφορές, με λίγο καλύτερη απόδοση σε Windows.

- **hardtest.jsp**

- Μέσοι όροι

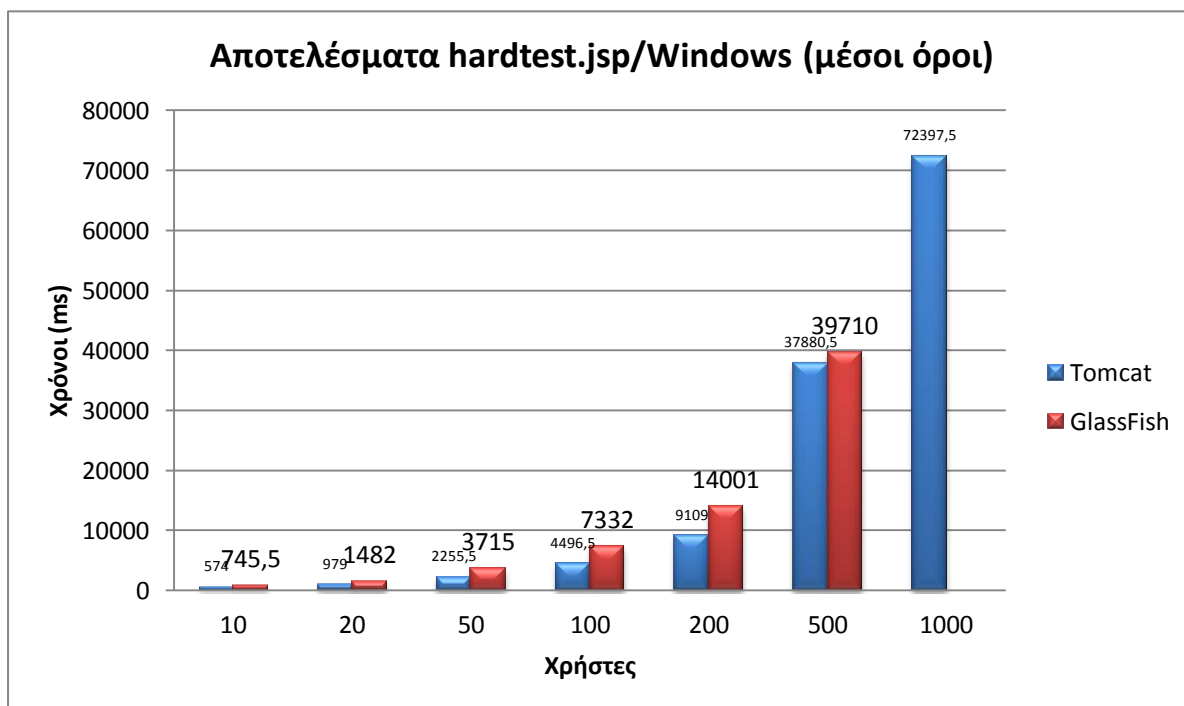
Οι μετρήσεις σε περιβάλλον Linux ήταν οι ακόλουθες:



Μέτρηση απόδοσης Web Server

Σε αυτή τη μέτρηση, προέκυψαν τα εξής ενδιαφέροντα στοιχεία. Στις πρώτες μετρήσεις ο Jetty έχει το προβάδισμα έναντι στους άλλους 2 web containers. Όμως τα πράγματα αλλάζουν στους 100 χρήστες και μετά, όπου ο Glassfish ανταποκρίνεται γρηγορότερα στα αιτήματα και με την διαφορά να αυξάνεται στις υπόλοιπες ομάδες. Επίσης ο Tomcat, εκτός ότι σε όλες τις μετρήσεις ήταν ο πιο αργός, δεν κατάφερε να εξυπηρετήσει πάνω από 100 χρήστες ενισχύοντας την ήδη αρνητική του επίδοση.

Ακολουθεί η αντίστοιχη μέτρηση σε Windows:



Φυσικά ούτε εδώ υπήρξε μέτρηση του Jetty για τους γνωστούς λόγους. Από τα αποτελέσματα βλέπουμε ότι ο Tomcat αποδίδει σχετικά καλύτερα από τον Glassfish και σε μέσους όρους αλλά και σε ποσότητα χρηστών που εξυπηρετήσε.

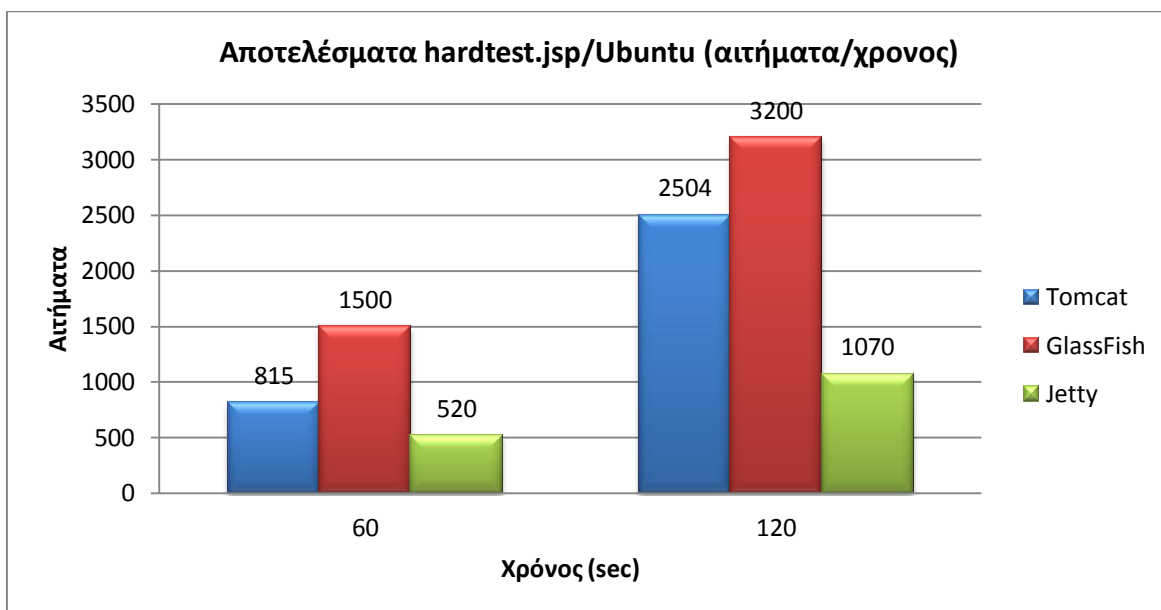
Αναλύοντας τώρα το αποτέλεσμα, μπορούμε να διαπιστώνουμε αρχικά ότι ο Tomcat λειτουργεί καλύτερα σε περιβάλλον Windows τόσο από άποψη των χρόνων απόκρισης όσο και από τους χρήστες που μπορεί να εξυπηρετήσει. Ο Glassfish από τη μεριά του, παρόλο που στις μετρήσεις του simpletest.jsp δεν έπιασε μεγάλη απόδοση, έδειξε ότι κάτω από συνθήκες όπου απαιτείται μεγαλύτερη επεξεργαστική ισχύ, μπορεί να ανταπεξέλθει ικανοποιητικά. Αυτό το

Μέτρηση απόδοσης Web Server

βλέπουμε στη μέτρηση που έγινε σε περιβάλλον Linux. Ο Jetty από την άλλη, φάνηκε να μην αντέχει στο αντίστοιχο βάρος, παρά μόνο για ένα μικρό ποσοστό χρηστών.

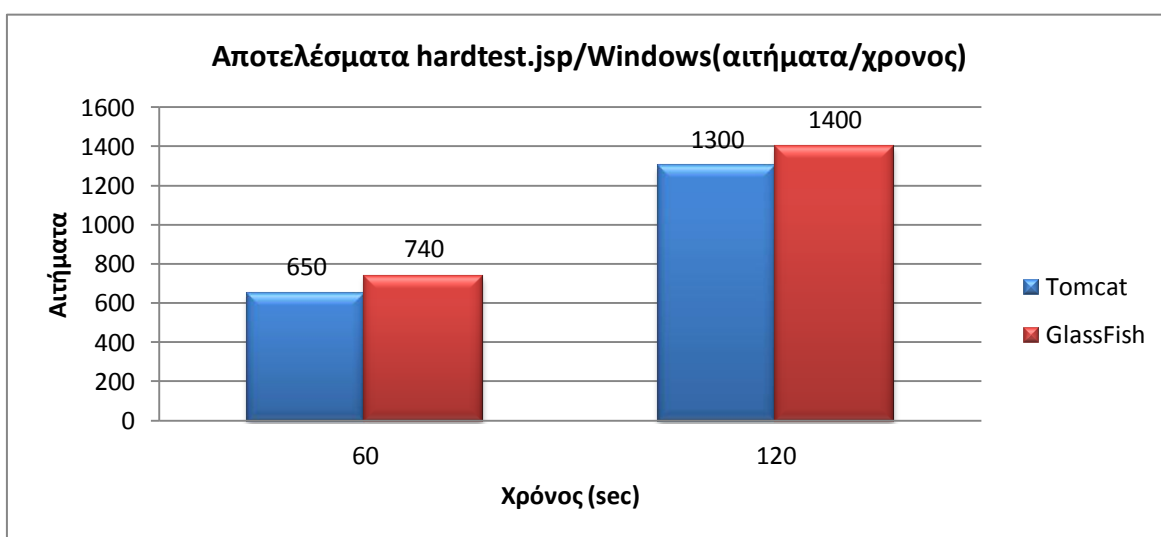
- Εξυπηρέτηση αιτημάτων σε 60 και 120 δευτερόλεπτα

Ακολουθούν οι μετρήσεις που έγιναν σε περιβάλλον Linux



Από το διάγραμμα βλέπουμε ότι ο GlassFish απέδωσε καλύτερα από τους άλλους 2 web containers, εξυπηρετώντας περισσότερα αιτήματα και σε απόσταση από τον δεύτερο Tomcat. Ο Jetty κινήθηκε σε χαμηλά επίπεδα.

Η μέτρηση σε Windows κατέγραψε τα παρακάτω:



Από το παραπάνω διάγραμμα, παρατηρούμε ότι ενώ ο Tomcat εξυπηρετεί πιο γρήγορα τα αιτήματα σύμφωνα με προηγούμενη μέτρηση, τα συνολικά αιτήματα που εξυπηρετεί είναι λιγότερα, χωρίς όμως μεγάλη διαφορά από τον GlassFish.

Συγκρίνοντας τώρα τα 2 αποτελέσματα, βλέπουμε ότι μέσω Linux εξυπηρετούνται περισσότερα αιτήματα, ακόμα και στην περίπτωση του Tomcat που έδειξε να λειτουργεί καλύτερα σε Windows. Επίσης, ο Glassfish σε περιβάλλον Linux έδειξε πιο ισχυρός, σε αντίθεση με τον Jetty που στην αναλογία μεγάλη επεξεργαστική ισχύ/αριθμός χρηστών έδειξε να μένει πίσω.

III. Συμπεράσματα

Έχοντας καταγράψει και αναλύσει τα αποτελέσματα μπορούμε τώρα να αναφερθούμε σε συγκεκριμένες διαπιστώσεις. Καταρχήν, σε καμία περίπτωση δεν μπορούμε να πούμε ότι υπάρχει ένας τέλειος web container. Όλοι έχουν τα θετικά και τα αρνητικά τους. Επομένως θα αναφερθούν κάποια συγκεκριμένα χαρακτηριστικά που διαπιστώθηκαν.

Κοιτώντας και τους 3 web containers, βλέπουμε ότι ανάλογα σε πιο λειτουργικό σύστημα «τρέχουν» εμφανίζουν διαφορετική απόδοση. Όπως ο Tomcat ο οποίος είχε αρκετά χειρότερη επίδοση σε περιβάλλον Linux και στους 2 δείκτες. Ο Glassfish, από τη μεριά του, είχε διαφορετική συμπεριφορά ανάλογα και με το περιβάλλον αλλά και με τον αριθμό των χρηστών.

Σε ένα σημείο που ο GlassFish υπερέχει σύμφωνα με τα παραπάνω, είναι η σταθερή του απόδοση κάτω από συνθήκες φόρτισης. Στην μέτρηση του απλού αρχείου simpletest.jsp έμεινε αρκετά πίσω, αλλά στο hardtest.jsp έδειξε μπορεί να ανταποκριθεί καλύτερα. Και στην περίπτωση που έμεινε πίσω από τον Tomcat (hardtest.jsp/Windows) η διαφορά ήταν μικρή.

Όσον αφορά τον Jetty, οι επιδόσεις του δεν περνάν απαρατήρητες, αλλά έκανε φανερό ότι θέλει αρκετή βελτίωση ακόμα μέχρι να καταφέρει να διεκδικήσει ένα μεγαλύτερο στον κόσμο του διαδικτύου. Κάτι που δεν φαίνεται ανέφικτο από τη στιγμή που οι επιδόσεις ήταν οι καλύτερες στη μέτρηση του simpletest.jsp. Τα προβλήματά του εντοπίζονται στην δυσκολία του να εξυπηρετήσει ικανοποιητικά

Μέτρηση απόδοσης Web Server

πολύπλοκα αρχεία κάτω από μεγάλο φόρτο και στο ότι η έκδοση για Windows είναι προβληματική, μην μπορώντας να ανταγωνιστεί τους άλλους 2 web containers σε αυτό το περιβάλλον.

Τέλος, σχετικά με τις δυνατότητες ρυθμίσεως τους, ο GlassFish φαίνεται να έχει το προβάδισμα, λόγω του (σχεδόν) πλήρους διαχειριστικού περιβάλλοντος που παρέχει, φιλικό προς έναν άπειρο χρήστη. Ο Tomcat και ο Jetty, σε αυτόν το τομέα, απευθύνονται σε πιο ειδικευμένο και έμπειρο κοινό.

Επίλογος

Οι web server και οι web containers αποτελούν ένα μεγάλο, αφανή κομμάτι του διαδικτύου, που κάνουν εφικτή την εξάπλωση του στον κόσμο. Ο καθένας δημιουργήθηκε από διαφορετικές ομάδες, με διαφορετικό τρόπο αλλά για να εξυπηρετήσει τον ίδιο σκοπό. Σκοπός του καθενός είναι να δημιουργήσει ένα πρόγραμμα σταθερό, αξιόπιστο και αποδοτικό που θα κατακτήσει τον κόσμο του διαδικτύου. Όσο για το ερώτημα ποιος είναι ο καλύτερος web server ή web container; Η απάντηση είναι ότι καλύτερος είναι αυτός που εξυπηρετεί καλύτερα τις ανάγκες μας και αυτά που θέλουμε να πετύχουμε.

Πιο συγκεκριμένα, μπορούμε να θεωρήσουμε τα παρακάτω σύμφωνα με τα αποτελέσματα. Αρχίζουμε από τους web server. Εάν θέλουμε ένα web server ο οποίος συνδυάζει λειτουργικότητα και σταθερότητα σε συνδυασμό με την ευκολία ρύθμισης τότε η επιλογή μας θα είναι ο Apache. Οι επιδόσεις του σε συνδυασμό με την δυνατότητα που δίνει για ρύθμιση και στον άπειρο διαχειριστή τον κάνει κορυφαίο στο είδος του. Οι υπόλοιποι web server δεν υπολείπονται πολύ σε επιδόσεις αλλά χάνουν στον τομέα των ρυθμίσεων, τόσο λόγω πολυπλοκότητας όσο και στο γεγονός ότι δεν υπάρχουν επαρκείς οδηγίες για αυτό (τουλάχιστον όχι τόσες όσες υπάρχουν για τον Apache). Τα παραπάνω αφορούν τις γλώσσες PHP, Perl και HTML. Για την περίπτωση της ASP.Net είναι σαφές ότι μόνο σε IIS θα μπορέσουμε να έχουμε το καλύτερο δυνατό αποτέλεσμα, και φυσικά σε περιβάλλον Windows, καθώς η έκδοση του εν λόγω web server υπάρχει μόνο για αυτό το λειτουργικό. Από την άλλη μεριά, όλες οι άλλες υλοποιήσεις φαίνονται να λειτουργούν καλύτερα σε Linux. Για τους web containers, προβάδισμα σε όλους τους τομείς φαίνεται να έχει ο GlassFish. Αυτό οφείλεται κυρίως στους τομείς της σταθερότητας και της ρύθμισης, που μαζί με τις επιδόσεις του, τον κάνουν έναν αξιόπιστο web container

Παράρτημα

Κώδικες αρχείων μέτρησης

- Simpletest.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
```

```
<html>
```

```
<head>
```

```
    <title>Webserver test</title>
```

```
</head>
```

```
<body>
```

```
    This is a test page.
```

```
</body>
```

```
</html>
```

- Hardtest.htm

```
<html>
```

```
<head>
```

```
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <meta name="GENERATOR" content="Quadralay WebWorks Publisher 5.0.4">
    <meta name="TEMPLATEBASE" content="Dynamic HTML">
    <meta name="LASTUPDATED" content="10/30/01 09:40:03">
    <link rel="StyleSheet" href="standard.css" type="text/css" media="screen">
    <title>HTTP/S Load</title>
```

```
</head>
```

```
<body background="images/backgrnd.gif">
```

```
<p> </p>
```

```
<table width="331" border="0" align="right" cellpadding="0" cellspacing="0">
```

```
<tr>
```

```
    <td><a href="index.htm"> </a></td>
```

```
    <td><a href="os-getti.htm"> </a></td>
```

```
    <td><a href="os-worki.htm"> </a></td>
```

```
    <td><a href="openstai.htm"> </a></td>
```

```
</tr>
```

```
</table>
```

```
<p><br clear="all"></p>
```

```
<hr align="left">
```

```
<blockquote>
```

```
<h2 class="NewHTMLPage">
```

```
    <a name="40499"> </a>HTTP/S Load
```

```
</h2><hr>
```

```
<ul>
```

```
    <li class="SmartList1"><a name="93282"> </a><a href="os-overv.htm#57905">Overview of HTTP/S Load</a>
```

```
    <li class="SmartList1"><a name="93286"> </a><a href="os-overv.htm#58289">Core Functions of HTTP/S Load</a>
```

```
    <li class="SmartList1"><a name="93290"> </a><a href="os-overv.htm#58489">Using HTTP/S Load</a>
```

```
    <li class="SmartList1"><a name="93294"> </a><a href="os-overv.htm#48137">The Commander Interface</a>
```

```
    <li class="SmartList1"><a name="93298"> </a><a href="os-overv.htm#23582">Commander Toolbars and Function Bars</a>
```

```
    <li class="SmartList1"><a name="93302"> </a><a href="os-overv.htm#23605">The Commander Main Window</a>
```

```
    <li class="SmartList1"><a name="93306"> </a><a href="os-overv.htm#29225">The Repository Window</a>
```

```
</ul>
```

```
<h2 class="Heading1">
```

```
    <a name="57905"> </a>Overview of HTTP/S Load
```

```
</h2>
```

```
<p class="Body">
```

```
    <a name="76480"> </a>HTTP/S Load supplies flexible software that enables you to quickly develop and run HTTP/S load Tests and production monitoring Tests, to help you assess the performance of Web Application Environments (WAEs).
```

```
</p>
```

```
<p class="Body">
```

[HTTP/S Load](#) is comprised of several [Modules](#) including the OpenSTA Architecture and the Commander GUI which runs within it. Use Commander to initiate and control the Test development process, including Script creation, Collector creation, Test creation, running Tests, monitoring Test-runs and displaying the results data for analysis.

[HTTP/S Load](#) combines HTTP/S recording and Script modeling functionality, using the Script Modeler Module, with Test creation and system data collection. It records browser requests issued during a Web session at the HTTP/S level, rather than recording the real time events of a browser, in order to create Scripts. This allows you to create and run load Tests, incorporating Scripts, that use minimum system resources enabling you to carry out large volume load Tests.

In Commander, a Test is represented as a table known as the Test Pane. This is the workspace where you can develop the contents of a Test by adding the Scripts and the Collectors you need from the Repository. Select them individually working from the Repository Window, then drag and drop them into the Test Pane in the required order.

Collectors are used to monitor and record performance data during a Test-run. They contain user-defined data collection queries and monitoring options that control the data collected from [Host](#) computers and other target devices during a Test-run.

Scripts are created from the recordings of HTTP/S browser requests issued during a Web session and written in SCL scripting language, which enables you to model their content. They encapsulate the Web activity you want to simulate during a Test-run and enable you to generate the load levels required against target WAEs by controlling the number of [Virtual Users](#) who run them.

Tests can be comprised of one or more Collectors, one or more Scripts or a combination of both, depending on whether you are performance testing a system within a development or a production environment. It is possible to modify a load Test to monitor the same target system in a production scenario by disabling the Scripts it includes so that no load is generated when the Test is run.

During a Test-run you can monitor Task Group activity from the Monitoring tab of the Test Pane. The results collected can be displayed as they are returned to the Repository while a Test is running or after a Test-run is complete, to assist you in the analysis of the target WAE performance.

See also:

[Core Functions of HTTP/S Load](#)

Core Functions of HTTP/S Load

[HTTP/S Load](#) supplies versatile software that caters for the needs of different users and the type of system you are evaluating, by supplying the full range of functions that e-business project managers and system performance testers need in order to develop transparent, easy to maintain Tests.

[HTTP/S Load](#) is a modular software system in which the creation of Scripts, Collectors and Tests are separate processes that can be conducted independently. It provides the functionality required to support the tasks you need to conduct, in order to achieve the objectives of your performance tests.

All Test development procedures are initiated from Commander. Use it to create Tests and to coordinate the development process.

Performance Testing Using HTTP/S Load

- [Create Scripts](#) ([Script Modeler](#)).

- [Model Scripts if required \(Script Modeler\).](#)

<li class="SmartList2" value="3"> Create data collection Collectors - optional (SNMP, NT Performance).

<li class="SmartList2" value="4"> Create Tests, by adding Task Groups containing the Scripts and Collectors required (Commander).

<li class="SmartList2" value="5"> Define Task Group settings (Commander), including:

- <li class="SmartList1"> Schedule settings to control when Task Groups start and stop during a Test-run.
- <li class="SmartList1"> Host computers used to run a Task Group: Script and Collector-based Task Groups.
- <li class="SmartList1"> Number of Virtual Users used: Script-based Task Groups only.
- <li class="SmartList1"> Task settings control the number of Script iterations and the delay between iterations during a Test-run: Script-based Task Groups only.

- <li class="SmartList2" value="6"> Run a Test (Commander).
- <li class="SmartList2" value="7"> Monitor a Test-run (Commander).
- <li class="SmartList2" value="8"> Display Test results (Commander).

<p class="Body">

</p>

<p class="Body">

 Note: It is not necessary to stick rigidly to this procedural sequence.

</p>

<p class="Body">

 HTTP/S Load supplies flexible software that enables you to work in ways that best suit you and the type of Test you are creating.

</p>

<h6 class="Heading5">

 See also:

</h6>

<p class="Body">

 Using HTTP/S Load

</p>

<h2 class="Heading1">

 Using HTTP/S Load

</h2>

<p class="Body">

 The main areas of procedure supported by HTTP/S Load are summarized below:

</p>

- <li class="SmartList1"> Creating Scripts
- <li class="SmartList1"> Modeling Scripts
- <li class="SmartList1"> Creating Collectors
- <li class="SmartList1"> Creating Tests
- <li class="SmartList1"> Running and Monitoring Tests
- <li class="SmartList1"> Displaying Results

<h6 class="Heading5">

 See also:

</h6>

<p class="Body">

 The Commander Interface

</p>

<h3 class="Heading2">

 Creating Scripts

</h3>

<p class="Body">

Μέτρηση απόδοσης Web Server

Creating Scripts involves deciding how you expect clients to use the WAE under test, then recording browser sessions which incorporate this behavior to produce Scripts. Scripts encapsulate the browser requests issued during a Web session at the HTTP/S level and form the basis of your Tests.

Browser requests and WAE responses are recorded using the OpenSTA Gateway. It is launched automatically when you begin recording a Script using the Script Modeler Module. The Gateway records the HTTP/S requests issued by a browser during Web sessions using SCL scripting language, which enables you to model their content.

Creating Scripts is a separate procedure within the Test development process, and can be carried out independently of Test and Collector creation. For more information see Recording Scripts.

See also:

Modeling Scripts

HTTP/S Scripts

Modeling Scripts

Modeling Scripts involves identifying and editing SCL code that represents user input during a browser session, so that the Scripts can be used in Tests to function as one or more Virtual Users during a Test-run. Modeling Scripts enables you to develop Tests that more accurately simulate the Web activity you want to reproduce during a Test-run.

Modeling Scripts is not an essential procedure, particularly if the WAE under test comprises static content only. But it is a useful facility if you need to record the dynamic changes of a WAE during a session. For example, you may need to use a unique user name and password for each Virtual User, so that the Test more accurately simulates real end user activity. You can achieve this by creating a Script then modeling it to include variables that change the user name and password for each Virtual User, every time the Script is run as part of a Test. Using just one modeled Script it is possible to create all the Virtual Users you need, each with unique identities just like real end users.

Script Modeling is enhanced beyond the addition of variables to a Script. The Web pages issued in response to browser requests are recorded at the same time as a Script is created. In HTTP/S Load there is the capability to use objects from these Web pages to model the corresponding Script. This modeling technique is known as DOM Addressing. This technique can be used to verify the results of a Test by checking the validity of WAE responses during Test-run. For more information see Recording Scripts and Modeling Scripts.

See also:

Creating Collectors

HTTP/S Scripts

Creating Collectors

Creating Collectors involves deciding which Host computers or other devices to collect performance data from and the type of data to collect during a Test-run. HTTP/S Load supports the creation of NT Performance for recording performance data from Hosts

running Windows NT or Windows 2000, and SNMP Collectors for collecting SNMP data from Hosts and other devices running an SNMP agent or proxy SNMP agent.

</p>

<p class="Body">

 Collector-based Task Groups can be monitored during a Test-run. The data collected can be displayed alongside other results to provide information about a Test-run.

</p>

<p class="Body">

 Creating Collectors is a separate procedure within the Test development process and can be carried out independently of Test and Script creation. For more information see Creating and Editing Collectors.

</p>

<h6 class="Heading5">

 See also:

</h6>

<p class="Body">

 Creating Tests

</p>

<h3 class="Heading2">

 Creating Tests

</h3>

<p class="Body">

 Creating Tests first involves creating the Scripts and Collectors you want to include in them. Then select the Scripts and Collectors you need from the Repository Window and add them one at a time to a Test. Scripts and Collectors are included in Tests by reference. This means that you can include them in multiple Tests in which different Task Group settings apply.

</p>

<p class="Body">

 The Scripts and Collectors you add are known as Tasks which are structured in Script-based and Collector-based Task Groups. A load Test must contain at least one Script-based Task Group which can include one, or a sequence of Scripts. Collector-based Task Groups are optional.

</p>

<p class="Body">

 Create and run Collector-only Tests for performance monitoring and data collection within production scenarios. Or alternatively, use a load Test that includes Collectors and disable the Script-Task Groups it includes, to turn off the load element they supply before running the Test within a production monitoring environment.

</p>

<p class="Body">

 The Test scenario you want to simulate during a Test-run can be controlled by adjusting the Task Group settings. Assemble the Scripts and Collectors of your Test then select the Task Group settings you want to apply in order to generate the level of load required. For Script-based Task Groups these settings include the Host used, the number of Virtual Users and the number of Script iterations per Virtual User. For Collector-based Task Groups the Host used to run the Task Group can be defined.

</p>

<p class="Body">

 Creating Tests is a separate procedure within the Test development process, and can be carried out independently of Script and Collector creation. For more information see Creating and Editing Tests.

</p>

<h6 class="Heading5">

 See also:

</h6>

<p class="Body">

 Running and Monitoring Tests

</p>

<h3 class="Heading2">

 Running and Monitoring Tests

</h3>

<p class="Body">

 Running a Test enables you to imitate real end user Web activity and accurately simulate the test conditions you want in order to generate the level of load required against target WAEs.

</p>

<p class="Body">

 The Task Groups that comprise a Test can be run on remote Hosts during a Test-run. Distributing Task Groups across a network enables you to run Tests that generate realistic heavy loads simulating the activity of many users.

</p>
<p class="Body">
 You can monitor the progress of a Test-run by selecting a Script-based Task Group and tracking the Scripts and the Virtual Users that are currently running from the Monitoring tab of the Test Pane.
</p>
<p class="Body">
 You can add Collector-based Task Groups to a Test which can be monitored by selecting the data collection queries defined in the Collector and displaying them in graphs. For more information see Running Tests.
</p>
<h6 class="Heading5">
 See also:
</h6>
<p class="Body">
 Displaying Results
</p>
<p class="Body">
 Distributed Architecture
</p>
<h3 class="Heading2">
 Displaying Results
</h3>
<p class="Body">
 Running a Test then displaying the results enables you to analyze and assess whether WAEs will be able to meet the processing demands that will be placed on them.
</p>
<p class="Body">
 HTTP/S Load provides a variety of data collection and display options to assist you in the analysis of Test results. When a Test is run a wide range of data is generated automatically. It enables you to collect and graph both Virtual User response times and resource utilization information from all WAEs under test, along with performance data from the Hosts used to run the Test.
</p>
<p class="Body">
 After a Test-run is complete the results can be displayed. Open the Test you want from the Repository Window and click on the Results tab in the Test Pane, then select the results you want to display. For more information see Results Display.
</p>
<h6 class="Heading5">
 See also:
</h6>
<p class="Body">
 The Commander Interface
</p>
<p class="Body">
 Creating and Editing Collectors
</p>
<h2 class="Heading1">
 The Commander Interface
</h2>
<p class="Body">
 Commander combines an intuitive user interface with comprehensive functionality to give you control over the Test development process, enabling you to successfully create and run performance Tests.
</p>
<p class="Body">
 Use the menu options or work from the Repository Window to initiate the creation of Collectors, Scripts and Tests. Right-click on the Repository Window folders and choose from the functions available.
</p>
<p class="Body">
 Work within the Main Window of Commander to create Collectors and Tests. The Main Window houses the Repository Window and supplies the workspace for Test creation using the Test Pane, and Collector creation using the Collector Pane. Use Script Modeler to create the Scripts you need.
</p>


```
<p class="Body">
  <a name="93548"> </a><a href="os-overv.htm#62884">Hide/Display Toolbars</a>
</p>
<p class="Body">
  <a name="93552"> </a><a href="os-overv.htm#23605">The Commander Main Window</a>
</p>
<h5 class="Heading4">
  <a name="62884"> </a>Hide/Display Toolbars
</h5>
<ul>
  <li class="SmartList1"><a name="62885"> </a>Click <strong class="Strong">View &gt; Toolbar</strong>.
  <dl>
    <dt class="Indented2"> <a name="62886"> </a>A tick to the left of the toolbar listed indicates that it is currently
    displayed.
  </dt>
  </dl>
</li>
</ul>
<h2 class="Heading1">
  <a name="23605"> </a>The Commander Main Window
</h2>
<p class="Body">
  <a name="23606"> </a>The Commander Main Window is located below the Menu Bar and functions as a workspace and
  container for the creation of Tests and data collection Collectors.
</p>
<p class="Body">
  <a name="64240"> </a>The Test Pane is displayed here when you open a Test by double-clicking a Test icon, 
  or 
  , in the Repository Window. Use the Test Pane to create and edit Tests, then run a Test and monitor its progress. When
  results are returned they can be displayed here for analysis. For more information, see <span style="color: #000000; font-
  style: normal; font-weight: normal; text-decoration: none; text-transform: none; vertical-align: baseline"><a href="os-
  creaa.htm#106934">The Test Pane</a></span>.
</p>
<p class="Body">
  <a name="65422"> </a>The Collector Pane is displayed in the Main Window when you open a Collector by double-clicking
  a Collector icon, 
  , 
  (NT&nbsp;Performance), and 
  , 
  (SNMP), in the Repository Window. Use this workspace to create and edit Collectors. For more information, see <span
  style="color: #000000; font-style: normal; font-weight: normal; text-decoration: none; text-transform: none; vertical-align:
  baseline"><a href="os-creat.htm#3423">Creating and Editing Collectors</a></span>.
</p>
<p class="Body">
  <a name="65447"> </a>The Repository Window is displayed in the Commander Main Window. Use it to initiate the
  creation of the Scripts, Collectors and Tests by right-clicking on the default folders within and selecting the menu options
  you need.
</p>
<h6 class="Heading5">
  <a name="93601"> </a>See also:
</h6>
<p class="Body">
  <a name="93605"> </a><a href="os-overv.htm#23621">Commander Main Window Display Options</a>
</p>
<p class="Body">
  <a name="93609"> </a><a href="os-overv.htm#29225">The Repository Window</a>
</p>
<h5 class="Heading4">
  <a name="23621"> </a>Commander Main Window Display Options
</h5>
<ul>
  <li class="SmartList1"><a name="23622"> </a>Resize the Main Window to increase your workspace area by adjusting the
  borders of the Repository Window.
  <dl>
    <dt class="Indented2"> <a name="29203"> </a>Click on the right-hand border of the Repository Window then drag the
    border to the desired position.
  </dt>
    <li class="SmartList1"><a name="29078"> </a>Float the Repository Window over the Main Window to increase the
    workspace to the full width of the Main Window.
  <dl>
    <dt class="Indented2"> <a name="29207"> </a>Click on the double bar at the top of the Repository Window then drag
    and drop it in the new location.
  </dt>
  </li>
  <li class="SmartList1"><a name="29063"> </a>Close the Repository Window to maximize the workspace area.
  <dl>
    <dt class="Indented2"> <a name="29220"> </a>Click 
  </dt>
  </li>
</ul>
```

Μέτρηση απόδοσης Web Server

, in the top right-hand corner of the window, or select **Tools > Show Repository**.

`<dt class="Indented2"> To display the Repository Window select Tools > Show Repository</dt>`

`</dt>`

`<li class="SmartList1"> Resize the Main Window by adjusting the borders.`

`<dl>`

`<dt class="Indented2"> Click on the border of a window and drag it to the required position.`

`</dt>`

``

`<h2 class="Heading1">`

` The Repository Window`




`</h2>`



`<p class="Body">`


` After you have planned your performance Test you can work from the Repository Window to initiate Test development procedures, including the creation of Scripts, Collectors and Tests. The Repository Window displays the contents of the Repository which stores all the files that define a Test. Use the Repository Window to manage the contents of the Repository by creating, displaying, editing and deleting Collectors, Scripts and Tests.`

`</p>`

`<p class="Body">`

` The Repository Window is located on the left-hand side of the Main Window by default and displays the contents of the Repository in three predefined folders  Collectors,  Scripts, and  Tests.`

` <strong class="Strong">Collectors,  Scripts, and  Tests.`

`<strong class="Strong">Scripts, and  Tests.`

`<strong class="Strong">Tests. These folders organize the contents of the Repository into a directory structure which you can browse through to locate the files you need. Double-click on the predefined folders to open them and display the files they contain. These folders have functional options associated with them, which you can access by right-clicking on the folders. They present separate right-click menus which contain options for creating new Collectors, Scripts and Tests.`

`</p>`

`<p class="Body">`

` When you double-click on a Test or Collector in the Repository Window, they are opened in the Commander Main Window, where they can be developed or edited. Double-click on a Script in the Repository Window to open it using the Script Modeler. This Module is launched in a separate window where you can create and model Scripts.`

`</p>`

`<p class="Body">`

` The Scripts, Collectors and Tests stored in the Repository are organized alphabetically and can be deleted by using the right-click menu option associated with each file or by using the keyboard.`

`</p>`

`<p class="Body">`

` The order and appearance of the predefined folders, Collectors, Scripts and Tests, cannot be modified.`


`</p>`

`<h6 class="Heading5">`

` Repository Path`

`</h6>`

`<p class="Body">`

` When you first run Commander the Repository that was automatically created in the default location within the program directory structure is displayed, which appears as  Repository.`

` <strong class="Strong">Repository. Additional Repositories can be created using Commander that can be located on your hard drive or on a networked computer.`

`</p>`

`<p class="Body">`

` We recommend changing the location of the Repository, especially if you expect to generate large volumes of Test related files, so that the performance of your computer is not compromised. Use the Select a New Repository Path option in the Tools menu to create a new Repository or change the path.`

`</p>`

`<h6 class="Heading5">`

` See also:`

`</h6>`

`<p class="Body">`

` Select a New Repository Path`

`</p>`

`<p class="Body">`

` Repository Window Display Options`

`</p>`

`<p class="Body">`

` The Commander Interface`

`</p>`

`<p class="Body">`

` Collectors Folder`

`</p>`

`<p class="Body">`

` Scripts Folder`

`</p>`

`<p class="Body">`

[Tests Folder](#)

Collectors Folder

The Collectors folder in the Repository Window displays all the Collectors stored in the Repository and has a right-click menu option associated with it that enables you to create new Collectors.

Open the Collectors folder and display the Collectors contained by double-clicking  **Collectors**.

See also:

[Collectors Folder and Collectors, Display Options and Functions](#)

[Scripts Folder](#)

[Tests Folder](#)

[Creating and Editing Collectors](#)

Collectors Folder and Collectors, Display Options and Functions

- Double-click  **Collectors** in the Repository Window, to expand or collapse the directory structure, or click , or , alongside the folder.
- Right-click  **Collectors**, to display a pop-up menu from where you can choose to create a **New Collector**, either **SNMP** or **NT Performance**.
- Double-click on a new Collector  (NT Performance) or  (SNMP), to open it in the Collector Pane in the Commander Main Window and define the performance data to be collected during a Test-run.

Note: When you first create a Collector no data collection queries have been defined, so it appears with a small crossed red circle over the Collector icon to indicate this,  or . After you have defined your data collection queries, the cross is removed,  **(NT Performance)** or  **(SNMP)**.

Double-click on a Collector  or , to open it in the Collector Pane in the Commander Main Window and make any edits required to the files.

Note: When a Collector is open in the Test Pane, the Collector icon in the Repository Window appears with a small, yellow lock icon overlaid, . This makes it easy to spot which Collector or Test is currently open in Commander. The name of the open Collector or Test is displayed in the Commander Title bar.

Right-click on a Collector  or , to display a pop-up menu which gives you the option to **Rename** or **Delete** the Collector from the Repository.

Note: If a Collector is open , it cannot be renamed or deleted.

Note: Collectors are included in Tests by reference so editing their data collection queries affects the type of results recorded during a Test-run for all the Tests that use them. **Renaming a Collector or deleting one from the Repository means that the Tests using them cannot run. A missing Collector is still**

Μέτρηση απόδοσης Web Server

referenced in a Task Group and the altered status of the Collector Task is indicated within the Configuration tab of an open Test by highlighting in red the cell it occupies in the `Test table`. Tests can only run if a missing Collector is recreated, an existing Collector is renamed, or the Collector Task is deleted from the Task Group.

```
</dl>
</ul>
<h6 class="Heading5">
  <a name="93678"> </a>See also:
</h6>
<p class="Body">
  <a name="93680"> </a><a href="os-creat.htm#3423">Creating and Editing Collectors</a>
</p>
<p class="Body">
  <a name="93684"> </a><a href="os-overv.htm#87696">Collectors Folder</a>
</p>
<h3 class="Heading2">
  <a name="85726"> </a>Scripts Folder
</h3>
<p class="Body">
  <a name="29679"> </a>The Scripts folder in the Repository Window displays all the Scripts stored in the Repository and has a right-click menu option associated with it that enables you to create new Scripts.
</p>
<p class="Body">
  <a name="50305"> </a>Open the Scripts folder and display your Scripts by double-clicking 
<strong class="Strong">&nbsp;Scripts</strong>.
</p>
<h6 class="Heading5">
  <a name="93689"> </a>See also:
</h6>
<p class="Body">
  <a name="93693"> </a><a href="os-overv.htm#50307">Scripts Folder and Scripts, Display Options and Functions</a>
</p>
<p class="Body">
  <a name="93697"> </a><a href="os-overv.htm#85465">Tests Folder</a>
</p>
<p class="Body">
  <a name="93701"> </a><a href="os-overv.htm#87696">Collectors Folder</a>
</p>
<h5 class="Heading4">
  <a name="50307"> </a>Scripts Folder and Scripts, Display Options and Functions
</h5>
<ul>
  <li class="SmartList1"><a name="50317"> </a>Double-click 
<strong class="Strong">Scripts</strong> in the Repository Window, to expand or collapse the directory structure or click

, or 
, alongside the folder.
  <li class="SmartList1"><a name="23724"> </a>Right-click 
<strong class="Strong">Scripts</strong>, to display a pop-up menu from where you can create a <strong
class="Strong">New&nbsp;Script &gt; HTTP</strong>.
  <li class="SmartList1"><a name="23728"> </a>Double-click on a new Script 
, to launch Script Modeler and record a web browser session.
  <dl>
    <dt class="Indented2"> <a name="23732"> </a>Note: When you first create a Script in Commander it contains no data because no HTTP/S recording has been performed, so it appears with a small crossed red circle over the Script icon to indicate this 
. <br>After you have recorded the Script, the cross is removed, 
.
  </dl>
  <li class="SmartList1"><a name="23742"> </a>Double-click on a Script 
, to launch Script Modeler and model the Script.
  <li class="SmartList1"><a name="23749"> </a>Right-click on a Script 
, to display a pop-up menu which gives you the option to <strong class="Strong">Rename</strong> or <strong
class="Strong">Delete</strong> the Script from the Repository.
  <dl>
    <dt class="Indented2"> <a name="29728"> </a>Note: Scripts are included in Tests by reference, so modeling them affects the type of web browser activity generated during a Test-run for all the Tests that use them.
    <dt class="Indented2"> <a name="85461"> </a>Renaming a Script or deleting one from the Repository means that the Tests using them cannot run. A missing Script is still referenced in a Task Group and the altered status of the Script Task is indicated within the Configuration tab of an open Test by highlighting in red the cell it occupies in the <span style="color: #000000; font-style: normal; font-weight: normal; text-decoration: none; text-transform: none; vertical-align: baseline"><a href="os-gloss.htm#1006729">Test table</a></span>. Tests can only run if a missing Script is recreated, an existing Script is renamed, or the Script Task is deleted from the Task Group.
```

</dl>

<h6 class="Heading5">
 See also:
</h6>
<p class="Body">
 Scripts Folder
</p>
<p class="Body">
 Working With Scripts
</p>
<h3 class="Heading2">
 Tests Folder
</h3>
<p class="Body">
 The Tests folder in the Repository Window displays all the Tests stored in the Repository and has a right-click menu option associated with it that enables you to create new Tests.
</p>
<p class="Body">
 Open the Tests folder and display your Tests by double-clicking
<strong class="Strong"> Tests.
</p>
<h6 class="Heading5">
 See also:
</h6>
<p class="Body">
 Tests Folder and Tests, Display Options and Functions
</p>
<p class="Body">
 Scripts Folder
</p>
<p class="Body">
 Collectors Folder
</p>
<h5 class="Heading4">
 Tests Folder and Tests, Display Options and Functions
</h5>

 <li class="SmartList1"> Double-click
 <strong class="Strong">Tests in the Repository Window, to expand or collapse the directory structure or click

 , or
 , alongside the folder.
 <li class="SmartList1"> Right-click
 <strong class="Strong">Tests, to display a pop-up menu from where you can create a <strong
 class="Strong">New Test > Test.
 <li class="SmartList1"> Double-click on a new Test
 , to open it in the Test Pane in the Commander Main Window, where you can add and delete Task Groups which contain
 the Scripts and Collectors you want. Apply the Task Group settings you need using the Properties Window, to control the
 load generated during a Test-run and which <span style="color: #000000; font-style: normal; font-weight: normal; text-
 decoration: none; text-transform: none; vertical-align: baseline">Hosts are
 used to run each Task Group. .
 <dl>
 <dt class="Indented2"> Note: When you first create a Test it contains no Task Groups
 containing Scripts or Collectors, so it appears with a small crossed red circle over the Test icon to indicate this
 . After you have added one or more Task Groups, the cross is removed,
 .
 </dl>
 <li class="SmartList1"> Double-click on a Test
 , to open it in the Test Pane in the Commander Main Window, where you can add, delete and reorganize Task Groups
 containing Scripts and Collectors, then edit your Task Group settings.
 <dl>
 <dt class="Indented2"> Note: When a Test is open in the Test Pane, the Test icon in the
 Repository Window appears with a small, yellow lock icon overlaid,
 . This makes it easy to spot which Test or Collector is currently open in Commander. The name of the open Test or
 Collector is displayed in the Commander Title Bar.
 </dl>
 <li class="SmartList1"> Right-click on a Test
 , to display a pop-up menu which gives you the option to <strong class="Strong">Rename or <strong
 class="Strong">Delete the Test from the Repository.
 <dl>
 <dt class="Indented2"> Note: If a Test is open
 , it cannot be renamed or deleted.

<dt class="Indented2"> Note: Scripts and Collectors are included in Tests by reference, so editing, renaming or deleting Tests does not affect the Scripts and Collectors they contain.
</dt>

<h6 class="Heading5">
 See also:
</h6>
<p class="Body">
 Tests Folder
</p>
<p class="Body">
 Creating and Editing Tests
</p>
<h3 class="Heading2">
 Repository Window Display Options
</h3>
<p class="Body">
 The Repository window is located on the left-hand side of the Main Window by default. You can hide it to increase the Main Window workspace area available, move it to a new position or float it over the Main Window.
</p>
<h6 class="Heading5">
 See also:
</h6>
<p class="Body">
 Hide/Display The Repository Window
</p>
<p class="Body">
 Move The Repository Window
</p>
<p class="Body">
 Resize The Repository Window
</p>
<p class="Body">
 Select a New Repository Path
</p>
<h5 class="Heading4">
 Hide/Display The Repository Window
</h5>

<li class="SmartList1"> Click
, in the double bar at the top of the Repository Window to close it.
<li class="SmartList1"> Select <strong class="Strong">Tools > Show Repository to hide and display the Repository Window.

<h5 class="Heading4">
 Move The Repository Window
</h5>
<ol type="1">
<li class="SmartList2" value="1"> Click on the double bar at the top of the Repository Window.
<li class="SmartList2" value="2"> Drag, then drop it in the new position within the Main Window.
<dl>
<dt class="Indented2"> Note: The Repository Window docks with the Main Window's borders if it contacts them.
</dt>
</dl>

<h5 class="Heading4">
 Resize The Repository Window
</h5>
<ol type="1">
<li class="SmartList2" value="1"> Move your cursor over part of the window edge.
<li class="SmartList2" value="2"> Click and drag, then drop the border in the required position.

<h5 class="Heading4">
 Select a New Repository Path
</h5>
<ol type="1">
<li class="SmartList2" value="1"> In Commander select <strong class="Strong">Tools > Repository Path.
<li class="SmartList2" value="2"> In the Browse for folder dialog box, select a new Repository path by moving through the directory structure displayed and choose the location you want.
<li class="SmartList2" value="3"> Click <strong class="Strong">OK to create a new Repository.
<dl>

Μέτρηση απόδοσης Web Server

```
<dt class="Indented2"> <a name="23817"> </a>Note: You can create several Repositories and use them to store
different performance Test projects. This procedure creates a new Repository if none exists in the location you choose, or
switches the Repository Path to reference an existing Repository.
```

```
</dt>
</ol>
<p class="Body">
  <a name="7853"> </a>
</p>
</blockquote>
```

```
<hr>
```

```
<table align="right" border="0" cellspacing="0" cellpadding="0">
  <tr>
    <td align="right"><font size="1">
      <a href="http://www.opensta.org">OpenSTA.org</a><br>
      <a href="http://opensta.org/lists.html">Mailing Lists</a><br>
      <a href="mailto:info@opensta.com">Further enquiries</a><br>
      <a href="mailto:docs@opensta.org">Documentation feedback</a><br>
      <a href="http://www.cyrano.com">CYRANO.com</a><br>
    </font></td>
  </tr>
</table>
```

```
<table width="331" border="0" align="left" cellpadding="0" cellspacing="0">
  <tr>
    <td><a href="index.htm"> </a></td>
    <td><a href="os-getti.htm"> </a></td>
    <td><a href="os-worki.htm"> </a></td>
    <td><a href="openstai.htm"> </a></td>
  </tr>
</table>
```

```
</body>
</html>
```

- Simpletest.php

```
<html>
<head><title>Php+MySQL</title></head>
<body>
<?php
    $link = mysql_connect("localhost", "root", "ubuntu");
    mysql_select_db("db1");

    $query = "SELECT * FROM test";
    $result = mysql_query($query);

    while ($line = mysql_fetch_array($result))
    {
        foreach ($line as $value)
        {
            print "$value\n";
        }
    }

    mysql_close($link);
?>
</body>
</html>
```

- Mediumtest.php

```
<?php Header ('Content-Type: text/html; charset=ANSI');
Header ('Content-type: image/png');

$height = 200;
$width=200;
```

```
$im=imagecreatetruecolor ($width, $height);
$white=imagecolorallocate ($im, 255, 255, 255);
$blue=imagecolorallocate ($im, 0, 0, 64);
```

```
imagefill($im, 0, 0, $blue);
imageline($im, 0, 0, $width, $height, $white);
imagestring($im, 4, 50, 150, 'Test', $white);
```

```
imagepng($im);
```

```
imagedestroy($im);?>
```

- **Hardtest.php**

```
<table border="1">
<?php
function compare_array($a, $b)
{ return strcmp($a["name"], $b["name"]); }
```

```
$i=0;
$myfile = fopen("mylist.txt", 'r');
while(!feof($myfile))
{
    $line=fgets($myfile);
    $parts=explode(" ", $line);
    $pin[$i]["name"]=$parts[0];
    $pin[$i]["surname"]=$parts[1];
    $pin[$i]["id"]=$parts[2];
    $i++;
}
```

```
fclose($myfile);
usort($pin, 'compare_array');
```

```
for($i=0;$i<1000;$i++)
{
    echo "<tr>";
    echo "<td>". $pin[$i]["name"]. "</td>";
    echo "<td>". $pin[$i]["surname"]. "</td>";
    echo "<td>". $pin[$i]["id"]. "</td>";
    echo "</tr>";
}
```

```
?>
</table>
```

- **Simpletest.aspx**

```
<%@ Page Language="C#" %>
```

```
<html>
<head>
```

```
<title> Simple test for ASP.net </title>
```

```
</head>
```

```
<asp:calendar showtitle="true" runat="server">
</asp:calendar>
```

- **Hardtest.aspx**

```
<%@ Import Namespace="System.IO" %>
<script language="vb" runat="server">
sub Page_Load(sender as Object, e as EventArgs)
'Open a file for reading
Dim FILENAME as String = Server.MapPath("mylist.txt")
```

```
'Get a StreamReader class that can be used to read the file
Dim objStreamReader as StreamReader
objStreamReader = File.OpenText(FILENAME)
```

```
'Now, read the entire file into a string
Dim contents as String = objStreamReader.ReadToEnd()
```

```
'Set the text of the file to a Web control  
lblRawOutput.Text = contents
```

```
"We may wish to replace carriage returns with <br>s  
lblNicerOutput.Text = contents.Replace(vbCrLf, "<br>")
```

```
objStreamReader.Close()  
end sub  
</script>
```

```
<b>Raw File Output</b><br />  
<asp:label runat="server" id="lblRawOutput" />  
<p>  
<b>Nicer Output</b><br />  
<asp:label runat="server" id="lblNicerOutput" Font-Name="Verdana" />
```

- Simplestest.pl

```
#!/usr/bin/perl -w
```

```
print "Content-type: text/html\n\n";  
print "Simple test for cgi server script.<br /> \n";
```

```
for ($i=0; $i<10; $i++)  
{  
print $i."<br />";  
}
```

- Hardtest.pl

```
#!/usr/bin/perl
```

```
print "Content-type: text/html\n\n";
```

```
open (MYLIST, "mylist.txt");
```

```
print "<table border=1>";
```

```
while ($record = <MYLIST>) {  
my @parts=split(' ', $record);  
print "<tr>";  
foreach my $val (sort(@parts)){  
print "<td>";  
print "$val";  
print "</td>";  
}  
print "</tr>";  
print "<tr>";  
foreach my $val (reverse(sort(@parts))){  
print "<td>";  
print "$val";  
print "</td>";  
}  
print "</tr>";  
foreach my $val (@parts[0]){  
push(@arrays1, $val);  
}  
foreach my $val (@parts[1]){  
push(@arrays2, $val);  
}  
foreach my $val (@parts[2]){  
push(@arrays3, $val);  
}  
}  
for($i=0; $i<1000; $i++){  
  
push(@final, join(" ", @arrays1[$i], @arrays2[$i], @arrays3[$i]));  
@final=sort(@final);  
}  
print "</table>";
```

```
foreach $val (@final){
    print "<br>";
    print "$val";
}

@final=reverse(@final);

foreach $val (@final){
    print "<br>";
    print "$val";
}
@final=reverse(@final);
print "<table border=1>";
foreach $val (@final){
    print "<tr><td>";
    print "$val";
    print "</td></tr>";
}
print "</table>";
close(MYLIST);
```

- Simplestest.jsp

```
<HTML>
<BODY>
<%
// This scriptlet declares and initializes "date"
System.out.println( "Evaluating date now" );
java.util.Date date = new java.util.Date();
%>
Hello! The time is now
<%
// This scriptlet generates HTML output
out.println( String.valueOf( date ));
%>
</BODY>
</HTML>
```

- Hardtest.jsp

```
<%@page import="java.util.Arrays"%>
<%@page import="java.util.Comparator"%>

<%@ page import="java.io.*" %>
<html>
<head>
<title>hard test JSP</title>
</head>

<body>
<%
String fileName=getServletContext().getRealPath("mylist.txt");
FileInputStream fis = new FileInputStream(fileName);
DataInputStream dis = new DataInputStream(fis);
String line = "";
String [][] array=new String[1000][3];
int i=0;
while ((line = dis.readLine()) != null){
    String[] tokens=line.split(" ");
    for(int j=0;j<3;j++)
        array[i][j]=tokens[j];
    i++;
}
dis.close();

Arrays.sort(array, new Comparator<String[]>() {
    @Override
    public int compare(final String[] entry1, final String[] entry2) {
        final String time1 = entry1[0];
        final String time2 = entry2[0];
        return time1.compareTo(time2);
    }
});
```

```
out.print("<table border='1'>");
for(i=0;i<1000;i++){
  out.print("<tr>");
  for(int j=0;j<3;j++){
    out.print("<td>");
    out.println(array[i][j]);
    out.print("</td>");
  }
  out.print("</tr>");}
out.print("</table>");
%>
Successfully write file
</body>
</html>
```


Βιβλιογραφία

- Σαλαμπάσης, Μ. (2008), Εισαγωγή στον Προγραμματισμό Διαδικτυακών Εφαρμογών, Αλεξάνδρειο Τεχνολογικό Ίδρυμα Θεσσαλονίκης
- Freestuff.gr. (2007), Εγκατάσταση Pho, Apache και Mysql σε Windows
- Alvin P. (2009), How to open and read files in Perl, devdaily.com
- Brown K. (2005), Perl array reverse() function – Quick Tutorial, perl.about.org
- Chinnathampi J. (2003), How to read a text file in ASP.Net, asplliance.com
- Denekamp E. (2008), Installing a PHP application on windows 2008, blog.infosupport.com
- Drupal Team (2011), Apache Bench (ab), drupal.org
- Frost S. (2011), How to – ASP.Net Running on Linux
- Gite V. (2008), Ubuntu Linux Install Sun Java Development Kit and Java Runtime Enviroment, cyberciti.biz
- Jared (2011), Installing WEMP (Windows, Nginx, MySQL,PHP), dyesi.com
- Josh D. (2006), Setting up Lighttpd with PHP, MySQL and Perl on Windows, joshdick.net
- Kerneis G. (2005), Web Server Benchmark, pps.jussieu.fr
- Klassa J. (2000), Sorting in Perl, Raleigh.pm.org
- Lung C. (2011), Installing Lighttpd, Php and Mysql on Ubuntu 10.10, giantflyingsaucer.com.
- Meloni, C. J. (2004), Μάθετε PHP, MySQL, Apache, Γκιούρδας Εκδόσεις
- Nixcraft (2006), HowTo: Performance Benchmarks a webserver, cyberciti.biz
- Parez T. (2008), Hosting ASP.Net with Ubuntu Linux and Apache2, aspspider.com
- Patterson, L. (2007), Οδηγός της HTML 4, Γκιούρδας Εκδόσεις
- Ricocheting.com (2011), How to install Perl on Windows
- Rose India (2008), How to read text files in Servlets, roseindia.net
- Thomson L., Welling L., (2009), Ανάπτυξη Web Εφαρμογών με PHP και MySQL, Γκιούρδας Εκδόσεις.
- Timme F. (2010), Installing Nginx with PHP5 and Mysql, HowtoForge.com
- Ubuntu Documentation (2009), ModMono, help.ubuntu.com
- W3Schools, ASP.Net Examples, w3schools.com
- Wikipedia (2011), Web server Benchmarking, Wikipedia.org
- Yacoub H. (2010), Install ASP.Net 3.5 Environment for Ubuntu Maverick 10.10, hdyconsultancy.com
- Yiodaditia M. (2010), How to Install Glassfish on Ubuntu 10.10, yoodey.com
- YoLinux (2009), Linux Tutorial, Web server Bencmarking, yolinux.com