ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

Υπολογιστική υψηλών επιδόσεων με τη χρήση Web Workers

Διπλωματική εργασία

Του φοιτητή

Μπιλμπίλη Βασίλειου

Α.Μ. : 07/15

Θεσσαλονίκη 11/2018

ΥΠΟΛΟΓΙΣΤΙΚΗ ΥΨΗΛΩΝ ΕΠΙΔΟΣΕΩΝ ΜΕ ΤΗΝ ΧΡΗΣΗ WEB WORKERS

Μπιλμπίλης Βασίλειος

Πτυχίο Διοίκησης Τεχνολογίας, Τμήμα Εφαρμοσμένης Πληροφορικής,

Πανεπιστήμιο Μακεδονίας, 2013

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής: Κασκάλης Θεόδωρος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 6/11/18

Κασκάλης Θεόδωρος Σακελλαρίου Ηλίας Μαργαρίτης Κωνσταντίνος

Μπιλμπίλης Βασίλειος

Περιεχόμενα

Κατάλογος Εικόνων

Περίληψη

Σκοπός αυτής της διπλωματικής είναι η μελέτη της τεχνολογίας των HTML5 Web Workers και της σχετικής βιβλιογραφίας, ώστε να παρουσιαστούν οι νέες δυνατότητες που μπορεί να προσδώσει στις web εφαρμογές, αξιοποιώντας την δυνατότητα για multithreading που υποστηρίζουν οι σύγχρονοι επεξεργαστές. Προκειμένου να γίνουν πιο κατανοητά τα οφέλη από την χρήση της τεχνολογίας αυτής, εκτελέστηκαν κάποια συγκριτικά τεστ (benchmarks) σε τρία συστήματα με διαφορετικό hardware, χρησιμοποιώντας τρεις διαφορετικές web εφαρμογές, οι οποίες επιλέχθηκαν για δείξουμε κατά πόσο μπορούν να βελτιωθούν οι επιδόσεις μια εφαρμογής από την χρήση multithreading, αξιοποιώντας το Web Workers API. Οι εφαρμογές αυτές εκτελέστηκαν σε μερικούς από τους πιο δημοφιλείς φυλλομετρητές (Chrome, Firefox, Edge, Opera) προκειμένου να δούμε κατά πόσο θα μπορούσε να επηρεαστεί η απόδοση ανάλογα με το ποιον browser χρησιμοποιούμε. Ολοκληρώνοντας, έχοντας λάβει υπόψιν την σχετική βιβλιογραφία αλλά ταυτόχρονα έχοντας εξάγει επιπλέον συμπεράσματα από τα συγκριτικά test που πραγματοποιήθηκαν, αποτυπώνονται τα οφέλη και τα πλεονεκτήματα που προκύπτουν την χρήση μιας τεχνολογίας όπως οι Web Workers, στις σύγχρονες web εφαρμογές.

Λέξεις Κλειδιά: HTML5 Web Workers, Πολυνηματικός προγραμματισμός, Σύγκριση επιδόσεων φυλλομετρητών, Υπολογιστική υψηλών επιδόσεων με Javascript, Παράλληλη επεξεργασία στον browser

Abstract

The subject of this dissertation is to study the technology of HTML5 Web Workers and the associated bibliography, in order to present all the features that this API can contribute to emerging web applications, by making use of multithreading in modern devices CPUs. In order to express the benefits of using this technology with more clarity, we ran some benchmarks on three different systems by using three web applications, which have been chosen as best candidates for this task. These web applications were executed on most popular web browsers (Chrome, Firefox, Edge, Opera), so that we can present the differences in performance that depend on the different browser's engine factor. Finally, we write down the conclusions that have emerged by studying the bibliography and by running the benchmarks so that we can present the importance and impact in performance of using Web Workers in modern web applications.

Keywords: HTML5 Web Workers, Client-Side Computing, Browser Benchmarking, Pervasive Computing with Javascript, Multithreading on Browser

1. Εισαγωγή

1.1 Πρόβλημα – Σημαντικότητα Θέματος

Τα τελευταία χρόνια παρατηρείται ότι όλο και περισσότερες εφαρμογές τείνουν να εκτελούνται μέσω ενός browser με τις εφαρμογές αυτές να εξελίσσονται συνεχώς και να γίνονται ολοένα και πιο απαιτητικές σε επεξεργαστική ισχύ. Παράλληλα, οι περισσότερες συσκευές που χρησιμοποιούμε καθημερινά για την πρόσβαση μας σε ιστοσελίδες και web εφαρμογές διαθέτουν πλέον επεξεργαστές με πολλούς πυρήνες. Καλείστε λοιπόν να απαντήσετε το παρακάτω ερώτημα. Αξιοποιείται πραγματικά όλη η ισχύς που μπορούν να παρέχουν οι σύγχρονες αυτές συσκευές; Έχοντας ως δεδομένο ότι η Javascript χρησιμοποιεί ένα single-thread μοντέλο για την εκτέλεση της, γίνεται εύκολα αντιληπτό ότι αυτή η ιδιαιτερότητα της περιορίζει αρκετά της επιδόσεις των σύγχρονων web εφαρμογών καθώς δεν μπορούν να αξιοποιηθούν οι επιπλέον πυρήνες που διαθέτουν οι νέες συσκευές, όπως γίνεται σε άλλες γλώσσες προγραμματισμού π.χ. C++ και Java. Λαμβάνοντας υπόψη το παραπάνω πρόβλημα, θεωρήθηκε σημαντικό να γίνει μελέτη του HTML5 Web Workers API, το οποίο έρχεται να δώσει λύση στο πρόβλημα αυτό προσθέτοντας την δυνατότητα για πολυνηματική εκτέλεση σε οποιαδήποτε web εφαρμογή.

1.2 Σκοπός – Στόχοι

Η παρούσα μελέτη έχεις ως στόχο να παρουσιαστούν αλλά και να αναδειχθούν οι δυνατότητες των Web Workers, καθώς και να διερευνηθεί σε ποιες περιπτώσεις θα μπορούσαν να επωφεληθούν οι σύγχρονες συσκευές όπως tablets, smartphones και υπολογιστές, από την δυνατότητα για multithreading εκτέλεση, των σύγχρονων web εφαρμογών στους browsers. Πιο συγκεκριμένα, οι τομείς οι οποίοι θεωρήθηκε σημαντικό να διερευνηθούν είναι οι εξής:

- Ποιος τύπος εφαρμογών θα μπορούσε να επωφεληθεί περισσότερο (π.χ. εφαρμογή επεξεργασία εικόνας)
- Ποιες συσκευές δείχνουν να επωφελούνται περισσότερο από την εκτέλεση των web εφαρμογών σε πολλαπλούς πυρήνες ή threads
- Ποιος είναι ο βέλτιστος αριθμός threads που θα πρέπει να εκτελούνται παράλληλα ώστε η απόδοση να είναι η καλύτερη δυνατή

- Διαφορά στις επιδόσεις των web εφαρμογών με την χρήση των Web Workers
- Σύγκριση επιδόσεων μεταξύ των δημοφιλών φυλλομετρητών για να αποτυπωθούν οι διαφορές στις επιδόσεις που οφείλονται στην χρήση διαφορετικών Javascript engines από του browsers
- Σύγκριση επιδόσεων μεταξύ διαφορετικών εκδόσεων του ίδιου φυλλομετρητή για να παρουσιαστεί η εξέλιξη στις επιδόσεις σε συνδυασμό με την εξέλιξη των Javascript Engines των browsers

1.3 Συνεισφορά

Η συνεισφορά της συγκεκριμένης έρευνας θα λέγαμε ότι είναι η παρουσίαση του αντίκτυπου που έχει η χρήση του Web Workers API στις σύγχρονες συσκευές όπως τα smartphones και τους προσωπικούς υπολογιστές, καθώς όλες αυτές οι συσκευές πλέον χρησιμοποιούν επεξεργαστές με πολλούς πυρήνες και επομένως οι επιδόσεις τους μπορούν να επωφεληθούν σημαντικά από την παραπάνω τεχνολογία. Αυτό αποτυπώνεται με μετρήσιμα μεγέθη, μέσα από τα συγκριτικά τεστ (benchmarks) που έγιναν αλλά και σε συνδυασμό με τα συμπεράσματα που προέκυψαν μέσα από με την μελέτη της υπάρχουσας βιβλιογραφίας.

1.4 Διάρθρωση της μελέτης

Στο δεύτερο κεφάλαιο πραγματοποιείται βιβλιογραφική επισκόπηση των HMTL5 Web Workers και των δυνατοτήτων τους, καθώς και μια σύντομη παρουσίαση του τρόπου με τον οποίο μπορεί να ενσωματωθεί το API σε μια ήδη υπάρχουσα εφαρμογή. Στην συνέχεια γίνεται μια επισκόπηση της βιβλιογραφίας που αφορά την τεχνολογία των Web Workers αλλά και των συμπερασμάτων που προκύπτουν από την μελέτη τους.

Στο τρίτο κεφάλαιο γίνεται μια επισκόπηση των συγκριτικών τεστ (benchmarks) που έγιναν στο πλαίσιο της διπλωματικής αυτής, ανάλυση των σχετικών αποτελεσμάτων αλλά και καταγραφή των συμπερασμάτων που προκύψανε από αυτά. Στόχος αυτού του κεφαλαίου είναι να μας δείξει με μετρήσιμο τρόπο το πόσο σημαντικό όφελος μπορεί να αποδώσει η τεχνολογία αυτή σε συγκεκριμένους τύπους εφαρμογών, όπως επεξεργασία εικόνας, image rendering, πολύπλοκοι υπολογισμοί.

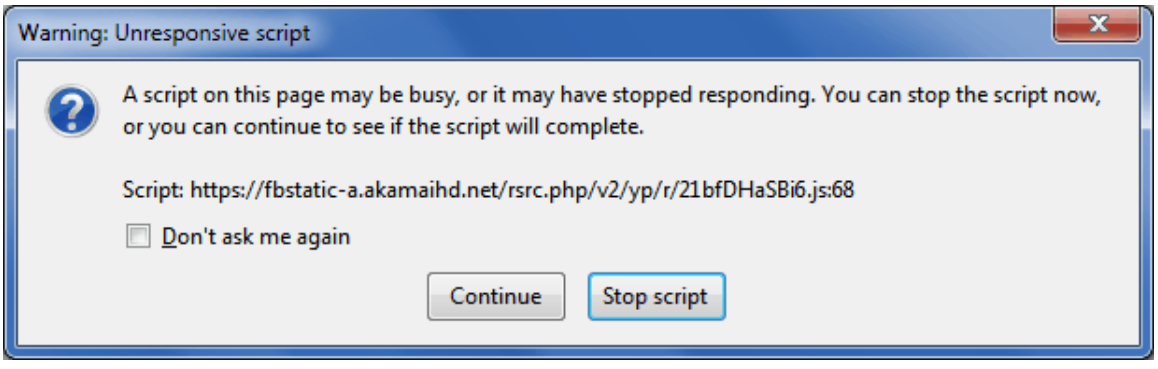
Κλείνοντας, στο τέταρτο και τελευταίο κεφάλαιο παρουσιάζονται τα συμπεράσματα που προέκυψαν μετά από την μελέτη της σχετικής βιβλιογραφίας αλλά και από τα συγκριτικά τεστ που εκτελέστηκαν προκειμένου να γίνει ένας συνολικός απολογισμός των πλεονεκτημάτων αλλά και των περιορισμών που προκύπτουν από την χρήση του Web Workers API.

2. Βιβλιογραφική Επισκόπηση – Θεωρητικό Υπόβαθρο

2.1 HTML5 Web Workers

Με τον όρο web worker αναφερόμαστε σε ένα script της Javascript το οποίο εκτελείται στο παρασκήνιο, ανεξάρτητα από το κεντρικό thread της σελίδας. Κάθε φορά που δημιουργείται ένας web worker σε μια εφαρμογή, δημιουργείται ένα νέο thread το οποίο μπορεί να εκτελέσει ένα «βαρύ» script διατηρώντας την αποκρισιμότητα της διεπαφής με τον χρήστη (UI), χωρίς έτσι να επηρεάζεται αρνητικά η εμπειρία του χρήστη.

Το πρόβλημα που υπάρχει μέχρι και σήμερα στις web εφαρμογές είναι ότι αν προσπαθήσουμε να εκτελέσουμε ένα script το οποίο χρειάζεται κάποιο σημαντικό χρονικό διάστημα για την ολοκλήρωση της εκτέλεσης του, θα παρατηρήσουμε ότι ο browser θα «παγώσει» και θα μας εμφανίσει ένα μήνυμα παρόμοιο με αυτό που δείχνει η εικόνα 1.



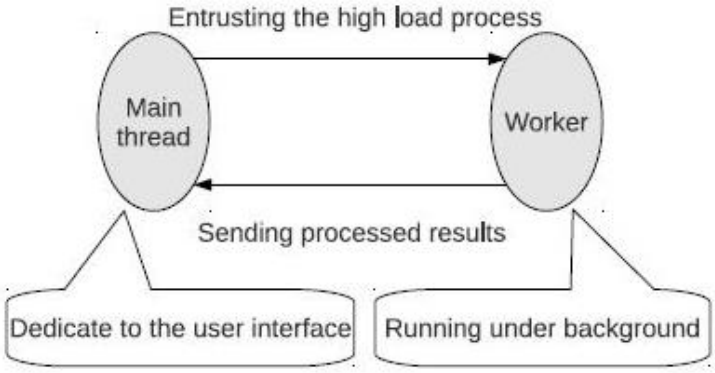
Εικόνα 1: Unresponsive script message

Το πρόβλημα αυτό δημιουργείται λόγω της φύσης της Javascript, η οποία όπως ήδη θα γνωρίζετε χρησιμοποιεί ένα single-threaded μοντέλο για την εκτέλεση της. Γεγονός το οποίο ουσιαστικά σημαίνει ότι όλες οι εφαρμογές ή ιστοσελίδες που «τρέχουν» σε έναν browser χρησιμοποιούν ένα μόνο thread για την εκτέλεση υπολογισμών, φόρτωση

περιεχομένου (όπως εικόνες, βίντεο κλπ.) αλλά και για την ανταποκρισιμότητα του user interface σε ενέργειες του χρήστη όπως π.χ. κλικ σε κάποιο κουμπί.

Μια εναλλακτική λύση που χρησιμοποιείται μέχρι και σήμερα αλλά δεν αποτελεί ουσιαστική λύση του προβλήματος, είναι η χρήση της συνάρτησης setTimeout(). Με αυτή, μπορούμε να τρέξουμε σχεδόν παράλληλα υπολογισμούς ή μεθόδους που χρειάζονται πολύ χρόνο για να ολοκληρωθούν. Ο λόγος που δεν αποτελεί λύση είναι ότι το μόνο που κερδίζουμε στην περίπτωση αυτή είναι η διατήρηση της αποκρισιμότητας του UI, ώστε να μην παγώνει ο browser του χρήστη, καθώς αυτό που συμβαίνει είναι η προσωρινή παύση (ανά τακτά χρονικά διαστήματα) του κεντρικού thread για ένα μικρό διάστημα προκείμενου να γίνει εκτέλεση του τμήματος κώδικα που θέλουμε να εκτελεστεί και έπειτα συνεχίζεται και πάλι η εκτέλεση του κεντρικού script.

Σε αντίθεση με την παραπάνω περίπτωση όταν δημιουργείται ένας Web Worker, εκτελείται παράλληλα με το κεντρικό thread. Συνεπώς η λύση των Web Workers είναι αυτή που δίνει πραγματικά την δυνατότητα για multithreading στην Javascript (παράλληλη εκτέλεση script), όπως ακριβώς συμβαίνει στις υπόλοιπες γλώσσες προγραμματισμού αλλά βέβαια αυτή η δυνατότητα έρχεται με κάποιους περιορισμούς, οι οποίοι θα αναφερθούν και εκτενέστερα παρακάτω. Στην παρακάτω εικόνα μπορείτε να δείτε ένα σχήμα το οποίο παρουσιάζει την σχέση μεταξύ κεντρικού thread και web worker.



Εικόνα 2: Σχέση Main thread – Worker thread

Πριν προχωρήσουμε όμως στις δυνατότητες και του περιορισμούς που υπόκεινται οι Web Workers είναι σημαντικό να αναφερθεί ο τρόπος με τον οποίο μπορείτε να χρησιμοποιήσετε την τεχνολογία αυτή. Αρχικά, θα πρέπει πρώτα να δημιουργήσετε μέσα από το κεντρικό script της ιστοσελίδας σας έναν νέο web worker. Ο νέος αυτός web

worker, θα είναι ουσιαστικά ένα ξεχωριστό αρχείο javascript το οποίο θα εκτελείται παράλληλα και ανεξάρτητα από τα υπόλοιπα αρχεία javascript, τα οποία ενδεχομένως να χρησιμοποιεί η ιστοσελίδα σας. Για να δημιουργήσετε έναν web worker το μόνο που χρειάζεται είναι να καλέσετε των κατασκευαστή worker(), προσδιορίζοντας την διαδρομή (path) του script το οποίο θέλετε να εκτελέσει ο νέος web worker (π.χ. worker.js). Όπως φαίνεται στο τμήμα κώδικα παρακάτω, αρχικά με την εντολή if (window.worker) που βρίσκεται στην γραμμή 1, ο φυλλομετρητής (browser) ελέγχει αν υποστηρίζει το Web Workers API και εφόσον η απάντηση είναι θετική επιστρέφει την τιμή true. Στη συνέχεια με την εντολή της γραμμής 2, δημιουργείτε ένας νέος web worker ο οποίο εκτελεί τον κώδικα που περιλαμβάνει το αρχείο «worker.js».

Τμήμα κώδικα στο script του main thread:

```
1   if (window.Worker) {
2   var myWorker = new Worker("worker.js");
3   }
```

Πίνακας 1: Script Δημιουργίας Web Worker

Πριν συνεχίσουμε είναι σημαντικό να αναφερθεί ότι υπάρχουν δύο τύποι web worker, dedicated και shared worker. Ο τύπος web worker με τον οποίο θα ασχοληθούμε στο πλαίσιο της μελέτης αυτής ονομάζεται Dedicated Worker και είναι ο πιο συνηθισμένος τύπος Web Worker. Σε έναν dedicated worker έχει πρόσβαση μόνο το κεντρικό script της εφαρμογής (main thread), μέσα από το οποίο και δημιουργήθηκε, και μόνο με αυτό μπορεί έχει επικοινωνία μέσω της ανταλλαγής μηνυμάτων.

Για την επικοινωνία μεταξύ του κεντρικού script και του script που εκτελεί ο Web Worker, χρησιμοποιείται το Messaging API της HTML5, μέσω του οποίου μπορεί να γίνει ανταλλαγή μηνυμάτων (strings, arrays, JSON objects). Για την αποστολή μηνύματος από την μια πλευρά προς την άλλη γίνεται χρήση της μεθόδου postMessage() και για την παραλαβή του μηνύματος στην άλλη πλευρά χρησιμοποιείται ο χειριστής συμβάντων onmessage ή ένας event listener. Με το παρακάτω σύντομο παράδειγμα (Πίνακες 2 και 3) μπορείτε να δείτε πως υλοποιείται η επικοινωνία μεταξύ του κεντρικού thread και του Web Worker thread. Όπως φαίνεται στην γραμμή 2 του παρακάτω κώδικα (Πίνακα 2), προκειμένου να στείλετε ένα μήνυμα «Hello World» στον web worker γίνεται κλήση της

μεθόδου postMessage() για το αντικείμενο myWorker που δημιουργήθηκε νωρίτερα και δίνεται στην μέθοδο αυτή ως όρισμα το επιθυμητό αλφαριθμητικό.

Τμήμα κώδικα στο script του main thread:

```
1   button.onclick = function() {
2     myWorker.postMessage("Hello World");
3   }
```

Πίνακας 2: Αποστολή μηνύματος από το main thread

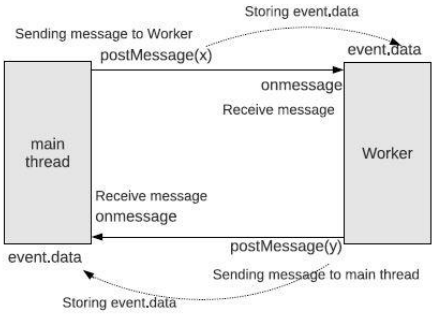
Από την πλευρά του web worker, χρησιμοποιείται ο χειριστής συμβάντων onmessage (Γραμμή κώδικα 1, Πίνακας 3), ο οποίος ελέγχει αν έχει έρθει κάποιο μήνυμα από το κεντρικό script (UI thread) και αν ναι, εκτελεί τις ενέργειες που του έχουν ανατεθεί και αποστέλλει πίσω στο κεντρικό thread κάποιο μήνυμα μέσω της postMessage() όπως φαίνεται και στην γραμμή 2 του παρακάτω κώδικα (Πίνακα 3).

Τμήμα κώδικα στο script του Web Worker:

```
1   onmessage = function() {
2     myWorker.postMessage("Hello World from Web Worker");
3   }
```

Πίνακας 3: Παραλαβή μηνύματος από τον Web Worker

Στην παρακάτω εικόνα μπορείτε να δείτε το διάγραμμα που απεικονίζει συνολικά τον τρόπο επικοινωνίας μεταξύ κεντρικού thread και web worker.



Εικόνα 3: Διάγραμμα επικοινωνίας μεταξύ κεντρικού thread και web worker

Κλείνοντας, είναι σημαντικό να αναφερθεί ότι κάθε web worker ξεκινάει να εκτελείται από την στιγμή που αρχικοποιείται και συνεχίζει να εκτελείται στο παρασκήνιο μέχρι να καλέσετε μία από τις μεθόδους close() ή terminate(), ώστε να τερματιστεί το thread του

web worker. Η διαφορά μεταξύ των δύο μεθόδων είναι ότι η μέθοδος close() καλείται μέσα από το script του web worker, ενώ η μέθοδος terminate() καλείται από το κεντρικό script. Είναι σημαντικό όταν ένας web worker έχει ολοκληρώσει την εργασία που του ανατέθηκε να τερματίσει την λειτουργία του προκειμένου να μην δεσμεύει πόρους από το σύστημα (ακόμη και αν είναι αδρανής).

2.2 Περιπτώσεις εφαρμογών όπου προτείνεται η χρήση Web Workers

Ο γενικός κανόνας είναι, ότι αν μια εφαρμογή χρησιμοποιεί μια διεργασία η οποία απαιτεί αρκετό χρόνο για ολοκληρώσει την εκτέλεσης της και επομένως επηρεάζει αρνητικά την εμπειρία του χρήστη τότε θα πρέπει να διερευνηθεί η δυνατότητα μεταφοράς του κρίσιμου κώδικα (που δημιουργεί την καθυστέρηση) σε ένα νέο thread με την χρήση ενός Web Worker. Οι πιο κλασσικές περιπτώσεις τέτοιων εφαρμογών είναι οι παρακάτω:

- Εφαρμογές που κάνουν πολύπλοκους μαθηματικούς υπολογισμούς
- Που ταξινομούν ή επεξεργάζονται μεγάλου μεγέθους πίνακες
- Που επεξεργάζονται μεγάλα αντικείμενα JSON (parsing)
- Που κάνουν ανάλυση ή επεξεργασία δεδομένων βίντεο, εικόνας και ήχου
- Που χειρίζονται αιτήματα μεταξύ client και server (e.g. Background I/O)
- Εφαρμογές που εκτελούν συντρέχοντα αιτήματα προς μια βάση δεδομένων
- Εφαρμογές που χρησιμοποιούνται για την επισήμανση του συντακτικού ενός κώδικα ή κάποιου άλλου τύπου ανάλυση κειμένου σε πραγματικό χρόνο

Αξίζει να σημειωθεί ότι σε κάποιες περιπτώσεις δεν είναι εφικτό να απομονωθεί το τμήμα που απαιτεί περισσότερο χρόνο και να μεταφερθεί σε έναν web worker, καθώς κάποιες εφαρμογές μπορεί να κάνουν χρήση κάποιας βιβλιοθήκης στην οποία δεν έχουν πρόσβαση οι web workers (όπως π.χ. η JQuery) ή γενικά να χρειάζονται άμεση πρόσβαση στο DOM που είναι ένας επίσης περιορισμός που έρχεται με τους web workers. Περισσότερα όμως για τους περιορισμούς αυτούς θα αναφερθούν στην συνέχεια.

2.3 Περιορισμοί και Δυνατότητες

Λόγω της multi-threaded φύσης τους, οι web workers υπόκειται σε κάποιους περιορισμούς, προκειμένου να αποφευχθούν τυχόν προβλήματα από συντρέχουσες διεργασίες. Παρακάτω περιγράφονται οι περιορισμοί αυτοί:

- Δεν έχουν πρόσβαση Document Object Model (DOM) και επομένως δεν έχουν πρόσβαση στα window, document και parent objects
- Δεν έχουν πρόσβαση σε βιβλιοθήκες Javascript οι οποίες χρησιμοποιούν τα παραπάνω objects, όπως π.χ. η jQuery
- Δεν υπάρχει κοινόχρηστη μνήμη μεταξύ main thread και web worker προκειμένου να μοιράζονται δεδομένα

Παρ' όλο που οι Web Workers έχουν πρόσβαση σε ένα περιορισμένο κομμάτι των δυνατοτήτων της Javascript, υπάρχουν ακόμη αρκετές χρήσιμες δυνατότητες τις οποίες μπορούν να αξιοποιήσουν, όπως είναι οι παρακάτω:

- Πρόσβαση στο navigator object που μας δίνει πληροφορίες σχετικά με την έκδοση και το όνομα του φυλλομετρητή αλλά και την έκδοση του λειτουργικού στο οποίο αυτός εκτελείται (appName, appVersion, platform, and userAgent)
- Πρόσβαση στο location object (read-only) που μας παρέχει πληροφορίες σχετικά με την τοποθεσία από την οποία έχει αποκτήσει πρόσβαση στο internet ο χρήστης
- Δυνατότητα χρήσης του XMLHttpRequest για την αποστολή και λήψη αντικειμένων (objects) από ένα url κάνοντας χρήση της AJAX
- Δυνατότητα χρήσης των συναρτήσεων setTimeout() / clearTimeout() and setInterval() / clearInterval()
- Δυνατότητα χρήσης της application cache (Web Sockets, Web Data Storage)
- Δυνατότητα εισαγωγής άλλων script για χρήση μέσα σε ένα web worker με την μέθοδο importScripts()
- Δυνατότητα δημιουργίας Web Workers (Subworkers) μέσα από ένα υπάρχον Web Worker (υποστηρίζεται μόνο από τον Firefox προς το παρόν)

2.4 Υποστήριξη Web Workers από τους φυλλομετρητές

Η υποστήριξη του Web Workers API σήμερα θα λέγαμε ότι είναι άριστη, καθώς η τεχνολογία υποστηρίζεται από όλους τους σύγχρονους φυλλομετρητές, σχεδόν χωρίς εξαίρεση όπως φαίνεται και στον παρακάτω πίνακα. Αυτό βέβαια δεν ήταν πάντα ο κανόνας, καθώς όταν πρωτοεμφανίστηκαν οι Web Workers το 2009, δεν υπήρχε το ίδιο θερμή υποδοχή από όλους τους web browsers.

Οι πρώτοι web browsers που έσπευσαν να ενσωματώσουν την τεχνολογία αυτή ήταν ο Firefox Version 3.5 (2009), ο Safari Version 4 (2009) και ο Chrome Version 4 (2010). Μια περίπτωση που ξεχώρισε είναι ο native browser της έκδοσης Android 2.1 (2009), ο οποίος υποστήριζε πλήρως τους web workers αλλά στην συνέχεια για σταμάτησε να τους υποστηρίζει μέχρι και την έκδοση Android 4.4 το 2013, όπου εδραιώθηκε η υποστήριξη τους στον mobile browser του Android και διατηρείται μέχρι και σήμερα. Αυτά βέβαια όσον αφορά την πιο διαδεδομένη μορφή των web workers, τους dedicated workers.

Όσο για τους shared workers, τα πράγματα είναι κάπως δύσκολα μέχρι και σήμερα καθώς η υποστήριξη τους δεν είναι επαρκής καθώς λείπουνε από την λίστα browser όπως ο Safari, Microsoft Edge, Chrome for Android και ο Safari for IOS, όπως φαίνεται και από τον σχετικό πίνακα παρακάτω. Οι shared workers υποστηρίζονται κυρίως από τους Chrome, Firefox και Opera, με αξιοσημείωτη την παρουσία του Firefox for Android στον πίνακα αυτόν. Επομένως όπως γίνεται αντιληπτό, αυτός είναι και ένας κύριος λόγος για τον οποίο δίνεται έμφαση στην μελέτη των dedicated web workers και όχι σε αυτή των shared workers.

Τέλος, παρακάτω παρατίθενται οι σχετικοί πίνακες στους οποίους περιλαμβάνονται όλοι οι γνωστοί φυλλομετρητές και επομένως μπορείτε να δείτε με μεγαλύτερη ακρίβεια ποιο είναι το επίπεδο υποστήριξης για κάθε τύπο web worker.

Πίνακας υποστήριξης Web Workers

Εικόνα 4: Πίνακας υποστήριξης Web Workers (Πηγή: https://caniuse.com/)

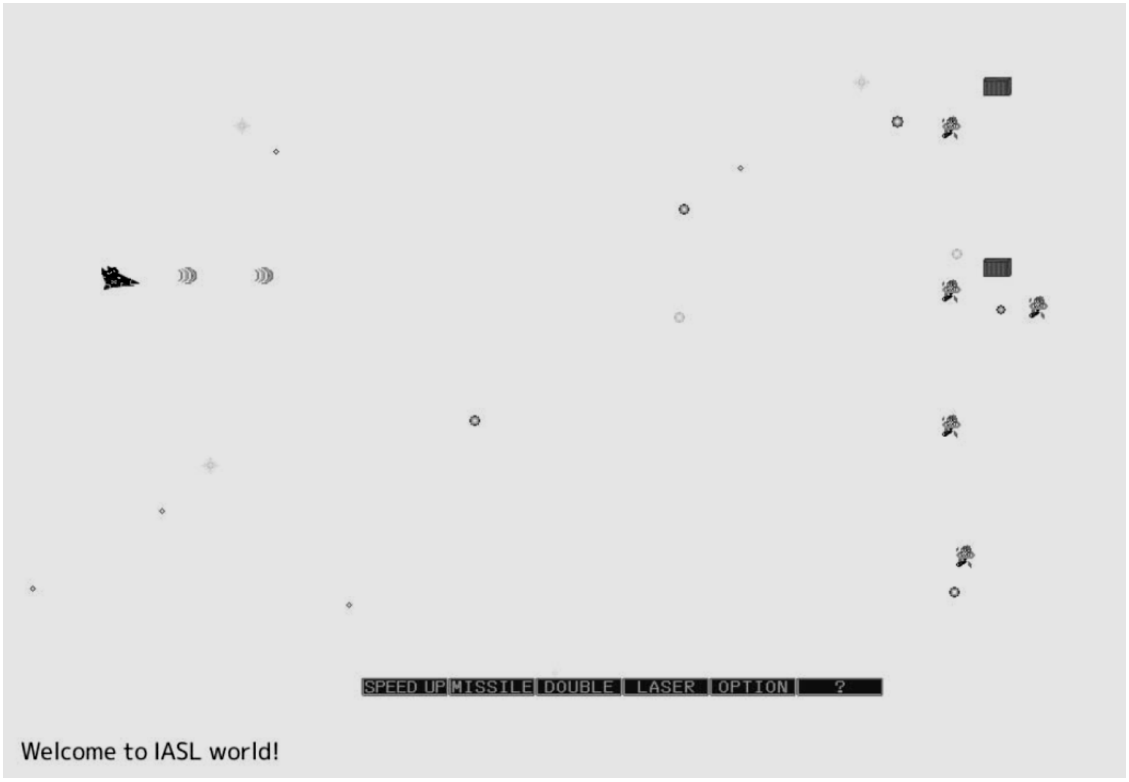
Πίνακας υποστήριξης Shared Workers

Εικόνα 5: Πίνακας υποστήριξης Shared Workers (Πηγή: https://caniuse.com/)

2.5 Επισκόπηση Άρθρων – Βιβλιογραφίας

2.5.1 **Παραλληλοποίηση Διαδραστικού Παιχνιδιού με χρήση Web Workers**

Στο άρθρο αυτό οι Yuta Watanabe, Shusuke Okamoto, Masaki Kohana, Masaru Kamada και Tatsuhiro Yonekura παρουσιάζουν ένα κλασσικό arcade game των 80s, στο οποίο ο χρήστης έχει ένα διαστημόπλοιο το οποίο ταξιδεύει στο διάστημα και έρχεται αντιμέτωπο με διάφορους αντιπάλους οι οποίοι προσπαθούν να το καταστρέψουν εκτοξεύοντας προς αυτό πυραύλους. Στην παρακάτω εικόνα βλέπετε μια εικόνα κατά την διάρκεια εκτέλεσης του παιχνιδιού.



Εικόνα 6: Στιγμιότυπο κατά την διάρκεια εκτέλεσης του Arcade Game

Το παιχνίδι αυτό εξελίσσεται σε περιβάλλον web browser και είναι γραμμένο σε Javascript. Ακριβώς για αυτό τον λόγο, όλοι οi υπολογισμοί του παιχνιδιού αλλά και η σχεδίαση του κάθε frame, γίνονται από ένα μόνο thread. Αυτός ο περιορισμός όμως δημιουργεί ένα πολύ σημαντικό πρόβλημα. Καθώς σχεδιάζονται όλο και περισσότερα αντικείμενα στον καμβά του παιχνιδιού, η εμπειρία του χρήστη χειροτερεύει σημαντικά, καθώς η σχεδίαση του κάθε frame απαιτεί όλο και περισσότερο χρόνο για να ολοκληρωθεί.

Προκειμένου να λυθεί το πρόβλημα αυτό γίνεται χρήση των Web Workers, ώστε να μπορούν να εκτελεστούν παράλληλα αρκετές από τις διαδικασίες που απαιτούνται για την σχεδίαση του κάθε frame.

Στην αρχική υλοποίηση του παιχνιδιού η εκτέλεση γινόταν με σειριακό τρόπο και περιλάμβανε τα εξής βήματα:

1) Δημιουργία ενός canvas element για την σχεδίαση της επιφάνειας στην οποία θα εμφανίζεται το παιχνίδι
2) Έλεγχος εάν κάποιο αντικείμενο έχει συγκρουστεί με κάποιο άλλο
3) Υπολογισμός της επόμενης θέσης για το κάθε αντικείμενο που απεικονίζεται στον καμβά
4) Σχεδίαση των αντικειμένων πάνω στον καμβά
5) Επανάληψη των βημάτων 3 και 4 για όλα τα αντικείμενα
6) Αφότου σχεδιαστούν όλα τα αντικείμενα, αναμονή για ένα ορισμένο διάστημα (που έχει οριστεί ανάλογα με τον επιθυμητό ρυθμό ανανέωσης των frames), εάν χρειάζεται και επιστροφή ξανά στο βήμα 2 για τον υπολογισμό και σχεδίαση του επόμενου frame

Στην συνέχεια όμως, με την χρήση των web workers έγινε ο απαραίτητος διαχωρισμός μεταξύ των διαδικασιών που μπορούν να εκτελεστούν στους web workers και αυτών που θα συνεχίσουν να εκτελούνται στο main thread, επομένως το παραπάνω μοντέλο εκτέλεσης άλλαξε και τα παραπάνω βήματα διαμορφώθηκαν ως εξής:

1) Δημιουργία ενός canvas element για την σχεδίαση της επιφάνειας στην οποία θα εμφανίζεται το παιχνίδι
2) Το κεντρικό thread αποστέλλει τα αρχικά δεδομένα για τα αντικείμενα σε όλους τους web workers με χρήση της μεθόδου postMessage()
3) Κάθε web worker λαμβάνει τα δεδομένα και αποστέλλει στο κεντρικό thread μια αναφορά ότι τα έλαβε
4) Το κεντρικό thread έπειτα στέλνει μια εντολή στους web workers για να ξεκινήσουν την εκτέλεση τους
5) Ο κάθε web worker υπολογίζει τις νέες θέσεις για τα αντικείμενα που του έχουν ανατεθεί και στη συνέχεια στέλνει τα δεδομένα αυτά στο κεντρικό thread

6) Εφόσον το κεντρικό thread συλλέξει τα δεδομένα για τις θέσεις των αντικειμένων από όλους του web workers, ελέγχει αν υπάρχει κάποια σύγκρουση μεταξύ των αντικειμένων αυτών ώστε να προβεί στις αντίστοιχες ενέργειες
7) Αφότου έχει προβεί στον παραπάνω έλεγχο ενημερώνει τους web workers σχετικά με το αποτέλεσμα αυτού και έπειτα ξεκινά να σχεδιάζει όλα τα αντικείμενα στον canva του παιχνιδιού
8) Το κεντρικό thread αλλά και οι web workers επαναλαμβάνουν τα βήματα 6 και 7 για κάθε frame που σχεδιάζεται

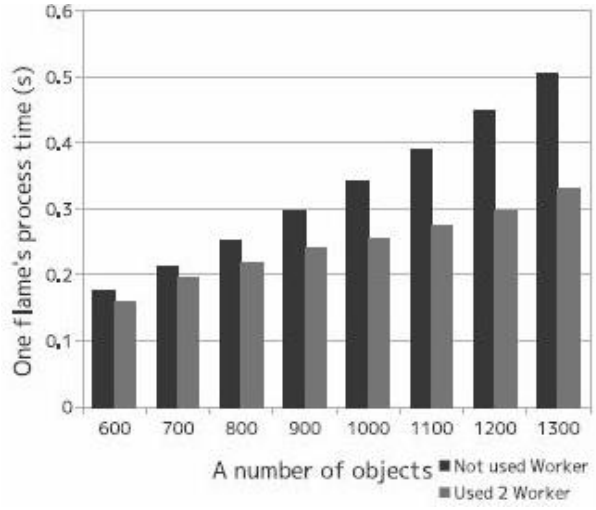
Συνεχίζοντας οι συγγραφείς του άρθρου (Parallelization Of Interactive Animation Software with Web Workers, 2013), πραγματοποίησαν μια σύγκριση των επιδόσεων μεταξύ της αρχικής έκδοσης (Single-threaded) και αυτής όπου γίνεται η χρήση web workers (Multi-threaded) όπου χρησιμοποιήθηκαν 2 threads, συγκρίνοντας τους χρόνους που χρειάζονται για την σχεδίαση του κάθε frame. Αξίζει να σημειωθεί ότι η εκτέλεση των παραπάνω εκδόσεων του παιχνιδιού έγινε σε ένα σύστημα με τα εξής χαρακτηριστικά:

• Λειτουργικό σύστημα: Fedora 16

• Επεξεργαστή: Intel(R) Core i7-2630QM CPU (4 cores, 8 Threads)

• Memory: 8GB

Παρακάτω βλέπουμε το διάγραμμα που παρουσιάζει την σύγκριση επιδόσεων μεταξύ της Single-Threaded (αρχικής έκδοσης) και της Multi-Threaded έκδοσης με την χρήση 2 web workers.



Εικόνα 7: Διάγραμμα Σύγκρισης Επιδόσεων του Arcade Game

Έχοντας μελετήσει τα αποτελέσματα από την σύγκριση των επιδόσεων των δύο εκδόσεων του παραπάνω παιχνιδιού καταλήγουμε στα εξής συμπεράσματα:

- Παρατηρείται ότι μέχρι τα 900 αντικείμενα ο χρόνος ανανέωσης του frame είναι αποδεκτός και από τις δύο υλοποιήσεις, έχοντας ως δεδομένο ότι η σχεδίαση του κάθε frame δεν θα πρέπει να διαρκεί περισσότερο από 0.3 δευτερόλεπτα.
- Από τα 900 μέχρι τα 1200 αντικείμενα μόνο η υλοποίηση με χρήση των web workers συνεχίζει να αποδίδει ικανοποιητικό χρόνο ανανέωσης των frames.
- Επομένως όπως φαίνεται από το διάγραμμα, η υλοποίηση που κάνει χρήση των web workers μπορεί να χειριστεί περίπου 30% περισσότερα αντικείμενα από το αρχικό.

Αξίζει να σημειωθεί ότι στο πλαίσιο των δοκιμών έγινε χρήση τεσσάρων web workers αλλά το αποτέλεσμα ήταν σχεδόν το ίδιο με την περίπτωση των 2 web workers, πράγμα το οποίο δικαιολογείται από το γεγονός ότι ένα σημαντικό μέρος της επεξεργασίας γίνεται στο κομμάτι της σχεδίασης των αντικειμένων στον canva, το οποίο μπορεί να γίνει μόνο από το κεντρικό thread καθώς μόνο αυτό έχει πρόσβαση στο DOM. (Parallelization Of Interactive Animation Software, 2013).

2.5.2 Ανάλυση της Δυνατότητας για Κλιμάκωση της Απόδοσης των Javascript Εφαρμογών με την χρήση Web Workers

Στο άρθρο αυτό οι Javier Verdú, Juan José Costa και Alex Pajuelo κάνουν μια περιγραφή των δυνατοτήτων βελτίωσης της απόδοσης των web εφαρμογών χρησιμοποιώντας πολλαπλούς web workers. Σκοπός του άρθρου είναι να παρουσιάσει πιθανούς τρόπους για να υπολογίσουμε τον βέλτιστο αριθμό web worker λαμβάνοντας κάθε φορά υπόψη το τρέχον φόρτο εργασίας του συστήματος στο οποίο αυτοί εκτελούνται. Γίνεται μελέτη δύο περιπτώσεων χρήσης, τα οποία αντιπροσωπεύουν δύο διαφορετικά μοντέλα εκτέλεσης των web workers τα οποία παρατίθενται παρακάτω:

- Multiple asynchronous workers: Στο μοντέλο αυτό, οι web workers αναλαμβάνουν την εκτέλεση μιας εργασίας και μόλις τελειώσουν αναλαμβάνουν άμεσα μια νέα εργασία χωρίς να υπάρχει εξάρτηση ή συγχρονισμός μεταξύ τους.

- Multiple synchronous workers: Εδώ, όλοι οι web workers αναλαμβάνουν να εκτελέσουν μια εργασία αλλά πριν προχωρήσουν στην επόμενη θα πρέπει να περιμένουν να ολοκληρώσουν την εκτέλεση των εργασιών και οι υπόλοιποι web workers.

Η 1η περίπτωση χρήσης αφορά το μοντέλο των multiple asynchronous workers και χρησιμοποιεί την εφαρμογή HashApp, η οποία είναι μια εφαρμογή που χρησιμοποιεί dedicated web workers οι οποίοι υπολογίζουν md5 hashes προκειμένου να αποκωδικοποιήσουν ένα κείμενο.

Η 2η περίπτωση χρήσης αφορά το μοντέλο των multiple synchronous workers και χρησιμοποιεί την εφαρμογή RayApp, η οποία απεικονίζει μια κινούμενη σκηνή, στην οποία χρησιμοποιούνται οι web workers για την σχεδίαση του κάθε frame.

Οι Javier Verdú, Juan José Costa και Alex Pajuelo για τα συγκριτικά test (benchmarks) των 2 case studies χρησιμοποιήσαν τους πιο δημοφιλείς browsers Chrome 42, Firefox 37, IE 11 και τα εκτελέσαν σε ένα σύστημα με τα παρακάτω χαρακτηριστικά:

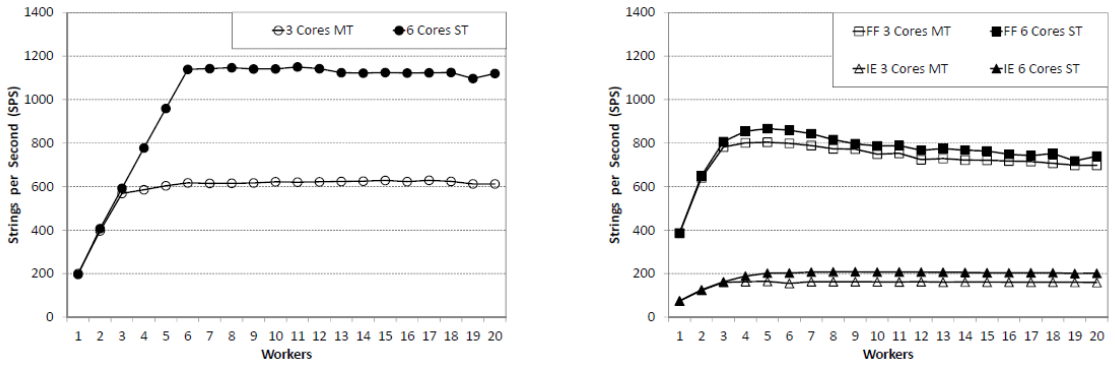
- Επεξεργαστή: Intel Core i7-3960X 3.3GHz (6 cores, 12 threads)
- Μνήμη: 16GB RAM
- Κάρτα γραφικών: Nvidia GTX560
- Λειτουργικό: Windows Server 2008 R2

Παράλληλα έγινε χρήση της εφαρμογής Process Explorer v15.21 μέσα από την οποία έγιναν οι απαραίτητες ρυθμίσεις προκειμένου να αλλάξει ο αριθμός των λογικών πυρήνων σε 6 από 12, για της ανάγκες μελέτης των παραπάνω περιπτώσεων χρήσης.

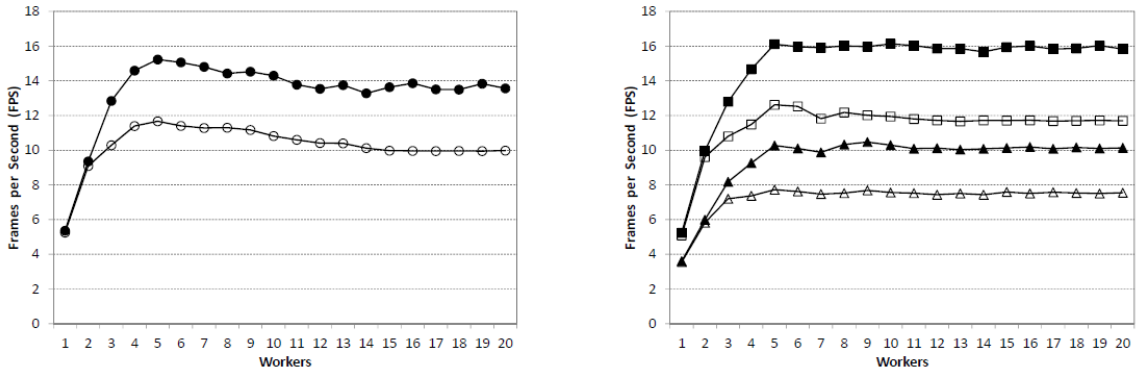
Εκτελώντας την εφαρμογή HashApp και χρησιμοποιώντας 6 λογικούς πυρήνες σε δύο διαφορετικές αρχιτεκτονικές, single-threaded (6 cores – 6 threads) και multi-threaded (3 cores – 6 threads) παρατηρούνται τα εξής (Εικόνα 8):

- Η εφαρμογή με την single-threaded αρχιτεκτονική έχει σχεδόν διπλάσιες επιδόσεις σε σχέση με την multithreaded multi-core αρχιτεκτονική όταν χρησιμοποιούνται 6 web workers

- Και στις δύο αρχιτεκτονικές βλέπουμε ότι η απόδοση της εφαρμογής βελτιώνεται γραμμικά, καθώς αυξάνεται ο αριθμός των web workers μέχρι να φτάσουμε στον αριθμό των φυσικών πυρήνων (πραγματικοί πυρήνες του επεξεργαστή)
- Όταν χρησιμοποιούνται περισσότεροι web workers από τους λογικούς πυρήνες, η απόδοση παραμένει κατά μέσο όρο σταθερή



Εικόνα 8: Διαγράμματα Εκτέλεσης Εφαρμογής HashApp



Εικόνα 9: Διαγράμματα Εκτέλεσης Εφαρμογής RayApp

Εκτελώντας στην συνέχεια την εφαρμογή RayApp χρησιμοποιώντας 6 λογικούς πυρήνες και δύο διαφορετικές αρχιτεκτονικές single-threaded (6 cores – 6 threads) και multi-threaded (3 cores – 6 threads) παρατηρούμε τα εξής (Εικόνα 9):

- Η εφαρμογή με την single-threaded αρχιτεκτονική παρουσιάζει καλύτερες επιδόσεις από την multi-threaded με την χρήση 3 ή περισσότερων web workers αλλά όχι με τόσο μεγάλη διαφορά όπως με την εφαρμογή Hash BruteForcer
- Οι επιδόσεις των δύο αρχιτεκτονικών δεν απέχουν πολύ λόγω του μοντέλου σύγχρονης εκτέλεσης των web workers που χρησιμοποιεί η εφαρμογή αυτή, κατά

το οποίο υπάρχουν χρονικά διαστήματα όπου κάποιοι web workers παραμένουν αδρανής

- Σε αντίθεση με την εφαρμογή HashApp, η χρήση περισσότερων web workers από τους λογικούς πυρήνες οδηγεί σε μια μικρή μείωση της απόδοσης

Όπως φαίνεται και στα παραπάνω διαγράμματα (Εικόνες 8 και 9), υπάρχουν διαφορές στις επιδόσεις των δύο αρχιτεκτονικών ανάλογα με τον browser στον οποίο εκτελούνται. Συγκεκριμένα η επίδοση της single-threaded (6 cores – 6 threads) αρχιτεκτονικής για την εφαρμογή HashApp στον Firefox είναι κατά 8% μεγαλύτερη από την επίδοση της multi-threaded (3 cores – 6 threads) και αντίστοιχα κατά 28% στον IE, όταν στον chrome υπάρχει σχεδόν 100% αύξηση απόδοσης μεταξύ των 2 αρχιτεκτονικών.

Αντιθέτως, η επιδόσεις των 2 αρχιτεκτονικών στην εφαρμογή RayApp είναι πολύ κοντά και στους 3 browsers και πάλι λόγω των περιορισμών του μοντέλου σύγχρονης εκτέλεσης.

Συγκεκριμένα η single-threaded αρχιτεκτονική δείχνει μια βελτίωση απόδοσης 3.07x στον Firefox και 2.91x στον IE, ενώ στην multi-threaded έχουμε μια βελτίωση απόδοσης 2.47x στον Firefox και 2.17x στον IE.

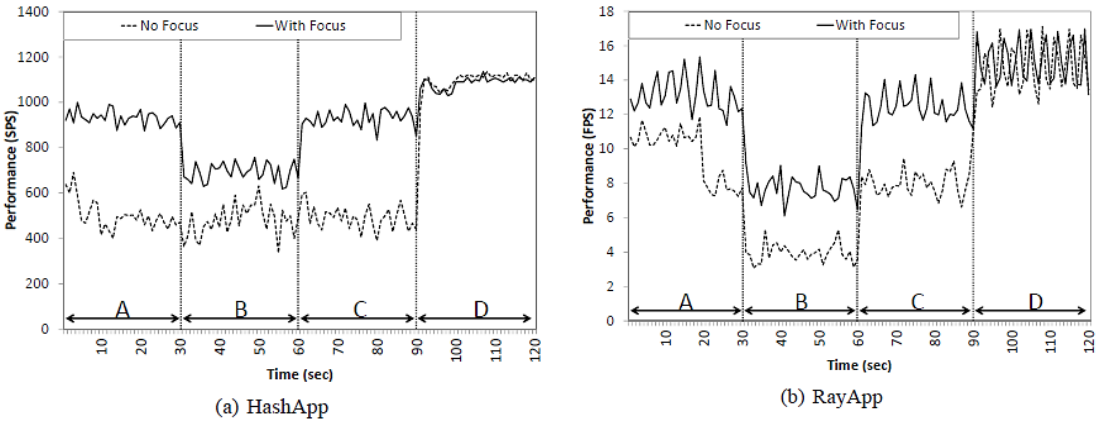
Προχωρώντας, προκειμένου να γίνει κατανοητό το πόσο μπορεί να επηρεαστεί η απόδοση ενός συστήματος από τις εφαρμογές που μπορεί να τρέχουν στο παρασκήνιο, οι δύο παραπάνω εφαρμογές (HashApp και RayApp) εκτελέστηκαν στον chrome χρησιμοποιώντας έξι πυρήνες και έξι threads. Η προσομοίωση των εφαρμογών παρασκηνίου έγινε με την παράλληλη εκτέλεση της ίδιας κάθε φορά εφαρμογής, σε 2 διαφορετικούς browsers (Firefox και IE) χρησιμοποιώντας ο καθένας από αυτούς 3 web workers. Έτσι οι εφαρμογές παρασκηνίου μπορούν να κάνουν χρήση του 50% της CPU καθώς χρησιμοποιούν τα 6 από τα 12 συνολικά threads του επεξεργαστή.

Στην εικόνα 10 φαίνονται τα διαγράμματα από την εκτέλεση των εφαρμογών HashApp και RayApp λαμβάνοντας υπόψη την εκτέλεση των εφαρμογών παρασκηνίου που εκτελούνται παράλληλα. Όπως φαίνεται και στο χρονοδιάγραμμα, υπάρχουν τέσσερις διαφορετικές φάσεις κατά τις οποίες αλλάζει ο τρόπος εκτέλεσης των εφαρμογών παρασκηνίου. Στην φάση Α γίνεται εκτελείται μόνο η μία από της δύο εφαρμογές παρασκηνίου, στην φάση Β εκτελούνται και οι δύο ταυτόχρονα και στην Γ έχει ολοκληρωθεί η εκτέλεση της πρώτης εφαρμογής και εκτελείται και πάλι μόνο μια.

Τέλος στην φάση Δ έχουν τελειώσει την εκτέλεση τους και οι δύο εφαρμογές παρασκηνίου οπότε η επιδόσεις τις εφαρμογής δεν επηρεάζονται.

Ένας ακόμη παράγοντας ο οποίος προσμετράται είναι αν το τρέχον επιλεγμένο το παράθυρο είναι αυτό της κεντρικής εφαρμογής (παρουσιάζεται στην εικόνα 10 με ενωμένη γραμμή) ή εάν έχει επιλεχθεί κάποια παράλληλα εκτελούμενες (παρουσιάζεται στην εικόνα 10 με διακεκομμένη γραμμή). Αυτό επηρεάζει καθώς το λειτουργικό ανάλογα με το ποιο παράθυρο είναι ενεργό αλλάζει και την αντίστοιχη προτεραιότητα έχει στην εκτέλεση του.

Όπως φαίνεται και στα διαγράμματα (εικόνα 10), όταν η κεντρική εφαρμογή είναι επιλεγμένη (έχει την μέγιστη προτεραιότητα) και τρέχει μόνο η μια από τις δύο εφαρμογές στο παρασκήνιο (φάση Α) οι εφαρμογές HashApp και RayApp παρουσιάζουν μείωση της επίδοσης κατά 15% και 17% αντίστοιχα. Ενώ όταν εκτελούνται παράλληλα και οι δύο εφαρμογές στο παρασκήνιο (φάση Β) η μείωση στις επιδόσεις των εφαρμογών αυτών είναι αρκετά μεγαλύτερη και φτάνει το 37% και 49% αντίστοιχα. Στην περίπτωση που το επιλεγμένο παράθυρο είναι μια από τις εφαρμογές παρασκηνίου υπάρχει ακόμη μεγαλύτερη μείωση στις επιδόσεις. Συγκεκριμένα για την εφαρμογή HashApp μείωση φτάνει κατά μέσο όρο το 55%, είτε εκτελείται μια, είτε με δύο παράλληλες εφαρμογές παρασκηνίου. Ενώ η εφαρμογή RayApp παρουσιάζει μια μείωση που ξεκινά από το 45% όταν εκτελείται μια εφαρμογή παρασκηνίου και φτάνει το 73% όταν εκτελούνται και οι δύο εφαρμογές παρασκηνίου.



(a) HashApp (b) RayApp

Εικόνα 10: Διαγράμματα Εκτέλεσης Παράλληλα με Εφαρμογές Παρασκηνίου

Κλείνοντας, το άρθρο αυτό καταλήγει στο συμπέρασμα ότι είναι πολύ δύσκολο να υπολογιστεί ο βέλτιστος αριθμός των πυρήνων για χρήση σε μια εφαρμογή όπου μπορεί να εκτελεστεί κάνοντας χρήση του web workers API καθώς ο αριθμός αυτός εξαρτάται από διάφορους παράγοντες όπως είναι:

- Το μοντέλο εκτέλεσης των web workers, σύγχρονο ή ασύγχρονο όπως αναφέρθηκε παραπάνω
- Την αρχιτεκτονική του επεξεργαστή (Single-threaded ή Multi-threaded cores)
- Ο Web browser στο οποίο εκτελείται η εφαρμογή

Στις περισσότερες περιπτώσεις όμως ένας μικρός σχετικά αριθμός web workers παρουσιάζει παρόμοια ή ελάχιστα καλύτερες επιδόσεις από έναν μεγάλο αριθμό. Ακόμη και έτσι και πάλι θα πρέπει να ληφθεί υπόψη ο αριθμός των διεργασιών που εκτελούνται στο παρασκήνιο.

2.5.3 Δυναμική Διαχείριση των Web Workers για την Υλοποίηση Εφαρμογών Javascript με χρήση Παράλληλης Επεξεργασίας

Στο άρθρο αυτό οι Javier Verdú, Juan José Costa και Alex Pajuelo παρουσιάζουν την δημιουργία ενός αλγόριθμου για την δυναμική εκμετάλλευση των web workers ανάλογα με την τρέχουσα χρήση των πόρων του συστήματος. Δημιούργησαν λοιπόν έναν αλγόριθμο ο οποίος κάνει χρησιμοποιεί μια λίστα με έναν προεπιλεγμένο αριθμό web workers, οι οποίοι ανάλογα με το τρέχον φόρτο και τις διεργασίες παρασκηνίου λαμβάνει απόφαση για τον βέλτιστο αριθμό web workers οι οποίοι πρέπει να εκτελεστούν ώστε η εφαρμογή που τρέχει ο χρήστης να έχει την καλύτερη απόδοση.

Η παραπάνω λίστα με τους web workers, δημιουργείται κάθε φορά κατά την εκκίνηση της εφαρμογής, με τον μέγιστο δυνατό αριθμό και έπειτα όλοι οι web workers παραμένουν σε αδράνεια μέχρι να ζητηθεί η εκκίνηση τους από τον αλγόριθμο. Για να κατανοήσετε καλύτερα τον τρόπο που λειτουργεί ο αλγόριθμος αυτός, αρκεί να δείτε τα επιμέρους τμήματα από τα οποία αποτελείται, τα οποία είναι τα εξής:

- Μια λίστα η οποία περιλαμβάνει τις 5 τελευταίες μετρήσεις απόδοσης της εφαρμογής (Performance Queue)
- Μεταβλητή που ορίζει το ποσοστό αύξησης της απόδοσης που πρέπει να έχει παρουσιάσει η εφαρμογή (από προηγούμενη προσθήκη web worker) ώστε να ξεκινήσει να εκτελείται ένας επιπλέον web worker
- Μεταβλητή που ορίζει το ποσοστό μείωσης της απόδοσης που πρέπει να έχει παρουσιάσει η εφαρμογή (από προηγούμενη προσθήκη web worker) ώστε να ξεκινήσει να μην προστεθεί νέος web worker. Εδώ πρέπει να διευκρινιστεί ότι ο web worker που οδήγησε σε μείωση της απόδοσης θα παραμείνει σε αδράνεια μόλις ολοκληρώσει την εργασία του (Εικονική μείωση του αριθμού των Web Workers).

Συνεχίζοντας οι Javier Verdú, Juan José Costa και Alex Pajuelo, έχοντας υλοποιήσει τον παραπάνω αλγόριθμο σε δύο υπάρχουσες εφαρμογές, HashApp και RayApp εκτέλεσαν μια σειρά από benchmarks προκειμένου να αποδείξουν την αξία του παραπάνω αλγόριθμου σε πραγματικές συνθήκες. Το σύστημα στο οποίο εκτέλεσαν τα συγκριτικά τεστ έχει τα εξής χαρακτηριστικά:

Επεξεργαστή: Intel Core i7 3960X 3.3GHz (6 cores, 12 threads)

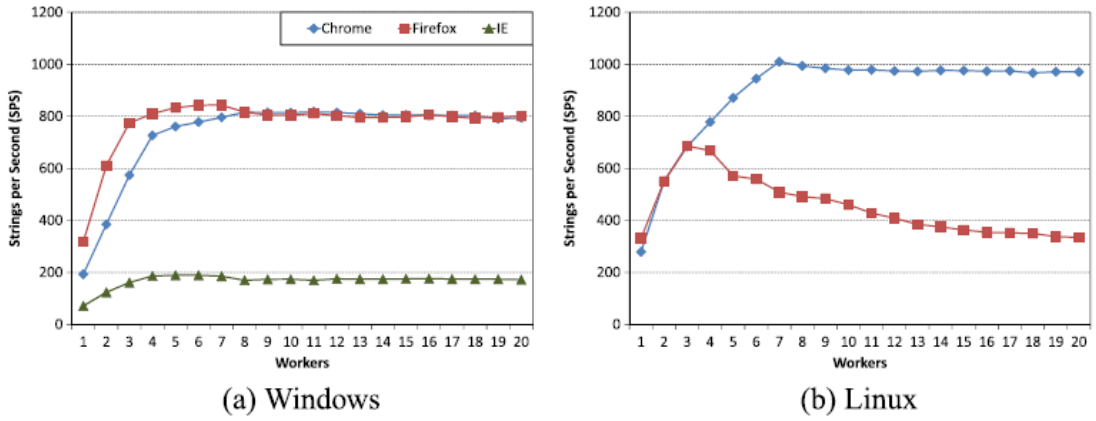
Μνήμη: 16 GB DDR3

Κάρτα γραφικών: Nvidia GeForce GTX 560

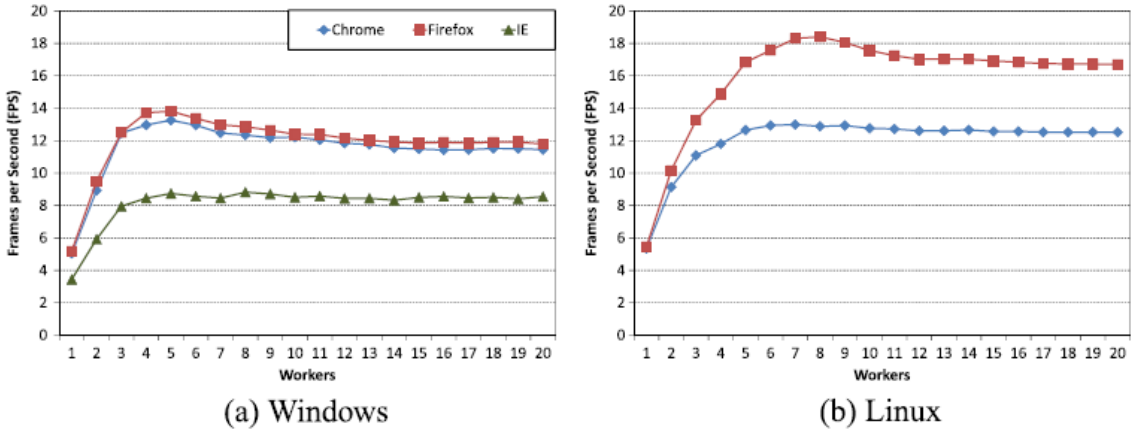
Λειτουργικό: Windows Server 2008 R2 / Ubuntu 14.04 LTS

Προκειμένου τα συγκριτικά τεστ να μην επηρεάζονται σε σημαντικό βαθμό από τις διεργασίες που τρέχουν στο παρασκήνιο έγινε χρήση του προγράμματος Process Explorer v15.4, όπου ορίστηκαν δύο από τους έξι πυρήνες για την εκτέλεση των απαραίτητων υπηρεσιών και διεργασιών του λειτουργικού και οι υπόλοιποι τέσσερις για χρήση στους Web Browsers, προσομοιώνοντας έτσι έναν τετραπύρηνο intel core i7 επεξεργαστή με δυνατότητα multithreading (4 cores, 8 threads).

Τέλος, οι Web Browsers που χρησιμοποίησαν οι συγγραφείς του άρθρου για την εκτέλεση των τεστ είναι ο Google Chrome v42.0.2311.90m, Mozilla Firefox v37.0.2, and Microsoft Internet Explorer v11.0.9600.16476. Στην συνέχεια παρουσιάζονται τα σχετικά γραφήματα τα οποία πιστοποιούν την αποτελεσματικότητα του αλγόριθμου που υλοποιήθηκε με μετρήσιμα μεγέθη.



Εικόνα 11: Διάγραμμα εκτέλεσης της εφαρμογής HashApp με 8 λογικούς πυρήνες



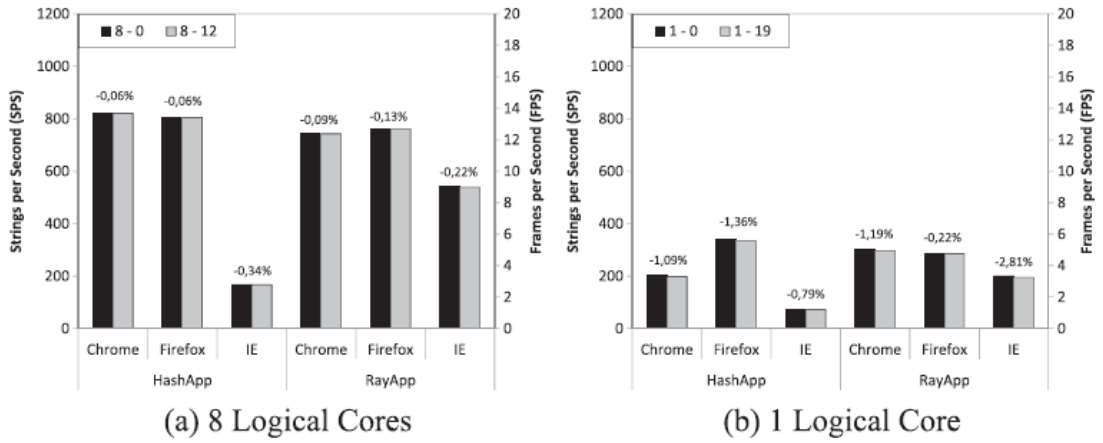
Εικόνα 12: Διάγραμμα εκτέλεσης της εφαρμογής RayApp με 8 λογικούς πυρήνες

Σύμφωνα με τα αποτελέσματα που φαίνονται και στα παραπάνω διαγράμματα οι Javier Verdú, Juan José Costa και Alex Pajuelo καταλήγουν στα παρακάτω συμπεράσματα:

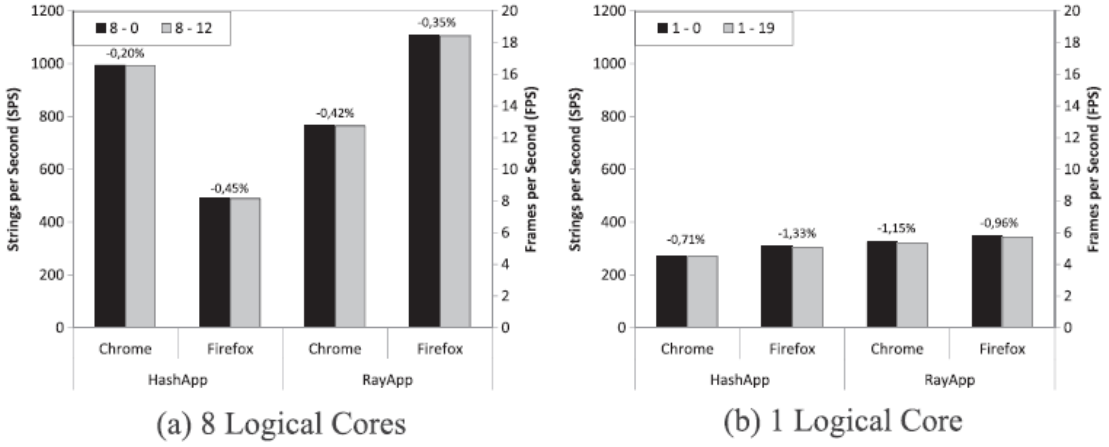
- Η απόδοση μιας εφαρμογής επηρεάζεται από τον αριθμό των workers σε διαφορετικό βαθμό ανάλογα με το λειτουργικό στο οποίο εκτελείται η εφαρμογή. Για παράδειγμα η απόδοση της εφαρμογής HashApp στον Firefox με λειτουργικό Ubuntu, μειώνεται σημαντικά εάν αυξηθεί αρκετά ο αριθμός των workers, γεγονός που δεν συμβαίνει στο λειτουργικό Windows με τον ίδιο browser

- Το μοντέλο εκτέλεσης των workers (σύγχρονη ή ασύγχρονη εκτέλεση) μπορεί να επηρεάσει σημαντικά την δυνατότητα βελτίωσης της απόδοσης μια εφαρμογής, με την αύξηση του αριθμού των web workers
- Συνήθως δεν χρειάζεται να χρησιμοποιηθεί μεγάλος αριθμός web workers για να φτάσουμε στην βέλτιστη απόδοση μιας εφαρμογής
- Επομένως δεν είναι εύκολο να καθοριστεί ένα σταθερός αριθμός workers ως βέλτιστος

Συνεχίζοντας την μελέτης τους, οι συγγραφείς του άρθρου εκτέλεσαν μερικά benchmarks προκειμένου να δουν το αντίκτυπο που έχουν οι web workers όταν έχουν δημιουργηθεί αλλά παραμένουν αδρανείς. Παρακάτω παρατίθενται τα σχετικά γραφήματα.



Εικόνα 13: Απεικόνιση αντίκτυπου αδρανών web worker σε Windows



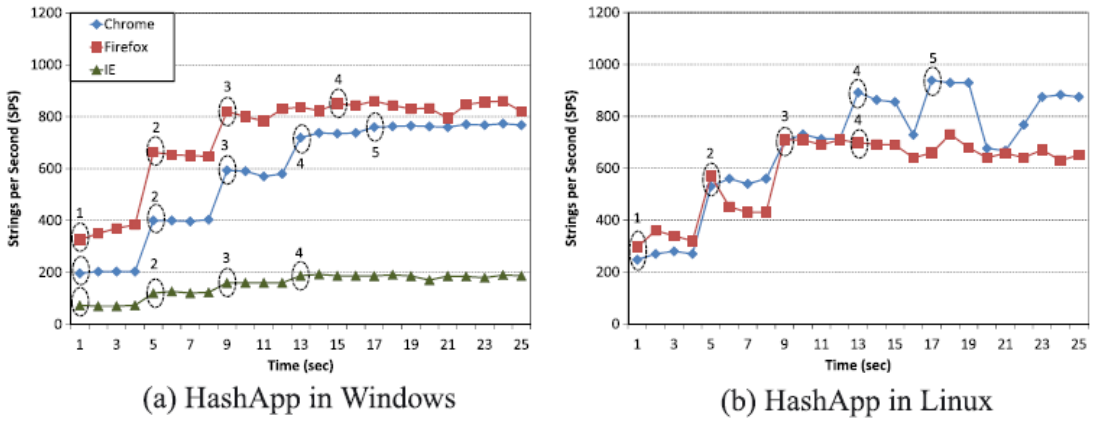
Εικόνα 14: Απεικόνιση αντίκτυπου αδρανών web worker σε Linux

Στο πρώτο benchmark έγινε εκτέλεση των εφαρμογών HashApp και RayApp με χρήση 8 λογικών πυρήνων, πρώτα χωρίς την χρήση web workers και έπειτα με την χρήση 12 αδρανών web workers. Τα αποτελέσματα (Εικόνα 13 σχήμα (a) και Εικόνα 14 σχήμα (a)) έδειξαν πως η μείωση των επιδόσεων δεν ήταν σημαντική καθώς κινήθηκε από 0.06 έως 0.45% για την εφαρμογή HashApp και από 0.09 έως 0.42% για την εφαρμογή RayApp.

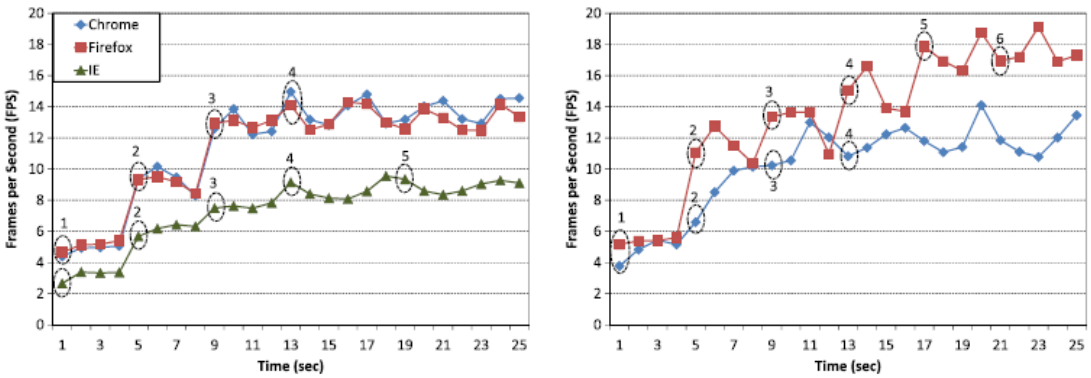
Ενώ συνεχίζοντας στο δεύτερο benchmark έγινε η εκτέλεση των παραπάνω εφαρμογών πρώτα με χρήση ενός μόνο web worker χωρίς αδρανής workers και έπειτα με την χρήση 19 αδρανών web workers. Παρόλο που θεωρείται η χρήση 19 αδρανών web workers ως worst case scenario τα αποτελέσματα δεν είναι απογοητευτικά. Το αντίθετο μάλιστα καθώς η μείωση επίδοσης για την εφαρμογή HashApp κινείται μεταξύ του 0.71% και 1.36% ενώ για την εφαρμογή RayApp η μείωση αντίστοιχα μεταξύ του 0.22% και 2.81%.

Τα παραπάνω αποτελέσματα λοιπόν οδηγούν στο συμπέρασμα ότι οι σχεδιαστές μια εφαρμογής μπορούν να δημιουργήσουν μια μεγάλη λίστα με web workers με μικρό κόστος στις επιδόσεις και στην συνέχεια να αποφασίσουν δυναμικά τον αριθμό των web workers που πρέπει να χρησιμοποιηθεί κάθε στιγμή της εκτέλεσης.

Τέλος, οι Javier Verdú, Juan José Costa και Alex Pajuelo, ολοκληρώνοντας την μελέτη τους και έχοντας αποδείξει παραπάνω ότι μπορεί να χρησιμοποιηθεί μια μεγάλη λίστα με web workers, η οποία θα αρχικοποιηθεί κατά την εκκίνηση της εφαρμογής με μηδαμινό κόστος σε πόρους, εκτελούν τις δύο παραπάνω εφαρμογές (HashApp και RayApp) με χρήση του αλγόριθμου που υλοποίησαν, προκειμένου να δουν την εφαρμογή που έχει σε πραγματικές συνθήκες εκτέλεσης. Στην συνέχεια παρουσιάζονται τα σχετικά διαγράμματα και παρατίθενται τα συμπεράσματα που προκύπτουν από αυτά.



Εικόνα 15: Εκτέλεσης της Εφαρμογής HashApp με χρήση του Αλγόριθμου



(c) RayApp in Windows (d) RayApp in Linux

Εικόνα 16: Εκτέλεσης της Εφαρμογής HashApp με χρήση του Αλγόριθμου

Στη συνέχεια, οι συγγραφείς του άρθρου, έχοντας εκτελέσει τις δύο παραπάνω εφαρμογές, με χρήση του αλγόριθμου που δημιούργησαν και αλλά και χωρίς την χρήση αυτού, καταλήγουν στα παρακάτω συμπεράσματα μετά από μελέτη των σχετικών αποτελεσμάτων:

- Ο αλγόριθμος στις περίπτωση του Internet Explorer καταφέρνει να βρει τον βέλτιστο αριθμό web workers που απαιτείται και από τις δύο εφαρμογές (HashApp και RayApp)
- Στην περίπτωση εκτέλεσης του αλγόριθμου στον Firefox, μπόρεσε να πλησιάσει πολύ κοντά στον βέλτιστο αριθμό για την εφαρμογή RayApp ενώ για την εφαρμογή HashApp. Παρ' όλα αυτά η διαφορά απόδοσης μεταξύ της εκτέλεσης που κάνει χρήση του βέλτιστου αριθμού και της εκτέλεσης του αλγόριθμου είναι μικρότερη του 4%.
- Όσον αφορά την εκτέλεση στον Chrome και πάλι δεν βρίσκει τον βέλτιστο αριθμό αλλά η διαφορά στην απόδοση με την εκτέλεση που έχει τον βέλτιστο αριθμό workers είναι 7.36% για την εφαρμογή HashApp και 2.23% για την εφαρμογή RayApp
- Ο αλγόριθμος όπως φαίνεται και από τα παραπάνω αποτελέσματα πλησίασε πολύ στις βέλτιστες τιμές των web workers αλλά και όταν δεν το κατάφερε η απόκλιση από την βέλτιστη απόδοση δεν ήταν αρκετά σημαντική.

Τέλος, έχοντας ολοκληρώσει την μελέτη τους οι συγγραφείς καταλήγουν στα εξής συμπεράσματα:

- Είναι δύσκολο να υλοποιηθεί ένας αλγόριθμος που να χρησιμοποιεί τον ελάχιστο αριθμό web workers, αποδίδοντας παράλληλα την καλύτερη επίδοση
- Η χρήση εργαλείων για την πρόβλεψη του αριθμού των threads που μπορεί να χειριστεί ένα σύστημα, μπορεί να οδηγήσουν σε λανθασμένα συμπεράσματα με αποτέλεσμα τελικά την μείωση της απόδοσης του συστήματος σε πραγματικές συνθήκες
- Ο καλύτερος τρόπος εκτίμησης του βέλτιστου αριθμού των threads που θα πρέπει να δημιουργηθούν σε μια εφαρμογή θα πρέπει να λαμβάνει υπόψη τους διαθέσιμους πόρους του συστήματος σε πραγματικό χρόνο.

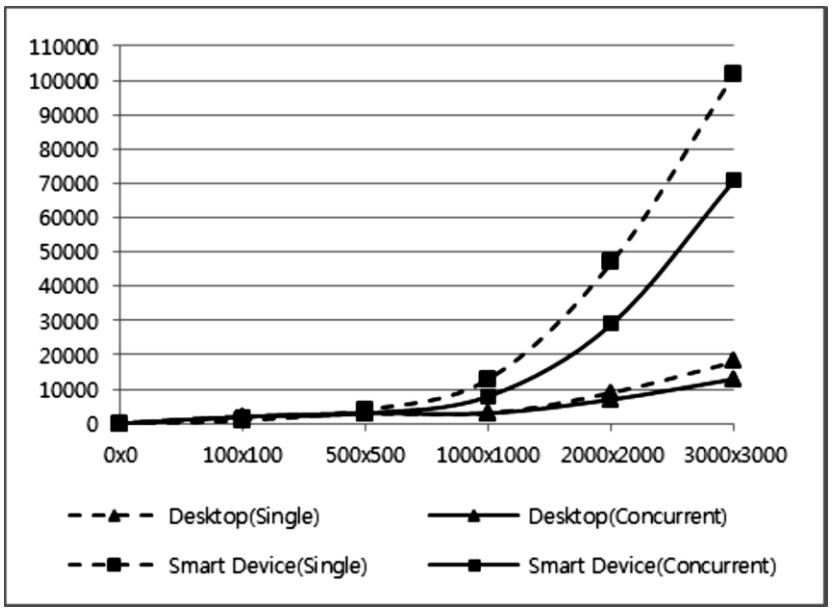
2.5.4 Πολυνηματικός Προγραμματισμός για την Απεικόνιση Γεωγραφικών Δεδομένων με χρήση WebGL και Web Workers

Στο άρθρο αυτό οι Hyung Woo Kim και Yang-Won Lee μελέτησαν την δυνατότητα για απεικόνιση raster δεδομένων ως εικόνα σε web browsers κάνοντας χρήση του WebGL API και αξιοποιώντας ταυτόχρονα τους Web Workers ώστε να μοιραστεί το υπολογιστικό βάρος αποδοτικά στο εκάστοτε σύστημα. Στη συνέχεια, έχοντας υλοποιήσει το απαραίτητο λογισμικό απεικόνισης raster δεδομένων (με χρήση των παραπάνω API), προκειμένου να γίνει σαφές το πλεονέκτημα που προκύπτει από την αξιοποίηση των Web Workers, εκτέλεσαν κάποια benchmarks για την υλοποίηση αυτή, κάνοντας πρώτα χρήση ενός μόνο thread (main thread) και έπειτα με χρήση πολλαπλών threads (Web Workers).

Κατά την φάση του benchmarking οι συγγραφείς του άρθρου χρησιμοποίησαν δύο διαφορετικές συσκευές και πέντε διαφορετικά μεγέθη raster data. Οι συσκευές στις οποίες έγιναν οι δοκιμές έχουν τα εξής χαρακτηριστικά:

- Ένα Desktop PC με επεξεργαστή Intel i5 3550 Quad Core
- Ένα Tablet (Nexus 7) με επεξεργαστή Nvidia Quad Core Tegra 3

Όσον αφορά τα μεγέθη raster data που χρησιμοποιήθηκαν είναι: 100x100, 500x500, 1000x1000, 2000x2000 και 3000x3000. Ακόμη, ο αριθμός των web workers που χρησιμοποιήθηκε είναι 4 workers, όσοι είναι κ οι επεξεργαστές των δύο συγκρινόμενων συσκευών, με το συνολικό μέγεθος των raster να διαιρείται σε τμήματα των 100 και έπειτα το κάθε τμήμα να εκχωρείται σε έναν διαφορετικό worker ώστε να γίνει ίσος καταμερισμός της συνολικής επεξεργασίας. Τέλος, οι browsers που χρησιμοποιήθηκαν είναι οι εκδόσεις του Chrome 28 σε Desktop και Mobile Version (Chrome For Android) αντίστοιχα.



Εικόνα 17: Διάγραμμα Χρόνου Εκτέλεσης για Απεικόνιση Raster Data

Όπως φαίνεται και από το παραπάνω διάγραμμα η διαφορά μεταξύ των μεθόδων single-thread και concurrent αρχίζει να είναι ξεκάθαρη από την επεξεργασία δεδομένων μεγέθους 500x500 και μετά. Είναι σημαντικό να αναφερθεί ότι στην περίπτωση δεδομένων raster μεγέθους 100x100 χρησιμοποιήθηκε μόνο ένας worker στην concurrent έκδοση, γι' αυτό και παρατηρούμε ότι ο χρόνος εκτέλεσης είναι ίδιος με την single-threaded έκδοση. Αυτό συνέβη καθώς ορίστηκε στην υλοποίηση της concurrent έκδοσης το συνολικό μέγεθος των raster data να διαιρείται σε τμήματα των 100 και επομένως σε αυτήν την περίπτωση δεν έγινε κάποια διαίρεση στην περίπτωση αυτή.

Όσον αφορά την Desktop συσκευή βλέπουμε ότι τα αποτελέσματα μεταξύ των δύο benchmarks (single και concurrent) δεν δείχνουν σημαντική διαφορά στην απόδοση ιδιαίτερα μέχρι την χρήση μεγέθους δεδομένων 2000x2000 και 3000x3000. Αυτό φαίνεται και από το μέσο όρο του χρόνου εκτέλεσης (ο μέσος όρος υπολογίζεται με βάση τους

χρόνους εκτέλεσης από όλα τα μεγέθη raster data 500x500, 1000x1000 κλπ.), ο οποίος στη single-thread υλοποίηση ήταν 9 δευτερόλεπτα ενώ με την concurrent υλοποίηση 7.25 δευτερόλεπτα. Ενώ αντίστοιχα στην περίπτωση του tablet η διαφορά μεταξύ των single-thread και concurrent thread μεθόδων εκτέλεσης είναι πολύ μεγαλύτερη. Με τον μέσο όρο χρόνου εκτέλεσης να φτάνει τα 43 δευτερόλεπτα στην single-thread υλοποίηση και τα 28.2 δευτερόλεπτα στην concurrent υλοποίηση.

Παρατηρούμε επίσης ότι από τα δύο συστήματα στα οποία έγινε η εκτέλεση των συγκριτικών τεστ, το σύστημα που φάνηκε να επωφελείται περισσότερο είναι το Tablet, το οποίο παρόλο που είχε αρκετά μεγαλύτερους χρόνους εκτέλεσης από το Desktop έδειξε πως μπορεί να έχει πολύ μεγαλύτερη βελτίωση (34,4%) με την χρήση των Web Workers.

2.5.5 Παράλληλη Υλοποίηση Κρυπτοσυστημάτων Δημοσίου Κλειδιού με την χρήση Web Workers

Σε αυτό το άρθρο οι Takuya Sumi, Tsukasa Ishiguro, Shinsaku Kiyomoto, Yutaka Miyake, Toru Kobayashi, & Tsuyoshi Takagi κάνουν μια μελέτη όσον αφορά τις δυνατότητες που μπορούν να δώσουν οι Web Workers στην ασφάλεια των επικοινωνιών μέσω διαδικτύου αξιοποιώντας την τεχνολογία αυτή στην κρυπτογραφία δημοσίου κλειδιού. Προκειμένου να αποδείξουν την χρησιμότητα των HTML5 Web Workers στον τομέα της ασφάλειας επικοινωνιών, υλοποιήσαν δύο συστήματα κρυπτογράφησης δημοσίου κλειδιού σε δημοφιλής web browsers όπως Internet Explorer, Google Chrome, Opera και Firefox.

Υλοποίησαν λοιπόν έναν 3072-bit RSA αλγόριθμο και ένα σχήμα Rainbow με χρήση ενός συστήματος F_{31} (multivariate quadratic polynomials) και σε συνδυασμό με την χρήση των Web Workers απέδειξαν ότι οι παραπάνω αλγόριθμοι, μπορούν να επαληθεύσουν μια ψηφιακή υπογραφή σε πραγματικό χρόνο. Για να το πετύχουν αυτό πραγματοποίησαν κάποια benchmarks χρησιμοποιώντας δύο διαφορετικά συστήματα με τα εξής χαρακτηριστικά:

1) Ένα υπολογιστή με επεξεργαστή AMD Phenom II X6 1090T 3.2 GHz (6-cores), μνήμη 4 GB RAM και λειτουργικό Windows 7 64bit
2) Ένα tablet Nexus 7 με επεξεργαστή NVIDIA Tegra 3 T30L (Cortex-A9) 1.3GHz (4 cores), 1 GB RAM και λειτουργικό Android 4.3

Παρακάτω παρατίθενται τα αποτελέσματα από τα συγκριτικά τεστ που πραγματοποιήθηκαν σε τέσσερις δημοφιλείς φυλλομετρητές (Chrome, Firefox, IE, Opera) και που αφορούν τους χρόνους εκτέλεσης που απαιτούνται για την δημιουργία και επαλήθευση των ψηφιακών υπογραφών χρησιμοποιώντας τους παραπάνω δύο αλγόριθμους.

TABLE I
RUNNING TIME OF RSA (MS)

Operation	Device	IE[1]	Chrome	Opera	Firefox
Signature generation[2]	PC	381.50	162.59	164.29	168.64
	Nexus 7	N/A	1304.94	1176.18	1048.57
Signature verification	PC	19.17	8.81	8.84	11.08
	Nexus 7	N/A	56.32	54.49	69.61

[1] There is no version of Internet Explorer that runs on the Android OS.
[2] The signature is computed in two parallel background workers using the Chinese remainder theorem.

Εικόνα 18: Εκτέλεση RSA για δημιουργία και επαλήθευση ψηφιακής υπογραφής

TABLE II
RUNNING TIME OF SIGNATURE GENERATION OF RAINBOW (MS)

Device	IE	Chrome	Opera	Firefox
PC	11.80	3.67	3.74	4.70
Nexus 7	N/A	20.21	20.32	23.87

Εικόνα 19: Εκτέλεση RAINBOW για δημιουργία ψηφιακής υπογραφής

TABLE III
RUNNING TIME OF SIGNATURE VERIFICATION OF RAINBOW (MS)

Device	Num.[1]	IE	Chrome	Opera	Firefox
PC	1	12.37	4.18	4.20	6.22
	2	6.30	2.13	2.14	3.15
	3	4.38	1.50	1.51	2.23
	4	3.37	1.16	1.16	1.70
	5	3.10	1.04	1.05	1.55
	6	3.07	1.03	1.04	1.52
Nexus 7	1	N/A	20.90	21.34	32.80
	2	N/A	10.53	10.83	16.43
	3	N/A	7.43	7.51	11.47
	4	N/A	5.49	5.54	8.41

[1] This means the number of background workers.

Εικόνα 20: Εκτέλεση RSA για επαλήθευση ψηφιακής υπογραφής

Όπως φαίνεται και από τους πίνακες παραπάνω, παρατηρούμε πως οι παραπάνω υλοποιήσεις κρυπτοσυστημάτων δημοσίου κλειδιού, εφόσον χρησιμοποιηθούν σε συνδυασμό με την Τεχνολογία των Web Workers μπορούν να επαληθεύσουν την ταυτότητα μιας ψηφιακής υπογραφής, σε χρόνους που φτάνουν μόλις τα 1.03ms στην περίπτωση ενός προσωπικού υπολογιστή και τα 5.49ms στην περίπτωση ενός tablet. Επομένως αντίστοιχες υλοποιήσεις θα μπορούσαν να συνεισφέρουν σημαντικά στον τομέα της ασφάλειας επικοινωνιών.

3. Web Browsers Benchmarking

Στο πλαίσιο της μελέτης και ανάλυσης της απόδοσης των Web Workers κρίθηκε αναγκαίο να γίνουν κάποια συγκριτικά τεστ (benchmarks), χρησιμοποιώντας τρεις web εφαρμογές, προκειμένου να εξάγουμε κάποια επιπλέον συμπεράσματα για τρόπο με τον οποίο η παραπάνω τεχνολογία αξιοποιεί τα διαθέσιμα threads της κάθε συσκευής στην οποία εκτελείται. Στο κεφάλαιο αυτό επομένως θα παρουσιαστούν διαγράμματα με τους χρόνους εκτέλεσης από τα benchmarks που υλοποιήθηκαν με κάθε εφαρμογή, χρησιμοποιώντας διαφορετικά κάθε φορά σενάρια εκτέλεσης (π.χ. διαφορετικός αριθμός Web Workers, διαφορετικός browser, διαφορετική συσκευή κλπ.).

Η πρώτη εφαρμογή που χρησιμοποιήθηκε για την μελέτη της συμπεριφοράς αλλά και της απόδοσης των Web Workers είναι μια εφαρμογή η οποία προσπαθεί να υπολογίσει το αποτέλεσμα της πράξης a^b mod m χρησιμοποιώντας το Κινέζικο Θεώρημα του υπολοίπου (http://pmav.eu/stuff/javascript-webworkers/). Λόγω της πολυπλοκότητας της πράξης αποτελεί ένα καλό παράδειγμα για να δείξουμε το πόσο σημαντική μπορεί να είναι η χρήση των Web Workers, σε περιπτώσεις όπου υπάρχει ένα «βαρύ» υπολογιστικά script, το οποίο μπορεί να κατανεμηθεί σε πολλά διαφορετικά threads και έτσι να επιταχυνθεί σημαντικά η εκτέλεση του.

Η δεύτερη εφαρμογή που χρησιμοποιήθηκε στο πλαίσιο της μελέτης των Web Workers, δημιουργεί μια εικόνα και στην συνέχεια την σχεδιάζει σε ένα HTML5 Canvas Element. (https://nerget.com/rayjs-mt/rayjs.html). Στην εφαρμογή αυτή γίνεται επιλογή του επιθυμητού αριθμού web workers σύμφωνα με τον οποίο καθορίζεται ο αριθμός των οριζόντιων τμημάτων στα οποία θα τεμαχιστεί η εικόνα, ώστε να γίνουν ταυτόχρονα σε διαφορετικά threads οι υπολογισμοί που απαιτούνται για την σχεδίαση της εικόνας. Η

τελική σχεδίαση όμως, γίνεται από το κεντρικό thread καθώς οι Web Workers δεν έχουν πρόσβαση στο DOM και επομένως και στο canvas element, όπου και εμφανίζεται η εικόνα.

Τέλος, χρησιμοποιήθηκε και μια τρίτη εφαρμογή, η οποία μπορεί να υπολογίσει md5 hashes με χρήση της μεθόδου brute force (https://feross.org/hacks/md5-password-cracker.js/). Η εφαρμογή αυτή μπορεί να μοιράσει το σύνολο πιθανών συνδυασμών σε διαφορετικά threads-workers ανάλογα με τον αριθμό που επιλέγει ο χρήστης και επομένως μπορεί πετύχει αρκετά καλύτερες επιδόσεις.

Στην συνέχεια, παρουσιάζονται διαγράμματα με τις επιδόσεις των τριών παραπάνω εφαρμογών σε διαφορετικά σενάρια εκτέλεσης κάθε φορά. Πιο συγκεκριμένα γίνεται εκτέλεση:

- Σε τέσσερις διαφορετικούς φυλλομετρητές (Chrome, Firefox, Edge, Opera), προκειμένου να δούμε τις διαφορές που οφείλονται στις διαφορετικές Javascript Engines που αυτοί χρησιμοποιούν
- Σε τρία διαφορετικά συστήματα (2 laptop και ένα smartphone), ώστε να δούμε πως επηρεάζεται η απόδοση μιας εφαρμογής από την χρήση διαφορετικού hardware
- Σε παλαιότερες εκδόσεις των δύο πιο δημοφιλών φυλλομετρητών (Chrome και Firefox), ώστε να δούμε πόσο σημαντική είναι η εξέλιξη των επιδόσεων, καθώς εξελίσσονται οι Javascript Engines των web browsers αλλά και κατά πόσο αυτό τελικά επηρεάζει την απόδοση των επιπλέον threads που εκτελούνται από τους web workers
- Σε Mobile εκδόσεις των Chrome και Firefox (για την περίπτωση του smartphone), προκειμένου να δούμε αν το όφελος από την χρήση των Web Workers είναι εξίσου σημαντικό ή και μεγαλύτερο, για τις σύγχρονες έξυπνες συσκευές που χρησιμοποιούμε

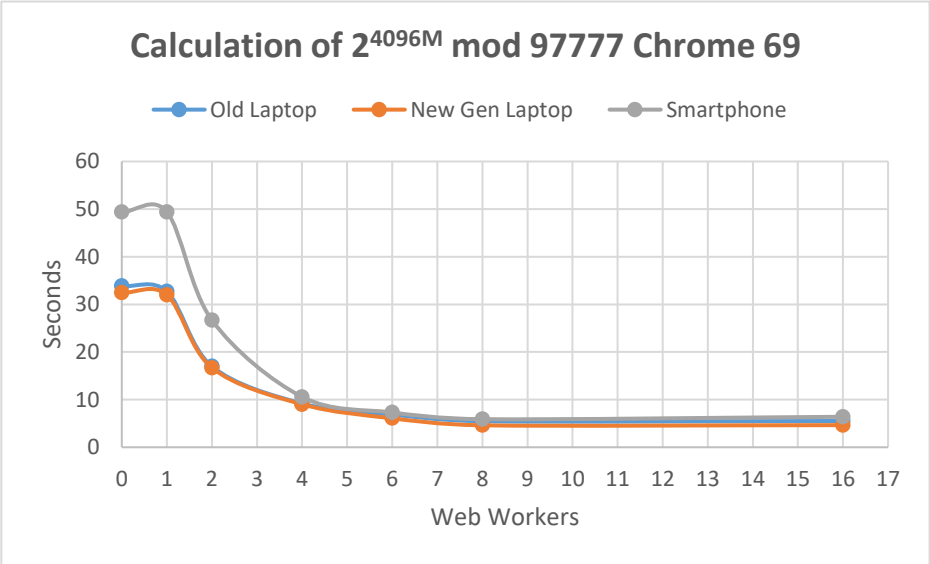
Ακόμη, είναι σημαντικό να αναφέρουμε τα χαρακτηριστικά των συστημάτων που χρησιμοποιούνται για την εκτέλεση των benchmarks, τα οποία παρατίθενται παρακάτω:

- Ένα laptop με επεξεργαστή i7 3630QM 2.4GHz (4-cores, 8-threads), μνήμη RAM 8GB DDR3 και λειτουργικό Windows 10, το οποίο θα αναφέρεται από εδώ και πέρα ως Old Laptop στα γραφήματα.

- Ένα ακόμη laptop με επεξεργαστή i7 8550U 1.8 GHz (4-cores, 8-threads) και μνήμη RAM 8GB DDR4 και λειτουργικό Windows 10, το οποίο θα αναφέρεται από εδώ και πέρα ως New Gen Laptop στα γραφήματα.
- Ένα smartphone με επεξεργαστή Snapdragon 625 2.0 GHz ARM Cortex A-53 (8-cores, 8-threads), μνήμη 3GB και λειτουργικό Android 7.0, το οποίο θα αναφέρεται από εδώ και πέρα ως Smartphone στα γραφήματα.

3.1 Χρήση των Web Workers για Πολύπλοκους Μαθηματικούς Υπολογισμούς

Στην ενότητα αυτή παρουσιάζεται η πρώτη εφαρμογή που χρησιμοποιήθηκε για τα συγκριτικά τεστ (benchmarks), και αφορά την χρήση web workers για την επίλυση ενός πολύπλοκου μαθηματικού υπολογισμού. Στα συγκριτικά τεστ που έγιναν χρησιμοποιήθηκαν πολλαπλοί web workers προκειμένου να υπολογιστεί το αποτέλεσμα της πράξης $2^{4096M} \bmod 97777$. Αρχικά γίνεται εκτέλεση χωρίς την χρήση web workers και στη συνέχεια αυξάνονται κατά έναν κάθε φορά ώστε να δούμε ακριβώς ποια είναι συμβολή του κάθε επιπλέον worker στην συνολική απόδοση της εφαρμογής. Παρακάτω παρουσιάζονται τα αποτελέσματα που προέκυψαν από την εκτέλεση της εφαρμογής στα τρία διαθέσιμα συστήματα χρησιμοποιώντας τέσσερις δημοφιλείς web browsers (στα διαγράμματα που συμμετέχει η συσκευή smartphone γίνεται χρήση των Chrome For Android 69 και Firefox For Android 62 αντίστοιχα).



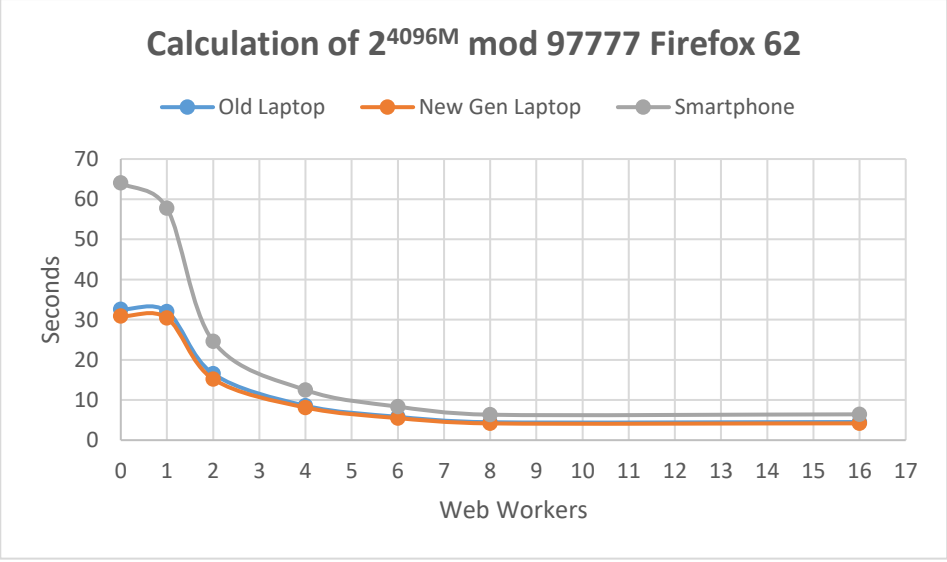
Εικόνα 21: Υπολογισμός $2^{4096M} \bmod 97777$ Chrome v69

Στο διάγραμμα της εικόνας 21, παρατηρούμε αρχικά ότι οι επιδόσεις των τριών συσκευών ακολουθούν την ίδια σχεδόν πορεία, ειδικά από την χρήση 2 workers και μετά η καμπύλη απόδοσης τους σχεδόν ταυτίζεται. Το γεγονός αυτό είναι πολύ σημαντικό για την περίπτωση του smartphone, καθώς ενώ αρχικά υπολείπεται αρκετά σε επιδόσεις σε σχέση με τα δύο laptop, στην συνέχεια όταν χρησιμοποιεί 8 threads φτάνει σχεδόν στα ίδια επίπεδα απόδοσης. Συγκεκριμένα, στους 8 workers οι χρόνοι εκτέλεσης των τριών συστημάτων είναι οι παρακάτω:

- Laptop νέας γενιάς (i7-8550u): 4.61 δευτερόλεπτα
- Laptop παλαιότερης γενιάς (i7-3630QM): 5.47 δευτερόλεπτα
- Smartphone (Snapdragon 625): 5.9 δευτερόλεπτα

Επίσης είναι σημαντικό να αναφέρουμε ότι ακόμη και όταν εκτελούνται 16 web workers, κανένα από τα τρία συστήματα δεν φαίνεται να παρουσιάζει κάποια ουσιαστική μείωση στις επιδόσεις παρόλο που και τα τρία συστήματα διαθέτουν 8 λογικούς πυρήνες (λιγότερους δηλαδή από τον αριθμό των ενεργών threads).

Ακόμη είναι αξιοσημείωτο το γεγονός ότι ήδη από την χρήση των τεσσάρων από τους 8 διαθέσιμους λογικούς πυρήνες των τριών συστημάτων παρουσιάζεται μια σημαντική αύξηση της απόδοσης της τάξης του 72,87% για το παλαιότερης γενιάς laptop, του 70% για το laptop νέας γενιάς και του 78,4% για το smartphone. Ενώ όταν χρησιμοποιούνται 8 web workers, όπου επιτυγχάνεται η μέγιστη δυνατή απόδοση και στα τρία συστήματα, η αύξηση της απόδοσης φτάνει το 83,8% για το laptop παλαιότερης γενιάς, το 85,7% για το laptop νέας γενιάς και 83,7% για το smartphone.

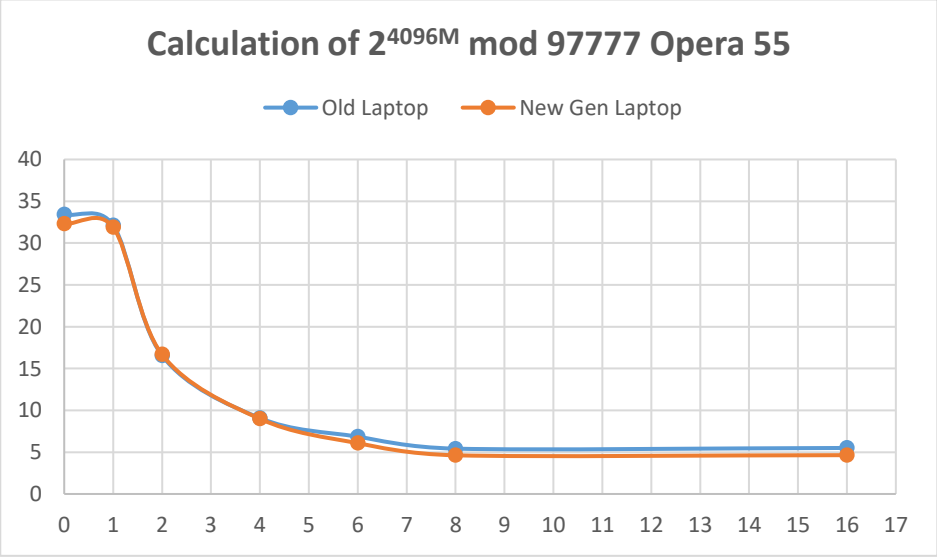


Εικόνα 22: Υπολογισμός 2^{4096M} mod 97777 Firefox v62

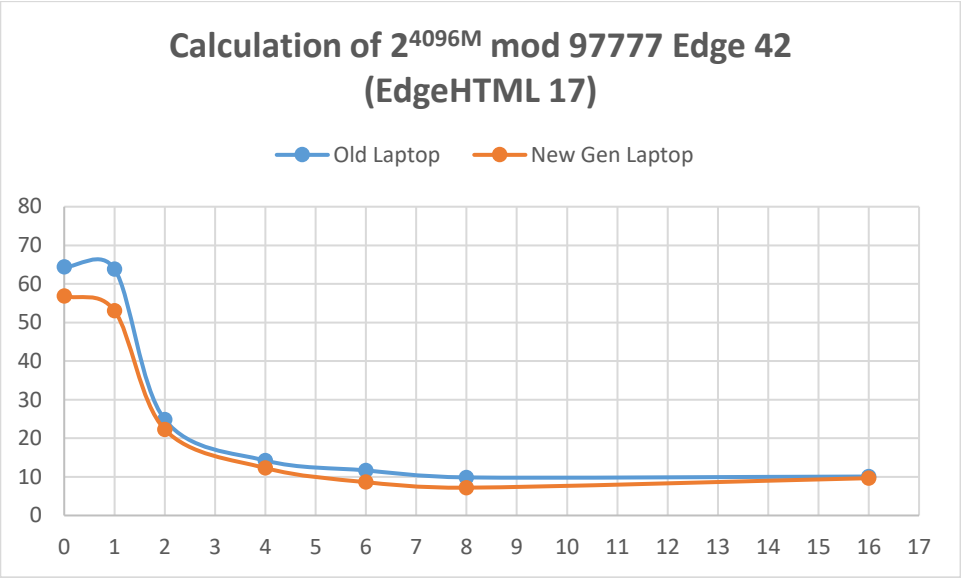
Συνεχίζοντας, παρατηρούμε ότι παρόμοια αποτελέσματα παίρνουμε και στον Firefox (Εικόνα 22), καθώς η καμπύλη απόδοσης των τριών συστημάτων έχει παρόμοια συμπεριφορά με αυτή του Chrome, με κύρια διαφορά μεταξύ των δύο φυλλομετρητών να είναι η απόδοση του smartphone (η σύγκριση επομένως αφορά τις mobile εκδόσεις των browsers) όταν γίνεται χρήση μηδέν ή ενός web worker. Σε αυτή την περίπτωση βλέπουμε τον χρόνο εκτέλεσης να φτάνει τα 64 δευτερόλεπτα με χρήση 0 workers και 57,7 όταν γίνεται χρήση μόνο ενός worker όταν οι αντίστοιχες τιμές στον Chrome είναι 49,37 με μηδέν workers και 49,4 με έναν worker.

Κοιτώντας και τα γραφήματα των Opera και Microsoft Edge, παρατηρούμε και πάλι παρόμοια συμπεριφορά στην καμπύλη απόδοσης, με αυτή των Chrome και Firefox. Μια σημαντική παρατήρηση είναι ότι ο Microsoft Edge είναι με διαφορά ο χειρότερος στις περιπτώσεις όπου γίνεται εκτέλεση σε 0 ή 1 worker με χρόνους σχεδόν διπλάσιους από τους υπόλοιπους browsers, οι οποίοι παρουσιάζουν σχεδόν ίσες μεταξύ τους επιδόσεις, στον ίδιο αριθμό workers.

Όσον αφορά το σημείο όπου επιτυγχάνονται οι μεγαλύτερες επιδόσεις, το οποίο είναι κοινό για όλους του browsers, είναι όταν εκτελούνται παράλληλα 8 workers. Σε αυτό το σημείο συγκρίνοντας τις επιδόσεις των τεσσάρων browsers με βάση τα αποτελέσματα που προέκυψαν, μπορούμε να πούμε ότι παρουσιάζουν πολύ μικρές διαφορές μεταξύ τους (περίπου 1 sec) εκτός από τον Microsoft Edge ο οποίος παρουσιάζει μειωμένες επιδόσεις σε σχέση με τους υπόλοιπους, σε ποσοστό από 44% έως και 55%.



Εικόνα 23: Υπολογισμός 2^{4096M} mod 97777 Opera v55



Εικόνα 24: Υπολογισμός 2^{4096M} mod 97777 Microsoft Edge v42 (EdgeHTML 17)

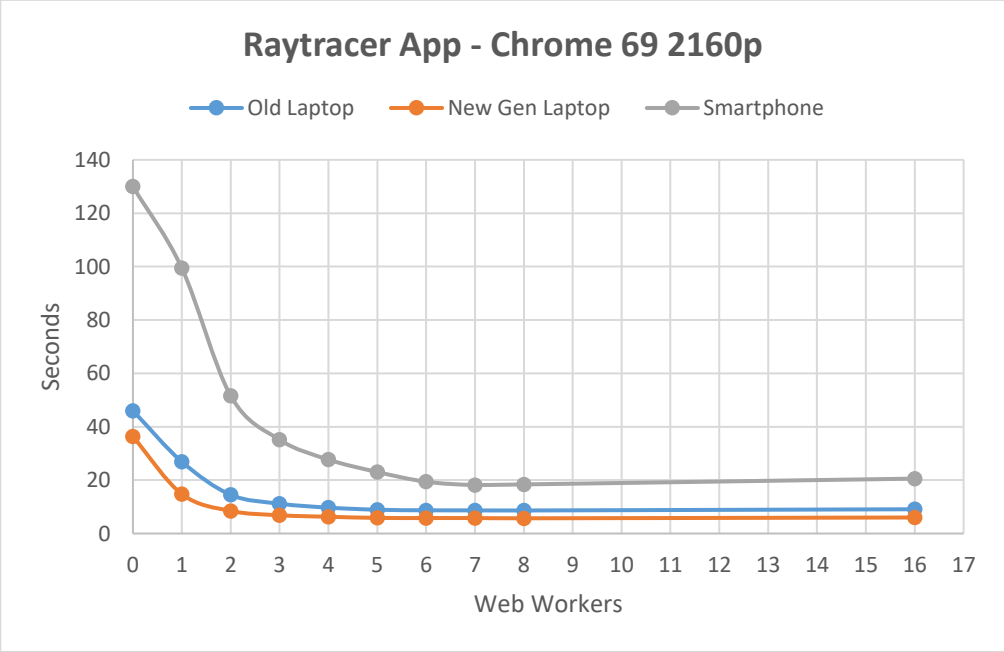
Συνοψίζοντας, έχοντας λάβει υπόψη τα αποτελέσματα από τα διαγράμματα και των τεσσάρων φυλλομετρητών καταλήγουμε στα εξής συμπεράσματα:

- Η απόδοση της εφαρμογής και στους τέσσερις browsers φαίνεται να έχει την ίδια καμπύλη εξέλιξης, καθώς αυξάνεται ο αριθμός των web workers
- Και οι τέσσερις browsers επιτυγχάνουν την μέγιστη απόδοση στους 8 workers, με τις διαφορές στις μεταξύ τους επιδόσεις να μην είναι τόσο μεγάλες ώστε να ξεχωρίσει κάποιος browser ως προς την ταχύτητα (περίπου 1 δευτερόλεπτο διαφορά μεταξύ των τριών συσκευών με ταχύτερη συσκευή το laptop νέας γενιάς)
- Οι περισσότεροι browsers δείχνουν να μην επηρεάζονται αρνητικά από την χρήση περισσότερων web workers από τους διαθέσιμους λογικούς πυρήνες με εξαίρεση τον Microsoft Edge ο οποίος παρουσιάζει μια μείωση της απόδοσης
- Οι mobile εκδόσεις των Chrome και Firefox που εκτελούνται στο smartphone, ενώ στην αρχή υστερούν σε επιδόσεις, στη συνέχεια καθώς πλησιάζουν στην εκτέλεση του βέλτιστου αριθμού workers (8 workers) σχεδόν φτάνουν τις επιδόσεις των Desktop εκδόσεων τους, που εκτελούνται στα δύο laptop
- Ο μόνος browser που φαίνεται να υστερεί κάπως σε σχέση με τους υπόλοιπους είναι ο Microsoft Edge

3.2 Αξιοποίηση των Web Workers για Σχεδίαση σε HTML5 Canvas Element

Σε αυτή την ενότητα παρουσιάζεται η δεύτερη εφαρμογή που χρησιμοποιήθηκε για τα benchmarks η οποία αφορά την σχεδίαση (rendering) μιας εικόνας σε ένα element canvas, με την χρήση πολλαπλών web workers. Στα συγκριτικά τεστ που έγιναν χρησιμοποιήθηκαν πολλαπλοί web workers προκειμένου να γίνει ο απαραίτητος υπολογισμός των δεδομένων (pixels) μιας εικόνας ανάλυσης 4K(3840x2160), χωρίζοντας κάθε φορά την εικόνα σε οριζόντια τμήματα ανάλογα με τον αριθμό των threads (workers).

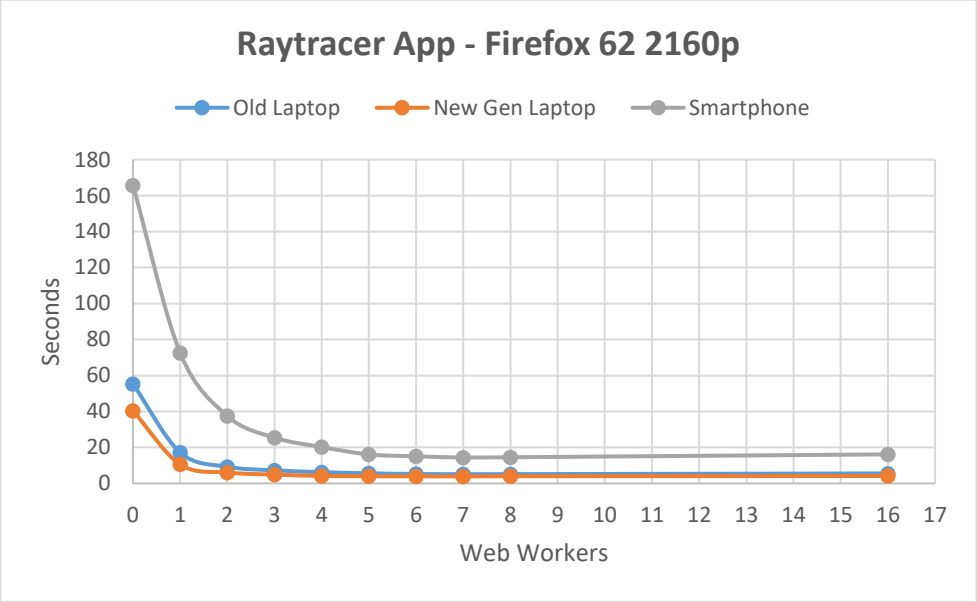
Αρχικά παρουσιάζονται τα αποτελέσματα που προέκυψαν από την εκτέλεση της εφαρμογής σε τρία διαφορετικά συστήματα, χρησιμοποιώντας τέσσερις δημοφιλείς web browsers (στα διαγράμματα που συμμετέχει η συσκευή smartphone γίνεται χρήση των Chrome For Android 69 και Firefox For Android 62 αντίστοιχα). Στη συνέχεια έγινε εκτέλεση του ίδιου benchmark με χρήση διαφορετικών εκδόσεων του ίδιου φυλλομετρητή (Διαφορετικές εκδόσεις των Chrome και Firefox) προκειμένου να παρουσιαστούν η διαφορές που προκύπτουν στην απόδοση, από την εξέλιξη των Javascript Engines που χρησιμοποιούν οι browsers.



Εικόνα 25: Σχεδίαση Εικόνας 2160p Chrome v69

Βλέποντας αρχικά τα διαγράμματα που αφορούν τον Chrome (Εικόνα 25) και τον Firefox (Εικόνα 26), το πρώτο που παρατηρούμε είναι η μεγάλη διαφορά μεταξύ των δύο laptop και του smartphone, όταν γίνεται χρήση 0 ή 1 web worker. Αυτό οφείλεται στο γεγονός ότι το κεντρικό thread σε αυτή την εφαρμογή συνεχίζει να εκτελεί ένα σημαντικό κομμάτι της όλης διαδικασίας για την σχεδίαση της εικόνας. Το κεντρικό thread θα πρέπει κάθε φορά που παραλαμβάνει ένα τμήμα της συνολικής εικόνας από έναν web worker (ο οποίος υπολογίζει τις τιμές των pixel για το τμήμα που του έχει ανατεθεί) να σχεδιάζει το συγκεκριμένο τμήμα πάνω στο canvas element ώστε τελικά να σχεδιαστεί τμήμα-τμήμα η συνολική εικόνα.

Όπως γίνεται λοιπόν αντιληπτό, όλη η παραπάνω διαδικασία από την πλευρά του κεντρικού thread έχει σημαντικό αντίκτυπο στον συνολικό χρόνο εκτέλεσης καθώς η single-thread επίδοση του κάθε επεξεργαστή παίζει σημαντικό ρόλο, παρόλο που γίνεται χρήση πολλαπλών threads.



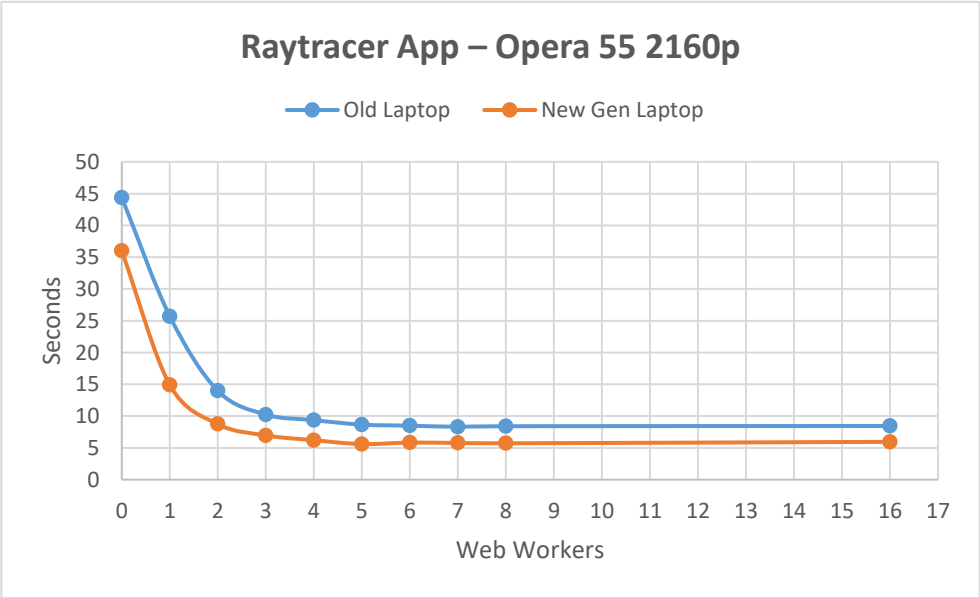
Εικόνα 26: Σχεδίαση Εικόνας 2160p Firefox v62

Στη συνέχεια, παρατηρώντας και τα υπόλοιπα διαγράμματα βλέπουμε ότι η βέλτιστη απόδοση επιτυγχάνεται στους 7-8 workers και ότι οι επιπλέον workers δεν συμβάλουν σημαντικά στις συνολική επίδοση της εφαρμογής, καθώς μετά από αυτό το σημείο ο παράγοντας που καθυστερεί την εφαρμογή είναι η χρήση του κεντρικού thread για την σχεδίαση πάνω στον καμβά. Παρ' όλα αυτά και παρατηρούμε σημαντική αύξηση στην απόδοση και τον τριών συστημάτων.

Πιο συγκεκριμένα, το ποσοστό αύξησης της απόδοσης των τριών συσκευών για τον Chrome, είναι 81.2% για το laptop παλαιότερης γενιάς (με χρήση 8 workers), 84.2% για το laptop νέας γενιάς (με χρήση 8 workers) και 86% για το smartphone (με χρήση 7 workers). Βέβαια την καλύτερη επίδοση όπως ήταν αναμενόμενο, έχει το laptop νέας γενιάς, το οποίο παρουσιάζει μια διαφορά 33.4% από το laptop παλαιότερης γενιάς και 68.6% από το smartphone.

Συνεχίζοντας με τον Firefox (Εικόνα 26), το διάγραμμα παρουσιάζει αντίστοιχα χαρακτηριστικά με αυτό του Chrome, με κύριες διαφορές, ότι η απόσταση μεταξύ των επιδόσεων των δύο laptop έχει μειωθεί και ότι γενικότερα τα τρία συστήματα παρουσιάζουν βελτιωμένους χρόνους εκτέλεσης σε σχέση με τους αντίστοιχους χρόνους στον Chrome (εκτός από την περίπτωση όπου δεν χρησιμοποιείται κάποιος web worker που παρουσιάζει χειρότερη απόδοση). Συγκεκριμένα, παρόλο που για άλλη μια φορά το laptop νέας γενιάς έχει την ταχύτερη επίδοση σε σχέση με τα άλλα δύο συστήματα, η διαφορά του με την επίδοση του laptop παλαιότερης γενιάς έχει μειωθεί σε 22.3% από 33.4%.

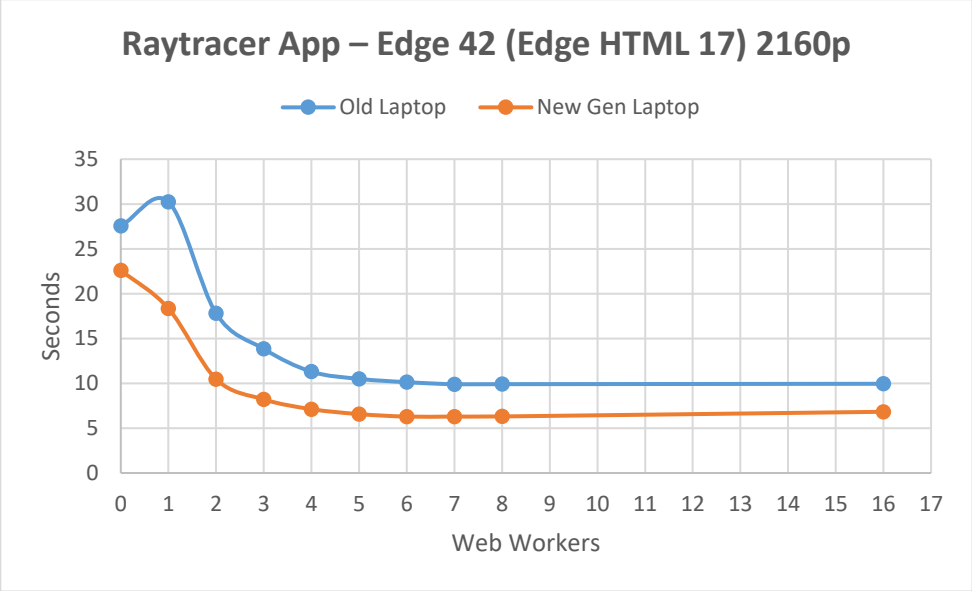
Όσον αφορά την βελτίωση στους χρόνους εκτέλεσης των τριών συστημάτων σε σχέση με τους χρόνους στον Chrome, αποτυπώνεται σε 29.8% για το laptop νέας γενιάς, 40.3% για το παλαιότερης γενιάς laptop και 15,5% για το smartphone.



Εικόνα 27: Σχεδίαση Εικόνας 2160p Opera v55

Προχωρώντας, μετά από μελέτη των διαγραμμάτων των Opera και Microsoft Edge (Εικόνες 27, 28), βλέπουμε ότι ο Opera browser έχει σχεδόν πανομοιότυπη συμπεριφορά με τον Chrome πράγμα το οποίο βέβαια δεν είναι παράλογο μιας και χρησιμοποιούν την ίδια Javascript Engine (V8).

Επίσης, βλέπουμε ότι ο Microsoft Edge για άλλη μια φορά φαίνεται να μένει πίσω (το ίδιο συνέβη και με την πρώτη εφαρμογή, στην προηγούμενη ενότητα) αλλά αυτή την φορά όχι με τόσο μεγάλη διαφορά. Συγκεκριμένα με 8 web workers σημείωσε χρόνο εκτέλεσης 6.31 δευτερόλεπτα στο laptop νέας γενιάς και 9.9 δευτερόλεπτα στο παλαιότερης γενιάς laptop, όταν οι αντίστοιχες τιμές είναι 5.7 και 8.63 δευτερόλεπτα στον Chrome, 4 και 5.15 δευτερόλεπτα στον Firefox και τέλος, 5.72 και 8.41 δευτερόλεπτα στον Opera. Επομένως παρουσιάζει μειωμένες επιδόσεις κατά μέσο όρο 11.6% σε σύγκριση με τον Chrome, 43.5% σε σχέση με τον Firefox και 12.8% σε σχέση με τον Opera.



Εικόνα 28: Σχεδίαση Εικόνας 2160p Microsoft Edge v42 (EdgeHTML 17)

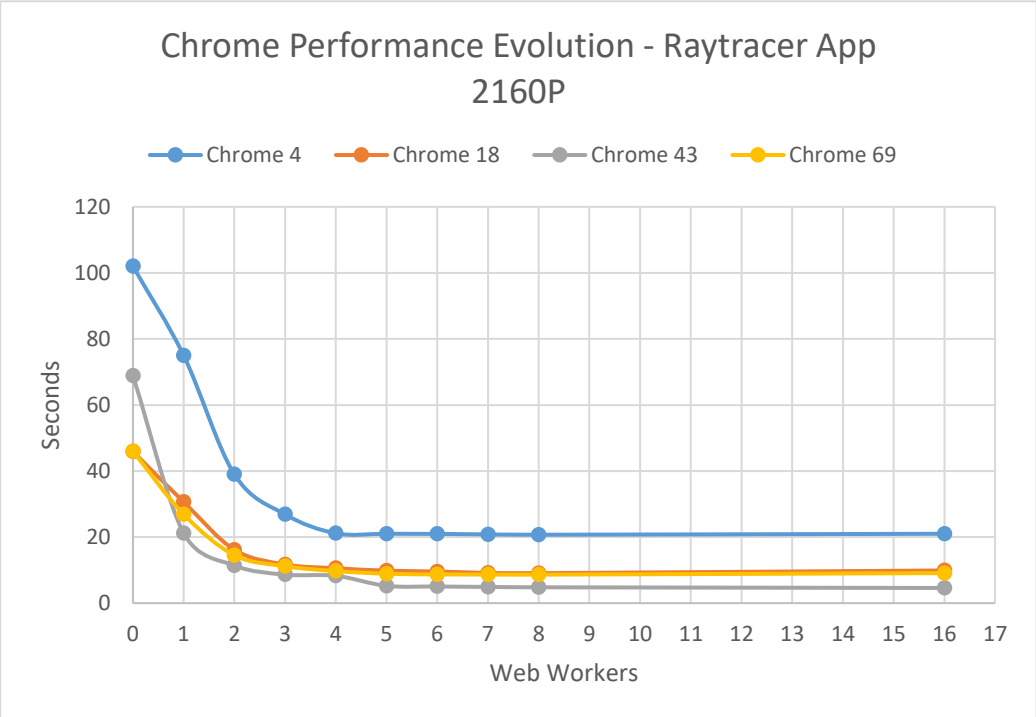
Συνοψίζοντας τα παραπάνω αποτελέσματα, καταλήγουμε στα εξής συμπεράσματα:

- Στην συγκεκριμένη εφαρμογή παρατηρούμε ότι τις καλύτερες επιδόσεις έχει ο Firefox ο οποίος επιτυγχάνει βελτίωση απόδοσης (σε σχέση με την εκτέλεση χωρίς web workers) σε ποσοστό που φτάνει το 90% και για τα τρία συστήματα .
- Όλοι οι browsers φτάνουν στο σημείο μέγιστης απόδοσης στους 8 workers και η προσθήκη επιπλέον workers οδηγεί σε μικρή μείωση των επιδόσεων.

- Παρατηρούμε ότι ο Opera έχει και πάλι (όπως και στην πρώτη εφαρμογή πιο πάνω) παρόμοιες επιδόσεις με τον Chrome, γεγονός που δικαιολογείται από την χρήση της ίδιας Javascript Engine.
- Ο Microsoft Edge συνεχίζει να παραμένει τελευταίος (όπως συνέβη και στην πρώτη εφαρμογή) έστω και με μικρή σχετικά διαφορά.

Συνεχίζοντας, προκειμένου να δούμε κατά πόσο η εξέλιξη των φυλλομετρητών έχει συνεισφέρει στην βελτίωση της απόδοσης γενικά (single-thread) αλλά και ειδικά στην απόδοση των επιπλέον threads (multithreading), εκτελέστηκε η παραπάνω εφαρμογή σε παλαιότερες εκδόσεις των Chrome (Εικόνα 29) και Firefox (Εικόνα 30), ξεκινώντας από τις εκδόσεις 4 και 3.5 αντίστοιχα, οι οποίες ήταν και οι πρώτες εκδόσεις που υποστήριζαν το Web Workers API. Το σύστημα που χρησιμοποιήθηκε στα παρακάτω συγκριτικά τεστ είναι το laptop παλαιότερης γενιάς (i7 3630QM 4-cores, 8-threads) του οποίου τα χαρακτηριστικά έχουν αναφερθεί παραπάνω.

Παρακάτω βλέπετε το διάγραμμα όπου παρουσιάζεται η διαφορά στην απόδοση ανάλογα με την έκδοση του Chrome που χρησιμοποιήθηκε.

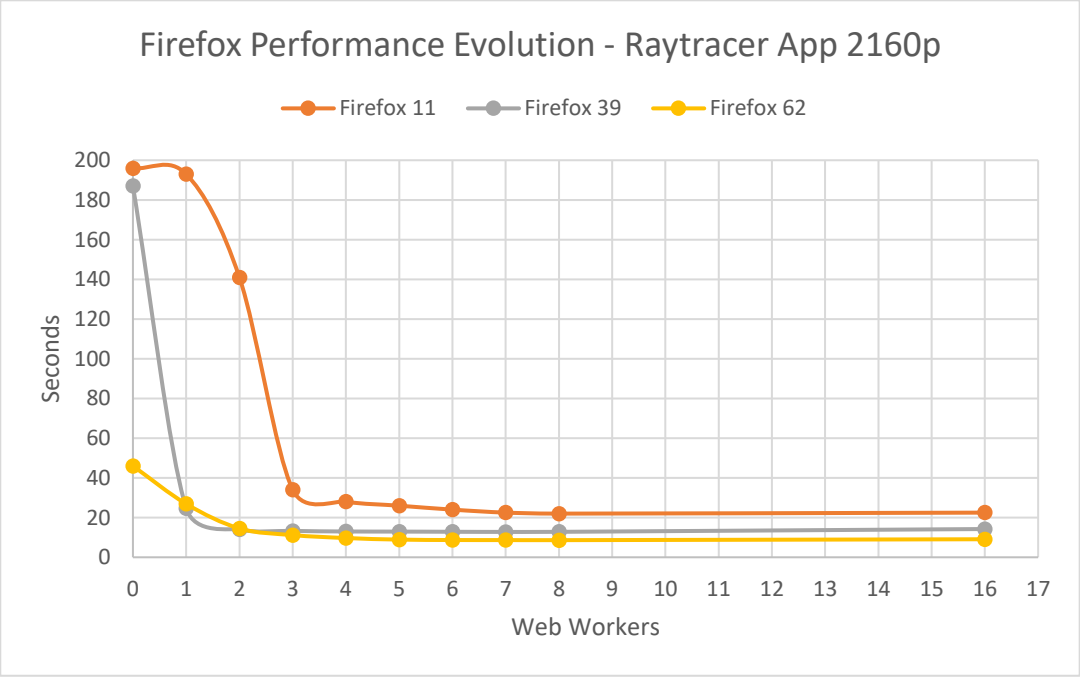


Εικόνα 29: Εξέλιξη Απόδοσης ανά Έκδοση για τον Chrome

Η πρώτη έκδοση του Chrome (Version 4) όπως είναι λογικό, παρουσιάζει σημαντικά μειωμένες επιδόσεις σε σχέση με τις νεότερες εκδόσεις του browser, με τα ποσοστά να φτάνουν το 55%, χωρίς την χρήση web workers και το 58.3% με την χρήση 8 web workers (σε σύγκριση με την πιο πρόσφατη έκδοση 69). Όσον αφορά την έκδοση 18, ο Chrome πετυχαίνει επιδόσεις που είναι πολύ κοντά με αυτές της έκδοσης 69, γεγονός που αποτελεί ευχάριστη έκπληξη.

Συνεχίζοντας, στην έκδοση 43 βλέπετε τις επιδόσεις του browser να ξεπερνούν με αισθητή διαφορά αυτές της αρκετά νεότερης έκδοσης 69. Ενώ στην περίπτωση εκτέλεσης χωρίς web workers οι επιδόσεις είναι μειωμένες κατά 33.1%, με την χρήση των πολλαπλών threads καταφέρνει να πετύχει επιδόσεις καλύτερες κατά 44.3% από την έκδοση 69.

Προχωρώντας, στο διάγραμμα παρακάτω (Εικόνα 30), μπορείτε να δείτε τα αποτελέσματα από την εκτέλεση της εφαρμογής σε διαφορετικές εκδόσεις του Firefox. Όπως θα παρατηρήσετε η έκδοση 3.5 η οποία αναφέρθηκε και πιο πάνω ως πρώτη έκδοση που υποστηρίζει τους web workers, λείπει από το διάγραμμα. Ο λόγος γι' αυτό είναι ότι παρόλο που φαίνεται να υποστηρίζει το Web Workers API, καταφέραμε να εκτελέσουμε μόνο έως 4 workers ταυτόχρονα, πριν αρχίσει να παγώνει η εφαρμογή και να μην ολοκληρώνει ποτέ την εκτέλεση της. Αξίζει να αναφέρουμε όμως ότι στις περιπτώσεις όπου ολοκληρώθηκε κανονικά η εκτέλεση της εφαρμογής, οι χρόνοι για την ολοκλήρωση της εκτέλεσης ήταν αρκετά μεγαλύτεροι σε σχέση με τις μεταγενέστερες εκδόσεις που δοκιμάσαμε.



Εικόνα 30: Εξέλιξη Απόδοσης ανά Έκδοση για τον Firefox

Συγκρίνοντας λοιπόν τις εκδόσεις 11, 39 και 62 του Firefox καταλήγουμε στα παρακάτω συμπεράσματα:

- Στην έκδοση 11 παρατηρούμε ότι ήδη από την χρήση τριών Web Workers η απόδοση βελτιώνεται κατά 82.6% και τελικά φτάνει στο βέλτιστη τιμή της στους 8 workers, χωρίς να υπάρχει μεγάλη διαφορά μεταξύ των σημείων αυτών.
- Αντίστοιχα στην έκδοση 39, με την χρήση μόλις ενός web worker γίνεται ένα άλμα στην απόδοση της εφαρμογής, της τάξης του 86,8%. Στην συνέχεια γίνεται μια ακόμη σημαντική αύξηση της απόδοσης με την χρήση ενός ακόμη worker και από εκεί και πέρα παρατηρούνται πολύ μικρές αυξήσεις μέχρι και τους 8 workers όπου έχουμε την βέλτιστη τιμή.
- Τέλος, βλέπουμε ότι η έκδοση 39 απέχει ελάχιστα από τις επιδόσεις της έκδοσης 62, εκτός από την περίπτωση όπου δεν γίνεται χρήση των Web Workers στην οποία παρουσιάζει επιδόσεις χειρότερες κατά 75.4%.

Κλείνοντας, καταλήγουμε στο συμπέρασμα ότι καθώς εξελίσσονται οι browsers, τείνει να βελτιώνεται και η απόδοση των web workers, απλά αυτό δεν αποτελεί απόλυτο κανόνα, καθώς όπως φάνηκε από τα παραπάνω διαγράμματα στην περίπτωση του Chrome είδαμε μια παλαιότερη έκδοση (Chrome 43) να εμφανίζει καλύτερες επιδόσεις από μια αρκετά νεότερη (Chrome 69).

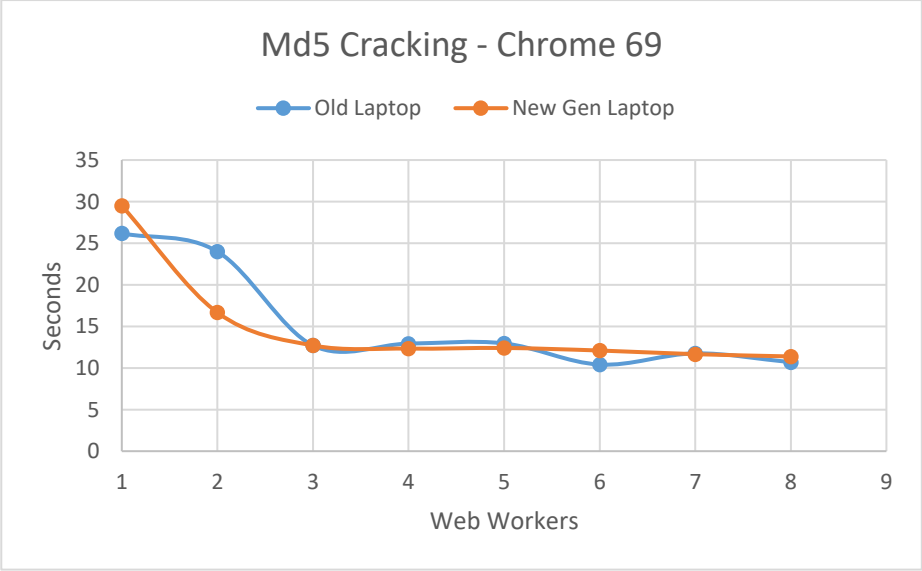
3.3 Χρήση των Web Workers για τον Υπολογισμό MD5 Hash

Σε αυτή την ενότητα παρουσιάζεται η τρίτη εφαρμογή που χρησιμοποιήθηκε για τα συγκριτικά τεστ και που αφορά την χρήση web workers για την υπολογισμό του αποτελέσματος μιας συνάρτησης κατακερματισμού τύπου MD5. Στα συγκεκριμένα τεστ γίνεται χρήση πολλαπλών web workers προκειμένου να υπολογιστεί η λέξη που αντιστοιχεί στο hash «54d75975e615f0638b6181592a4d929f» (λέξη “heya”), δοκιμάζοντας όλους του πιθανούς συνδυασμούς. Όπως και στα προηγούμενα benchmarks χρησιμοποιήθηκαν και εδώ τέσσερις δημοφιλείς φυλλομετρητές και τρία διαφορετικά συστήματα.

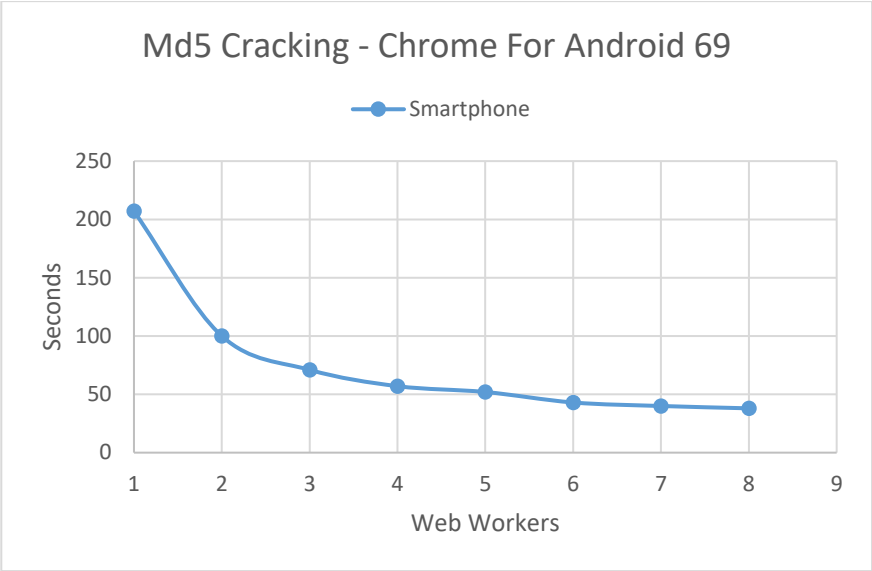
Στην συνέχεια παρατίθενται τα σχετικά διαγράμματα όπου και παρουσιάζονται οι επιδόσεις των προαναφερθέντων συστημάτων στους τέσσερις φυλλομετρητές.

Στο πρώτο διάγραμμα (Εικόνα 31), βλέπετε τα δύο laptop που χρησιμοποιήθηκαν καθώς και τις αντίστοιχες καμπύλες που υποδηλώνουν τους χρόνους εκτέλεσης που χρειάστηκαν για τον υπολογισμό του ζητούμενου Md5 hash, με βάση τον αριθμό των web workers που έκαναν χρήση. Ενώ στο δεύτερο διάγραμμα (Εικόνα 32), βλέπετε τους χρόνους εκτέλεσης για το smartphone.

Μελετώντας τα δύο διαγράμματα, παρατηρούμε ότι όπως και στις προηγούμενες εφαρμογές, η βέλτιστη απόδοση επιτυγχάνεται στους 8 web workers. Επίσης, βλέπουμε ότι οι επιδόσεις των δύο laptop είναι σχεδόν πάντα πολύ κοντά μεταξύ τους με μικρές γενικά αποκλίσεις. Όσον αφορά το smartphone παρατηρούμε ότι μέχρι να φτάσει στους 8 workers παρουσιάζει σημαντική βελτίωση της απόδοσης του. Συγκρίνοντας τις επιδόσεις των τριών συστημάτων μεταξύ του ενός και οκτώ workers, βλέπουμε μια αύξηση κατά 59.1% για το laptop παλαιότερης γενιάς, 61.3% για το νέας γενιάς laptop και 81.6% για το smartphone. Επομένως το όφελος από την χρήση πολλαπλών web workers είναι πολύ πιο σημαντικό για την συσκευή smartphone.



Εικόνα 31: Υπολογισμός MD5 Hash με χρήση του Chrome

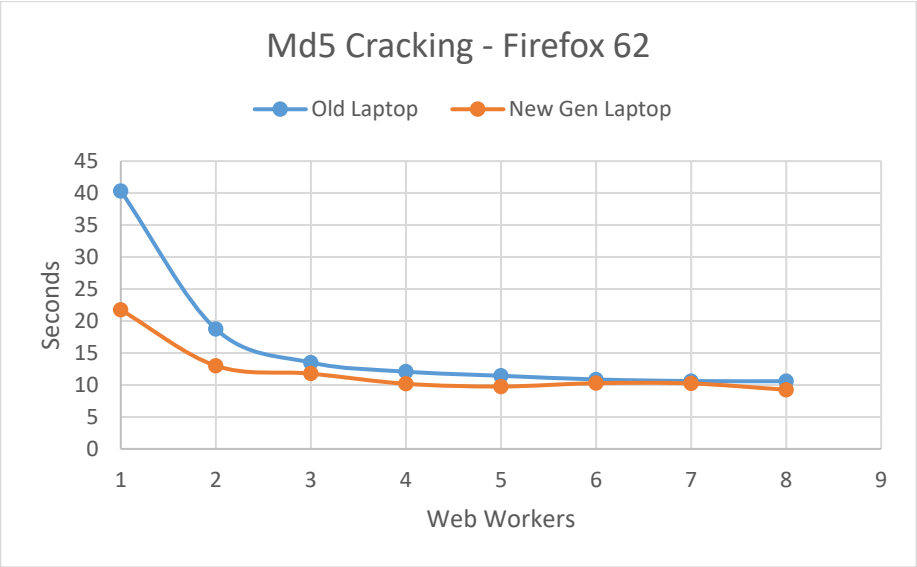


Εικόνα 32: Υπολογισμός MD5 Hash με χρήση του Chrome for Android

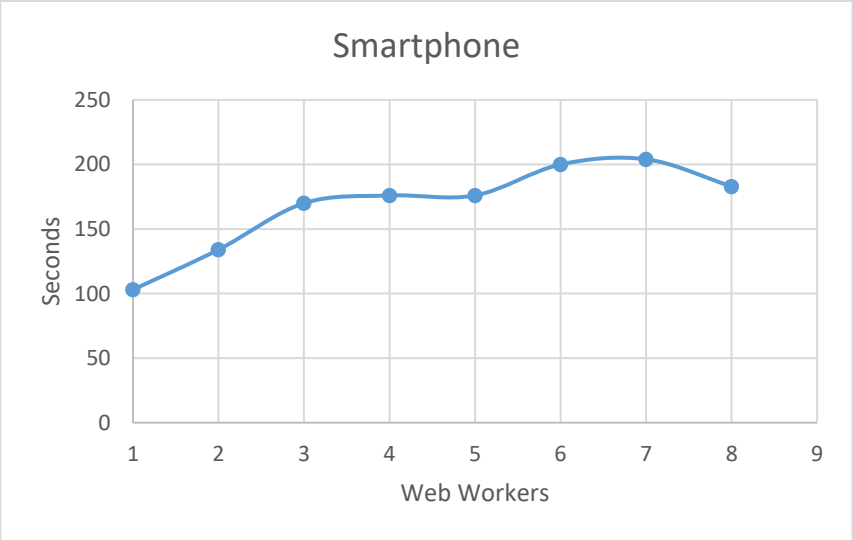
Συνεχίζοντας με τον Firefox, στα σχετικά διαγράμματα (Εικόνες 33 και 34), παρατηρούμε αρχικά το laptop νέας γενιάς να αποκτά στην μια σημαντική διαφορά από το παλαιότερης γενιάς laptop, το οποίο στη συνέχεια όμως μειώνει την διαφορά αυτή και φτάνει πολύ κοντά στην επίδοση του πρώτου (10.58 δευτερόλεπτα αντί 9.25).

Ακόμη, παρατηρούμε ότι το smartphone στην συγκεκριμένη εφαρμογή δείχνει να μην μπορεί να επωφεληθεί από την χρήση των web workers, καθώς ενώ αρχικά με την χρήση ενός worker παρουσιάζει διπλάσια επίδοση από την αντίστοιχη στον Chrome (103 δευτερόλεπτα αντί 207), στην συνέχεια παρουσιάζει μια μείωση στις επιδόσεις, αποτέλεσμα που θα πρέπει να διερευνηθεί περισσότερο για να βγάλουμε κάποιο συμπέρασμα.

Τέλος, αν εξαιρέσουμε το συγκεκριμένο θέμα που μας προβλημάτισε, παρατηρούμε ότι οι επιδόσεις στον Firefox για το laptop παλαιότερης γενιάς είναι σχεδόν ίδιες με πολύ μικρές διαφοροποιήσεις ενώ στην περίπτωση του laptop νέας γενιάς βλέπουμε ότι οι επιδόσεις στον Firefox παρουσιάζουν μια βελτίωση της τάξης του 18.7% (με χρήση 8 workers).

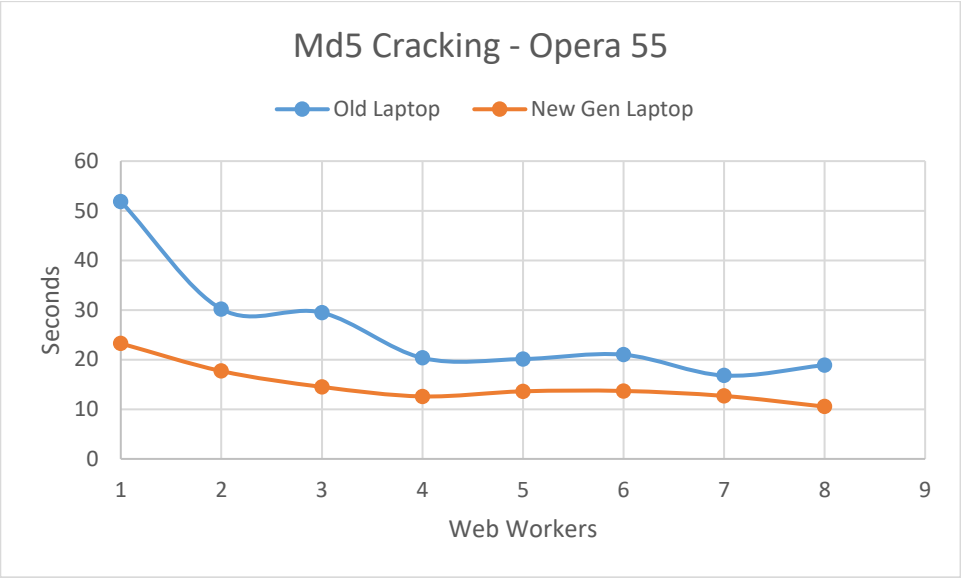


Εικόνα 33: Υπολογισμός MD5 Hash με χρήση του Firefox

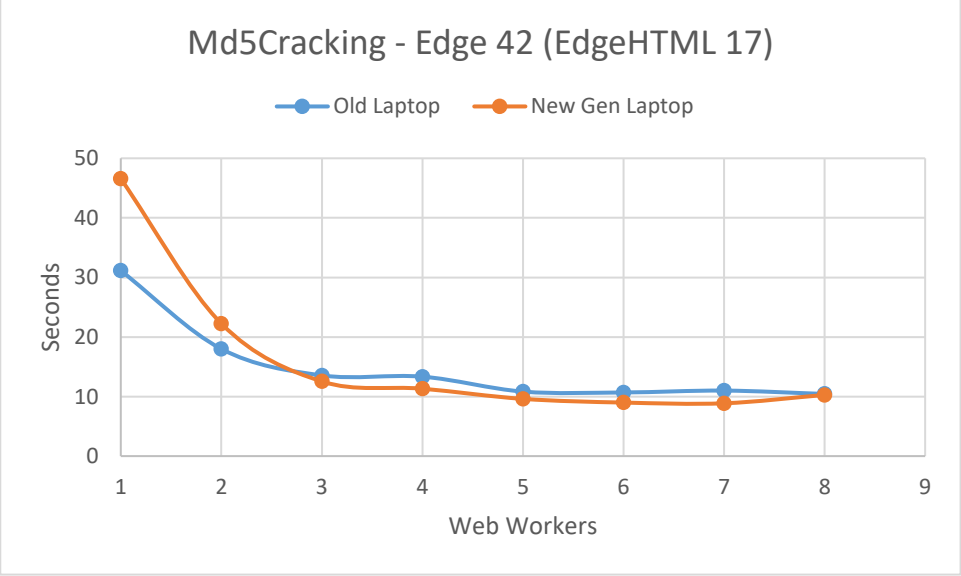


Εικόνα 34: Υπολογισμός MD5 Hash με χρήση του Firefox for Android

Προχωρώντας με τα διαγράμματα των Opera και Microsoft Edge (Εικόνες 35 και 36), παρατηρούμε ότι στον πρώτο οι διαφορά στις επιδόσεις μεταξύ των δύο laptop είναι σημαντικές, καθώς το laptop νέας γενιάς παρουσιάζει κατά μέσο όρο 43.17% καλύτερες επιδόσεις, διαφορά που δεν υπάρχει στους υπόλοιπους τρεις browsers, οι οποίοι παρουσιάζουν παρόμοιες επιδόσεις για τα δύο laptop. Τέλος, συγκρίνοντας τα αποτελέσματα από όλους του browsers που εξετάσαμε ότι ο φυλλομετρητής με την καλύτερη επίδοση είναι ο Microsoft Edge, ο οποίος επιτυγχάνει χρόνο εκτέλεσης 8.87 δευτερόλεπτα όταν οι αντίστοιχοι χρόνοι στους υπόλοιπους browsers είναι 11.39 στον Chrome, 9.25 στον Firefox, και 10.54 δευτερόλεπτα στον Opera.



Εικόνα 35: Υπολογισμός MD5 Hash με χρήση του Opera



Εικόνα 36: Υπολογισμός MD5 Hash με χρήση του Microsoft Edge

4. Επίλογος

4.1 Σύνοψη και Συμπεράσματα

Έχοντας ολοκληρώσει την μελέτη της σχετικής βιβλιογραφίας αλλά και την εκτέλεση των παραπάνω συγκριτικών τεστ (benchmarks) μπορούμε να πούμε ότι έχουμε καταλήξει σε αρκετά συμπεράσματα όσον αφορά το API των Web Workers και τον τρόπο με τον οποίο μπορούν να συνεισφέρουν στις σύγχρονες web εφαρμογές. Παρακάτω συνοψίζονται αυτά τα συμπεράσματα:

- Είναι ξεκάθαρο πλέον πως οι Web Workers μπορούν να συνεισφέρουν σημαντικά στην υλοποίηση web εφαρμογών με σημαντικό υπολογιστικό φόρτο, καθιστώντας εφικτή την χρήση τέτοιων εφαρμογών σε επίπεδο client-side, καθώς μέχρι σήμερα οι περισσότερες εφαρμογές που έχουν να επιτελέσουν κάποια «βαριά» εργασία εκτελούνται συνήθως σε επίπεδο server-side.

- Είδαμε πως η χρήση των Web Workers ακόμη σε συσκευές με χαμηλότερη επεξεργαστική δύναμη όπως ένα smartphone, μπορούν να βελτιώσουν σε πολύ μεγάλο βαθμό τις επιδόσεις των συσκευών αυτών δεδομένου ότι είναι σύνηθες πλέον να χρησιμοποιούν επεξεργαστές με πολλούς πυρήνες

- Ο αριθμός των Web Workers που θα πρέπει να χρησιμοποιήσει μια εφαρμογή ώστε να επιτύχει την καλύτερη απόδοση εξαρτάται από διάφορους παράγοντες και δεν είναι πάντα εύκολο να υπολογισθεί. Γενικά, η χρήση αριθμού web workers ίσου με τους πραγματικούς πυρήνες (physical cores) ενός συστήματος συνήθως αποδίδει ένα αποτέλεσμα πολύ κοντά ή και ίσο με την βέλτιστη απόδοση αλλά ακόμη και έτσι με την προϋπόθεση ότι δεν γίνεται εκτέλεση άλλων σημαντικών εφαρμογών στο παρασκήνιο

- Ο φυλλομετρητής που θα χρησιμοποιηθεί για την εκτέλεση μιας web εφαρμογής είδαμε ότι επηρεάζει την απόδοση αλλά θα λέγαμε ότι δεν υπάρχουν τόσο σημαντικές διαφορές ώστε να ξεχωρίσουμε κάποιο ως καλύτερη επιλογή. Ένας παράγοντας που επηρεάζει την απόδοση με διαφορετικό τρόπο σε κάθε φυλλομετρητή είναι ο τύπος της εφαρμογής που εκτελείται.

Κλείνοντας, θεωρούμε πως ο βασικός στόχος της διπλωματικής αυτής υλοποιήθηκε, καθώς έγινε παρουσίαση της τεχνολογίας των Web Workers και της δυναμικής που μπορούν να προσδώσουν στις σύγχρονες εφαρμογές διαδικτύου, δεδομένου ότι οι ανάγκες τους σε επεξεργαστική ισχύ αυξάνονται συνεχώς. Με την αποτίμηση των συμπερασμάτων που αποκομίσαμε από την σχετική βιβλιογραφία αλλά και από τα συγκριτικά τεστ που υλοποιήθηκαν, εκτιμούμε ότι έγιναν πιο ξεκάθαρα τα οφέλη και η αξία των HTML5 Web Workers.

4.2 Μελλοντικές Επεκτάσεις

Ως μελλοντική επέκταση της παραπάνω μελέτης, θεωρούμε ότι θα είχε αρκετό ενδιαφέρον να δούμε πως θα μπορούσε να προσαρμοστεί το Web Worker API στις υπάρχουσες βιβλιοθήκες τις Javascript που εκτελούν πολύπλοκες και χρονοβόρες διαδικασίες. Θα είχε μεγάλη αξία να δούμε τις δυσκολίες που θα έπρεπε να αντιμετωπιστούν για την ενσωμάτωση του API αλλά και τα οφέλη που θα μπορούσαν οι βιβλιοθήκες αυτές να αποκομίσουν από την αξιοποίηση της δυνατότητας για multithreading εκτέλεση, στους σύγχρονους επεξεργαστές.

Βιβλιογραφία

Aiman Erbad, Norman C. Hutchinson, Charles ’Buck’ Krasic, (2012). DOHA: scalable real-time web applications through adaptive concurrent execution, WWW '12 Proceedings of the 21st international conference on World Wide Web. DOI:10.1145/2187836.2187859

Emily Fortuna, Owen Anderson Luis Ceze Susan Eggers, (2010). A limit study of JavaScript parallelism, IEEE International Symposium on Workload Characterization (IISWC'10). DOI: 10.1109/IISWC.2010.5649419

Hyung Woo Kim, Yang-Won Lee (2013). Single and Multiple Thread Programming for Geo-visualization by Using WebGL with Web Workers. Proceedings of the World Congress on Engineering and Computer Science 2013 Vol I WCECS 2013, 23-25 October, 2013, San Francisco, USA. Retrieved from: https://pdfs.semanticscholar.org/ca06/2406df394fbb17b6a463f257ec4112c21e31.pdf

IBM, Use Web Workers to improve the usability of your web applications, 2010. Available from:https://www.ibm.com/developerworks/library/wa-webworkers/index.html
[Accessed on October 2018]

Javier Verdú, Juan José Costa and Alex Pajuelo, (2015). Dynamic web worker pool management for highly parallel javascript web applications. Concurrency and Computation: Practice and Experience, Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3739

Javier Verdú, Juan José Costa and Alex Pajuelo (2016) Performance Scalability Analysis of JavaScript Applications with Web Workers, IEEE Computer Architecture Letters, DOI: 10.1109/LCA.2015.2494585

Jayakrishnan Radhakrishnan, (2015). Hardware dependency and performance of JavaScript engines used in popular browsers, 2015 International Conference on Control Communication & Computing India (ICCC). DOI: 10.1109/ICCC.2015.7432981

Martinsen, J. K., Grahn, H., & Isberg, A. (2011). A comparative evaluation of JavaScript execution behavior. Paper presented at PLDI 2011 - 32nd ACM SIGPLAN conference on Programming Language Design and Implementation, San Jose, CA, United States.

Matthias Wenzel, Christoph Meinel, (2015). Parallel Network Data Processing in Client Side JavaScript Applications, 2015 International Conference on Collaboration Technologies and Systems (CTS). DOI: 10.1109/CTS.2015.7210414

Mozilla Developer Network. Web Workers API, 2015. Available from: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API)
[Accessed on October 2018]

Takuya Sumi, Tsukasa Ishiguro, Shinsaku Kiyomoto, Yutaka Miyake, Toru Kobayashi, Tsuyoshi Takagi (2014). Parallel Implementation of Public Key Cryptosystems using Web Workers. 2014 IEEE 11th Consumer Communications and Networking Conference DOI: 10.1109/CCNC.2014.6940513

Vishal Anand, Deepanker Saxena, (2013). Comparative study of modern web browsers based on their performance and evolution, 2013 IEEE International Conference on Computational Intelligence and Computing Research. DOI: 10.1109/ICCIC.2013.6724273

Web Workers Multithreaded Programs in JavaScript, O'Reilly Media (2012). http://shop.oreilly.com/product/0636920024446.do

Yuta Watanabe, Shusuke Okamoto, Masaki Kohana, Masaru Kamada, Tatsuhiro Yonekura, (2013). A Parallelization of Interactive Animation Software with Web Workers. 16th International Conference on Network-Based Information Systems. DOI: 10.1109/NBiS.2013.74