Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Κοινόχρηστο με τον host σύστημα αρχείων σε unikernel

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Φώτιος Ζαφείρης Μ. Ξενάκης

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2020

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Κοινόχρηστο με τον host σύστημα αρχείων σε unikernel

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Φώτιος Ζαφείρης Μ. Ξενάκης

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την -ΤΟΔΟ: ημερομηνία-.

.....	.....	.....
Νεκτάριος Κοζύρης	Γεώργιος Γκούμας	Διονύσιος Πνευματικάτος
Καθηγητής Ε.Μ.Π.	Επίκουρος καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2020

.....

Φώτιος Ζαφείρης Μ. Ξενάκης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Το cloud computing είναι πλέον εδραιωμένο ως η κυρίαρχη προσέγγιση προς την υπολογιστική υποδομή. Η πληθώρα των υπηρεσιών που παρέχονται από αυτήν την υποδομή έχει ως θεμέλιο λίθο την τεχνολογία της εικονικοποίησης (virtualization). Καθώς το cloud υιοθετείται όλο και περισσότερο σε παγκόσμια κλίμακα, το αυξανόμενο μέγεθος των υπολογιστικών πόρων καθιστά ολοένα και πιο επιτακτική την αποδοτική χρήση αυτών από το λογισμικό. Προς αυτό τον σκοπό, μια λύση είναι τα *unikernels*, πυρήνες λειτουργικών συστημάτων εξειδικευμένοι για να τρέχουν ένα στιγμιότυπο μίας μόνο εφαρμογής κάθε φορά, εξοικονομώντας πόρους σε σχέση με έναν γενικού σκοπού πυρήνα. Έτσι, πλεονεκτούν έναντι στα γενικού σκοπού λειτουργικά συστήματα ως προς την αποδοτικότητα, ενώ ταυτόχρονα προσφέρουν πιο ισχυρή απομόνωση από τα containers, τα οποία αποτελούν μια πιο πρόσφατη, δημοφιλή εναλλακτική.

Η αποδοτική και ταυτόχρονα ασφαλής πρόσβαση των virtualized guests στους πόρους του host συστήματος είναι μια μεγάλη πρόκληση που συνοδεύει την εικονικοποίηση. Σε αυτόν τον τομέα, σημαντική συνεισφορά αποτελεί το *virtio*, μια προδιαγραφή για paravirtualized συσκευές που καθιστά εφικτή την αποδοτική χρήση των πόρων του host. Για την κοινοχρησία αρχείων ανάμεσα σε host και guest μέχρι πρόσφατα οι διαθέσιμες λύσεις βασίζονταν στην εικονική σύνδεση δικτύου μεταξύ των δύο, με υποβέλτιστα αποτελέσματα αφού δεν εκμεταλλεύονταν το γεγονός της συνύπαρξης τους στο ίδιο φυσικό μηχάνημα. Προκειμένου να επιλύσει αυτό το ζήτημα έχει προταθεί το *virtio-fs*, μια εικονική virtio συσκευή που προσφέρει πρόσβαση σε έναν κατάλογο του συστήματος αρχείων του host από τον guest, παρέχοντας υψηλές επιδόσεις και σημασιολογία τοπικού συστήματος αρχείων.

Αντικείμενο της παρούσας διπλωματικής εργασίας είναι η υλοποίηση και αξιολόγηση της χρήσης του virtio-fs σε ένα unikernel. Δείχνουμε ότι ο συνδυασμός αυτών των δύο έχει σημαντικά πλεονεκτήματα, τόσο από άποψη επιδόσεων, όσο και από διαχειριστική άποψη στο πλαίσιο του cloud. Επίσης, φροντίζουμε ώστε το προϊόν της εργασίας να έχει πρακτική αξία, συνεισφέροντας το στο έργο ανοιχτού λογισμικού του unikernel στο οποίο βασιστήκαμε.

Λέξεις κλειδιά

εικονικοποίηση, νέφος, σύστημα αρχείων, unikernel, virtio, OSv, virtio-fs, QEMU

Abstract

Cloud computing has been established as the dominant approach to compute infrastructure. The multitude of services it offers is grounded in virtualization technology. As cloud adoption grows worldwide, the expansion of its compute resources renders their efficient utilization by software imperative. One solution towards that is *unikernels*, operating system kernels specialized to run a single instance of a single application at a time, sparing resources compared to a general-purpose kernel. As such, they have an advantage over general-purpose operating systems as far as resource efficiency is concerned, while also providing stronger isolation than containers, a more recent, popular alternative.

Efficient while also secure virtualized guests' access to the underlying host's resources is a substantial challenge that comes with virtualization. In this aspect, *virtio* has been a significant contribution, as a specification of paravirtual devices enabling efficient usage of the host's resources. For host-guest file sharing, until recently the solutions available relied on the virtual network connection between the two, with suboptimal results, since their colocation on the same physical machine was exploited. To address this issue, *virtio-fs* has been proposed, as a virtio device offering guest access to a file system directory on the host, providing high performance and local file system semantics.

This thesis is concerned with the implementation and evaluation of virtio-fs in a unikernel. We demonstrate that combining the two offers great benefits, both with regard to performance and the operational aspect in a cloud context. Moreover, we cater to the practical value of this work by contributing it to the open source project behind the unikernel we based it on.

Keywords

virtualization, cloud, file system, unikernel, virtio, OSv, virtio-fs, QEMU

Ευχαριστίες

Για την παρούσα εργασία, που σηματοδοτεί την ολοκλήρωση μίας πορείας ετών, θα ήθελα να ευχαριστήσω τα μέλη του εργαστηρίου υπολογιστικών συστημάτων, υπό την αιγίδα του οποίου πραγματοποιήθηκε. Πιο πολύ όμως θα ήθελα να τους ευχαριστήσω, όπως και άλλα μέλη της σχολής ΗΜΜΥ, για τη διδασκαλία, το γνήσιο ενδιαφέρον και την καλλιέργεια της κουλτούρας του μηχανικού που μου προσέφεραν.

Επίσης, οφείλω ένα μεγάλο ευχαριστώ στους ανθρώπους των κοινοτήτων του OSv και του virtio-fs για την υποστήριξη, τις συμβουλές και το χρόνο τους, αλλά κυρίως για την ανοικτότητα, το ήθος και το έργο τους, που ενέπνευσαν αυτή τη συμβολή.

Τέλος, αυτοί για τους οποίους είμαι περισσότερο ευγνώμων είναι η οικογένεια και οι φίλοι μου, που πάντα βρίσκονται στο πλευρό μου, με ανέχονται και με στηρίζουν και χωρίς τους οποίους τίποτα δεν θα μπορούσε να επιτευχθεί.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

Κεφάλαιο 2

Υπόβαθρο

Κεφάλαιο 3

Υλοποίηση

Κεφάλαιο 4

Αξιολόγηση

Για την αξιολόγηση του νέου συστήματος αρχείων στο OSv, διεξάγαμε δοκιμές συγκρίνοντας με τα υπόλοιπα διαθέσιμα συστήματα αρχείων αλλά και το virtio-fs στο linux, όπου η σύγκριση είχε νόημα. Οι δοκιμές περιλαμβάνουν τρία σενάρια, όπως αναλύονται στη συνέχεια: ένα συνθετικό μετροπρόγραμμα (synthetic benchmark ή microbenchmark), μια δοκιμή χρόνου εκκίνησης και τέλος μία πραγματική εφαρμογή (application benchmark).

4.1 Μεθοδολογία

Όλες οι δοκιμές πραγματοποιήθηκαν σε προσωπικό υπολογιστή με τα στοιχεία που αναφέρονται στον πίνακα 4.1, ενώ ο πίνακας 4.2 περιλαμβάνει τις προδιαγραφές των OSv και linux guests. Ως προς τη διεξαγωγή τους πρέπει να αναφερθούν τα εξής:

- Για την εκτέλεση όλων των δοκιμών χρησιμοποιήθηκε ένα προσωρινό σύστημα αρχείων (*tmpfs*) στον host, τόσο για τις εικόνες του OSv όσο και για τους κοινόχρηστους καταλόγους στην περίπτωση των virtio-fs και NFS. Αυτό έγινε ώστε να μην επηρεάζονται οι επιδόσεις από τη συσκευή αποθήκευσης του host.

- Η δυναμική προσαρμογή της συχνότητας της CPU ήταν απενεργοποιημένη, χρησιμοποιώντας τον *"performance" CPU scaling governor* στον host.

- Η διεργασία του QEMU ήταν απομονωμένη από τις υπόλοιπες (virtiofsd, perf, vegeta) ως προς τις CPUs στις οποίες εκτελούνταν (*CPU pinning*). Συγκεκριμένα, στο QEMU αφιερώνονταν επεξεργαστικοί πυρήνες ισάριθμοι με το πλήθος των CPUs του guest, ενώ οι υπόλοιπες προαναφερόμενες διεργασίες δεσμεύονταν στους υπόλοιπους διαθέσιμους. Αυτό έγινε με σκοπό την ελαχιστοποίηση των παρεμβολών που θα μπορούσαν να επηρεάσουν τις επιδόσεις.

- Για τη μέτρηση της χρησιμοποιούμενης επεξεργαστικής ισχύος (CPU usage) επελέγη το *perf*. Συγκεκριμένα, χρησιμοποιήθηκε η εντολή `perf stat`, με το "task-clock" perf event. Η μέτρηση έγινε για τη διεργασία του QEMU, το

Επεξεργαστής	Intel Core i7-6700 @3.4GHz
Μνήμη	2×8 GiB @2666MHz
Swap	Όχι
Linux kernel	5.8.13-arch1-1
QEMU	5.1.50 @ c37a890d12e57a3d28c3c7ff50ba6b877f6fc2cc

Πίνακας 4.1: Προδιαγραφές του host όπου διεξήχθησαν οι δοκιμές.

virtiofsd, το vhost kernel thread και τα NFS kernel threads (για το καθένα όπου ήταν εφαρμόσιμο):

- Όλες οι δοκιμές επαναλήφθηκαν *10 φορές*, από τις οποίες στη συνέχεια παρουσιάζονται η μέση τιμή mean και η τυπική απόκλιση standard deviation. Στην περίπτωση του OSv κάθε επανάληψη ήταν μία εκ νέου εκκίνηση της εικονικής μηχανής, ενώ στο linux όλες οι επαναλήψεις έγιναν στην ίδια εκτέλεση της εικονικής μηχανής.

- Το virtiofsd εκτελούνταν με το cache mode απενεργοποιημένο (`cache=none`) και ένα thread (`-thread-pool-size=1`). Το δεύτερο επελέγη διότι οδηγούσε σε ελαφρώς πιο συνεπείς μετρήσεις, χωρίς όμως να παρατηρείται καλύτερη επίδοση όπως είχε αναφερθεί σε συζήτηση στη mailing list του virtio-fs[1].

- Για την εκτέλεση του OSv έγινε χρήση του βοηθητικού script (scripts/run.py) που παρέχει, το οποίο τροποποιήθηκε για την διεξαγωγή των δοκιμών, ώστε να ενορχηστρώνει την εκτέλεση όλων των υπόλοιπων εργαλείων (virtiofsd, perf, vegeta). Ως προς τη δικτύωση της εικονικής μηχανής, χρησιμοποιήθηκε το *tap backend* του QEMU με το *vhost* ενεργοποιημένο, ενώ στο VM δινόταν στατική διεύθυνση IP.

- Ως NFS server χρησιμοποιήθηκε η αντίστοιχη υλοποίηση του linux στον host. Όλες οι δοκιμές έγιναν με την έκδοση 3 του NFS, καθώς η έκδοση 4 δεν ήταν λειτουργική από την πλευρά του OSv και με readahead στον NFS client (libnfs) ίσο με 2 MiB (όσο και το αντίστοιχο μέγεθος στην υλοποίηση μας του virtio-fs DAX).

4.2 Microbenchmark

4.2.1 Περιγραφή

Για τη μέτρηση των επιδόσεων του συστήματος αρχείων χρησιμοποιήσαμε το flexible I/O tester (fio) στην έκδοση 3.23. Προκειμένου να εκτελεστεί στο OSv χρει-

[1] `https://www.redhat.com/archives/virtio-fs/2020-September/msg00068.html`

CPUs	4
Μνήμη	4 GiB
DAX window	4 GiB
OSv	5372a230ce0abf0dc72e92ec1116208145e595c5
Linux kernel	5.8.0-rc4-33261-gfaa931f16f27

Πίνακας 4.2: Προδιαγραφές των guests.

άστηκαν μόνο δύο τροποποιήσεις[2] και τα κατάλληλα ορίσματα στο configure script του fio ώστε να απενεργοποιηθούν χαρακτηριστικά που δεν παρέχονται από το OSv. Σημειώνουμε ότι το ίδιο ακριβώς εκτελέσιμο (με τα παραπάνω απενεργοποιημένα) χρησιμοποιήθηκε και στο linux, εκτός από το OSv.

Στη σύγκριση περιλαμβάνονται τα ZFS, rofs, ramfs, virtio-fs (με και χωρίς DAX window, με ramfs root file system) και NFS στο OSv, το virtio-fs (με και χωρίς DAX window, με ext4 root file system) στο linux, καθώς και το tmpfs στον host, το οποίο συμπεριλάβαμε για πληρότητα, ως σημείο αναφοράς (baseline). Συγκεκριμένα για τη διαδικασία έχουμε:

- Σε όλες τις περιπτώσεις εκτός του ramfs, τα αρχεία ελέγχου (test files) του fio έχουν παραχθεί εκ των προτέρων από το ίδιο και έχουν τοποθετηθεί στην εκάστοτε τοποθεσία (εικονικό δίσκο ή κοινόχρηστο κατάλογο). Αυτό γίνεται παρότι το fio μπορεί να τα παράξει δυναμικά κατά το χρόνο εκτέλεσης, διότι κάποια από τα συστήματα αρχείων είναι read-only. Επειδή το ramfs είναι περιορισμένο ως προς το μέγεθος μεμονωμένων αρχείων στο image του, στην περίπτωση του τα αρχεία παράγονται κατά το χρόνο εκτέλεσης.

- Εξετάζονται δύο περιπτώσεις ως προς τα αρχεία ελέγχου:

  - Ένα μεγάλο αρχείο (1 GiB).
  - Πολλαπλά (10) μικρότερα αρχεία (80-100 MiB). Σημειώνουμε ότι τα αρχεία σε όλες τις δοκιμές είναι πανομοιότυπα (το αντίστοιχο αρχείο έχει το ίδιο μέγεθος πάντα).

  Και στις δύο περιπτώσεις το συνολικό μέγεθος των αρχείων δεν ξεπερνά το 1 GiB. Αυτό εν μέρει γίνεται διότι το μέγεθος αυτό είναι περιορισμένο από τη μνήμη του guest για τα rofs και ramfs και το μέγεθος του DAX window για το virtio-fs με DAX.

- Εξετάζονται δύο περιπτώσεις ως προς το πρότυπο (pattern) ανάγνωσης των αρχείων: σειριακό (`rw=read`) και τυχαίο (`rw=randread`).

- Σε όλες τις περιπτώσεις υπάρχει μόνο *ένα thread* ανάγνωσης (`numjobs=1`).

[2]Όλες οι αλλαγές βρίσκονται στο 'osv' branch του git repository `https://github.com/foxeng/fio`.

pattern	virtio-fs	NFS	virtio-fs DAX
σειριακό	1,0	3,1	6,5
τυχαίο	1,0	0,4	5,7

Πίνακας 4.3: Κανονικοποιημένο fio throughput virtio-fs και NFS στο OSv.

- Στο linux, για την αυτοματοποίηση των δοκιμών βασιστήκαμε στο σχετικό βοήθημα του Vivek Goyal. Αξίζει να σημειωθεί ότι το συγκεκριμένο ακυρώνει τις page, inode και dentry caches χρησιμοποιώντας το sysctl (/proc/sys/vm/drop_caches) πριν από κάθε εκτέλεση του fio.
- Στην περίπτωση του linux (guest και host) δεν έγινε μέτρηση του CPU usage γιατί μία τέτοια σύγκριση κρίθηκε ότι δεν έχει αξία, δεδομένου του διαφορετικού χαρακτήρα του από το OSv (λειτουργικό γενικού σκοπού από τη μία και unikernel από την άλλη).

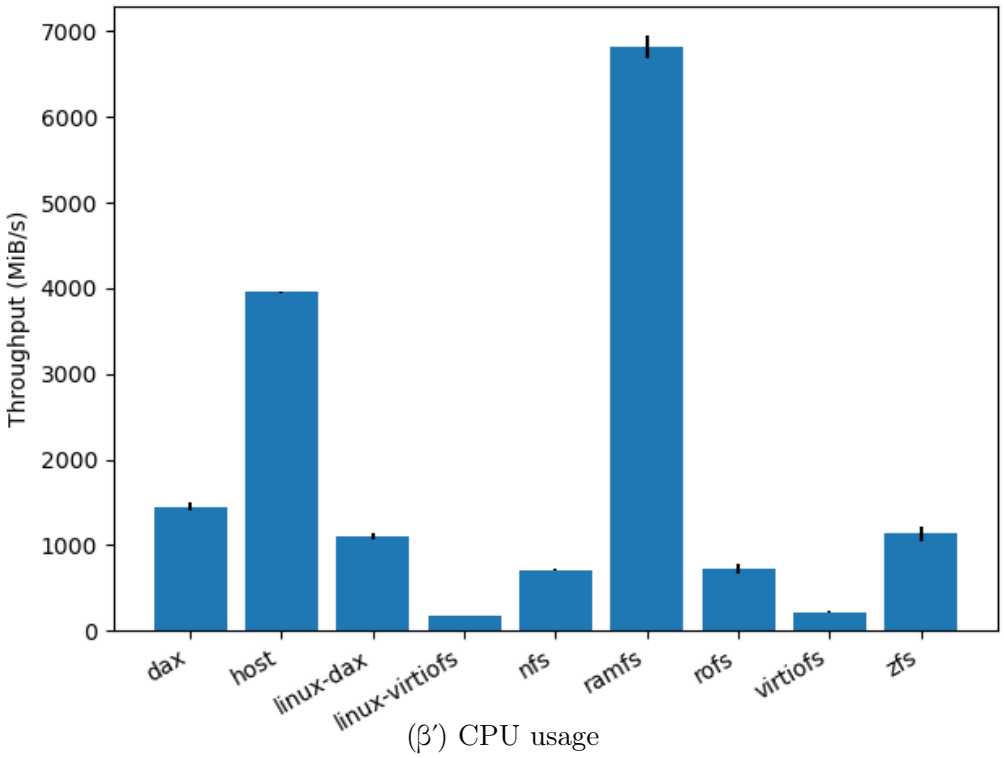
4.2.2 Αποτελέσματα

Όπως βλέπουμε στα σχήματα 4.1 έως 4.4, το υψηλότερο throughput επιτυγχάνεται από το ramfs στο OSv, υψηλότερο και από αυτό του tmpfs στον host. Αυτό συμβαίνει διότι, με τα δεδομένα στη μνήμη το virtualization overhead ελαχιστοποιείται, ενώ ταυτόχρονα η απλοποιημένη υλοποίηση του συστήματος αρχείων και η έλλειψη mode switches λόγω κλήσεων συστήματος στο OSv φαίνεται ότι κάνουν τη διαφορά. Το virtio-fs με DAX προσφέρει τις αμέσως καλύτερες επιδόσεις στο OSv, σε όλες τις περιπτώσεις, και παράλληλα έχει τη μικρότερη επιβάρυνση σε όρους επεξεργαστικής ισχύος, και πάλι μετά το ramfs.

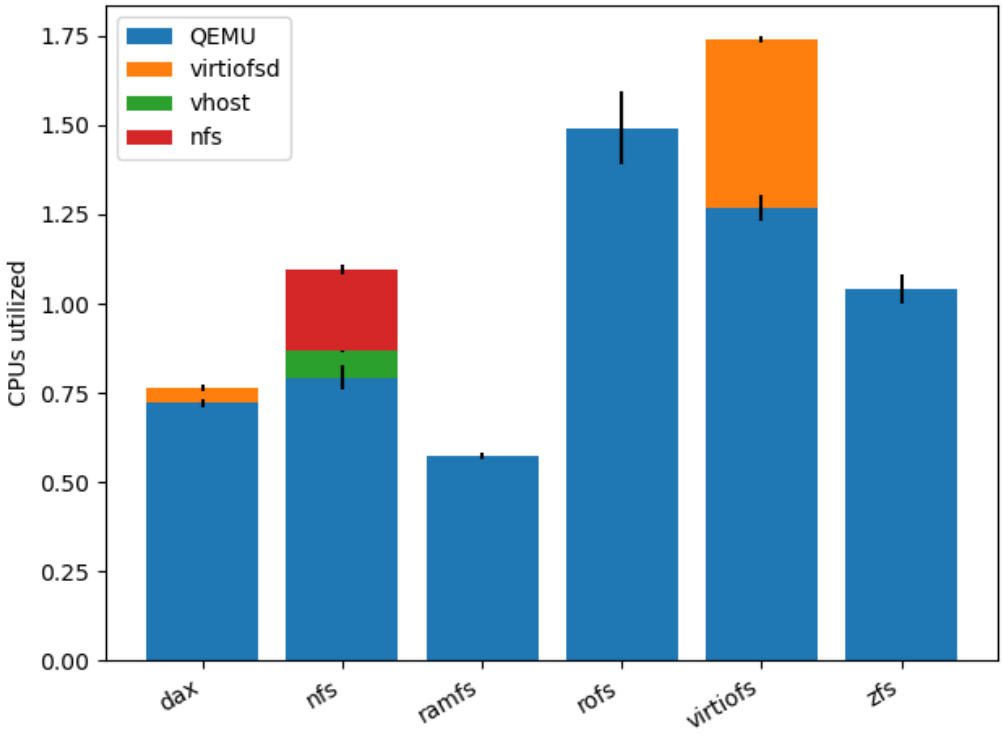
Εστιάζοντας στο virtio-fs και συγκρίνοντας ανάμεσα σε OSv και linux, διακρίνουμε συνεπή συμπεριφορά σε όλες τις περιπτώσεις: σε αμφότερα τα λειτουργικά το DAX window υπερτερεί του virtio-fs χωρίς αυτό με μεγάλη διαφορά ($> 6\times$ throughput), ενώ στο OSv βλέπουμε $20 - 30\%$ καλύτερες επιδόσεις από ότι στο linux. Το δεύτερο είναι αναμενόμενο, αφενός λόγω της απλούστερης υλοποίησης στο OSv, που υποστηρίζει πολύ λιγότερες λειτουργίες και αφετέρου λόγω των συγκριτικών πλεονεκτημάτων ενός unikernel.

Ιδιαίτερη μνεία οφείλουμε στη σύγκριση ανάμεσα σε virtio-fs και NFS, τα μόνα κοινόχρηστα συστήματα αρχείων στο OSv. Εδώ το virtio-fs με DAX window έχει το προβάδισμα στις επιδόσεις, με το NFS να ακολουθεί και το virtio-fs χωρίς DAX να βρίσκεται τελευταίο, στις δοκιμές με σειριακή ανάγνωση. Όπως φαίνεται και στον πίνακα 4.3, στις δοκιμές τυχαίας ανάγνωσης τα αποτελέσματα διαφοροποιούνται, με το NFS να έχει το χειρότερο throughput (και μεγαλύτερο CPU usage), έχοντας επηρεαστεί σαφώς περισσότερο σε σχέση με το virtio-fs από την αλλαγή στο access pattern.

(α′) Throughput

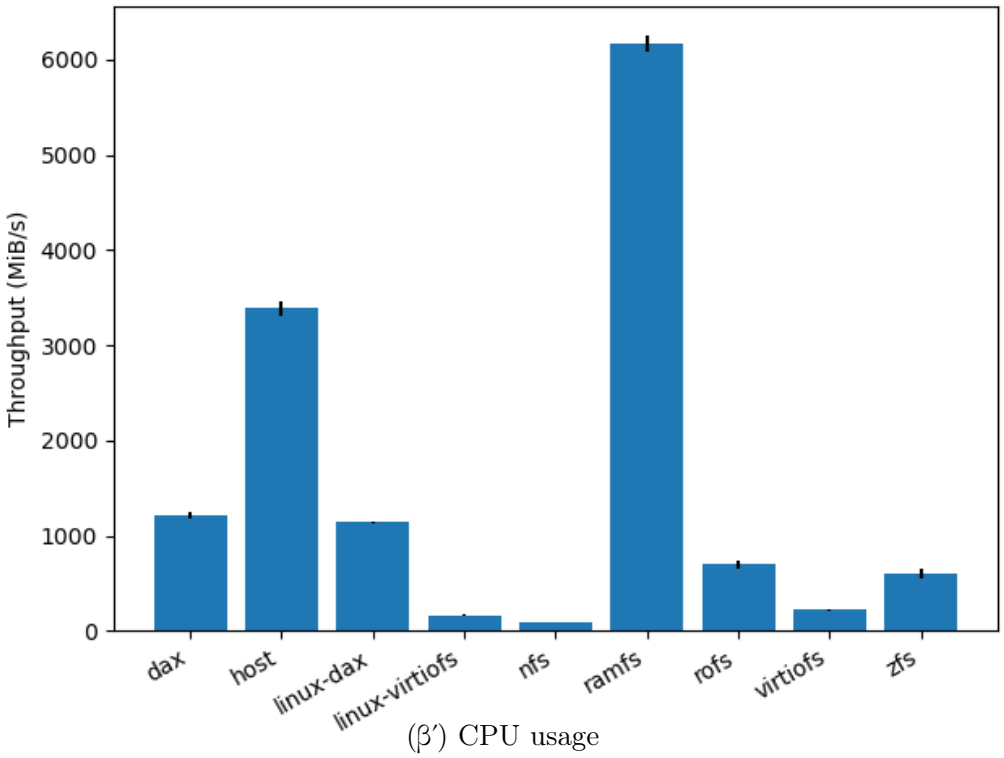


(β′) CPU usage

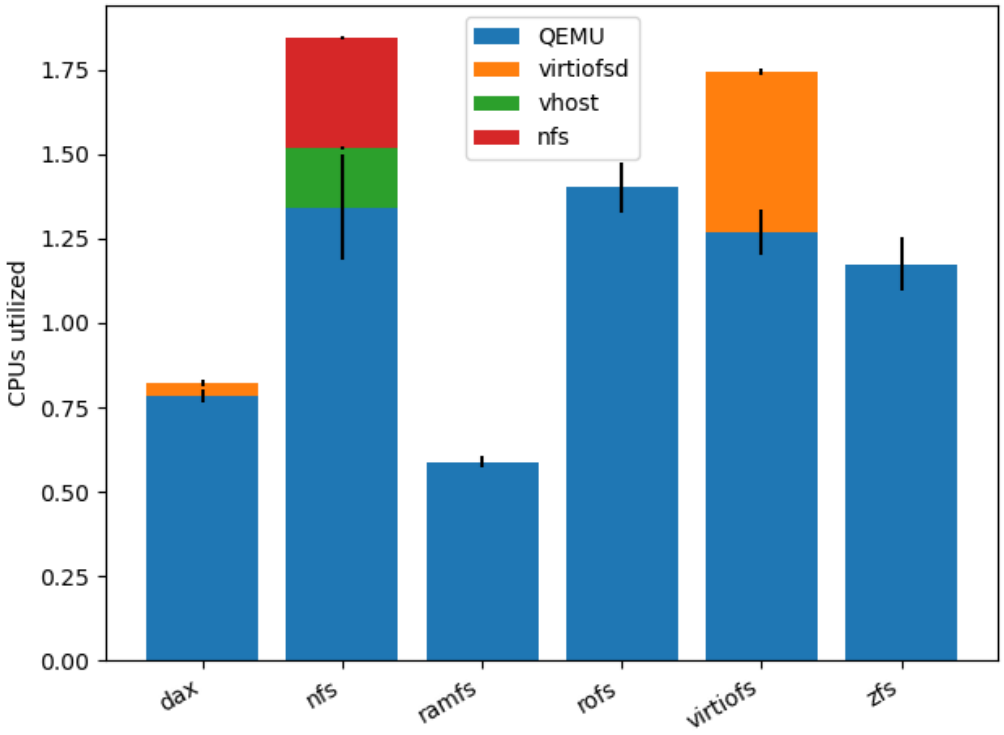


Σχήμα 4.1: fio, ένα αρχείο, σειριακή ανάγνωση

(α′) Throughput

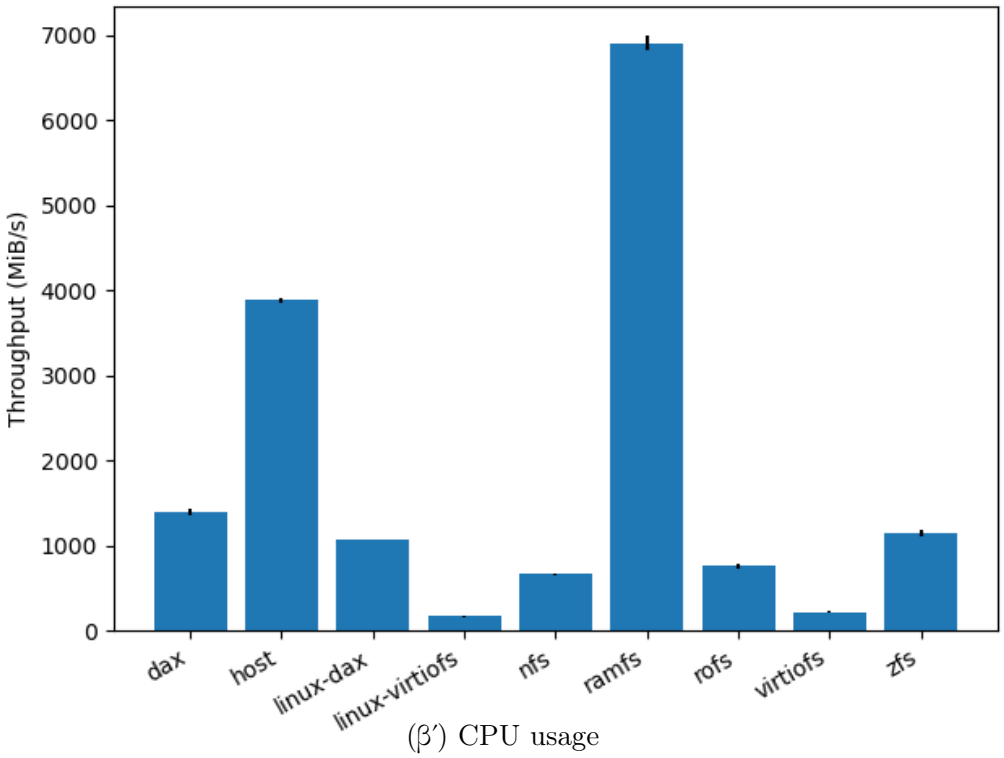


(β′) CPU usage

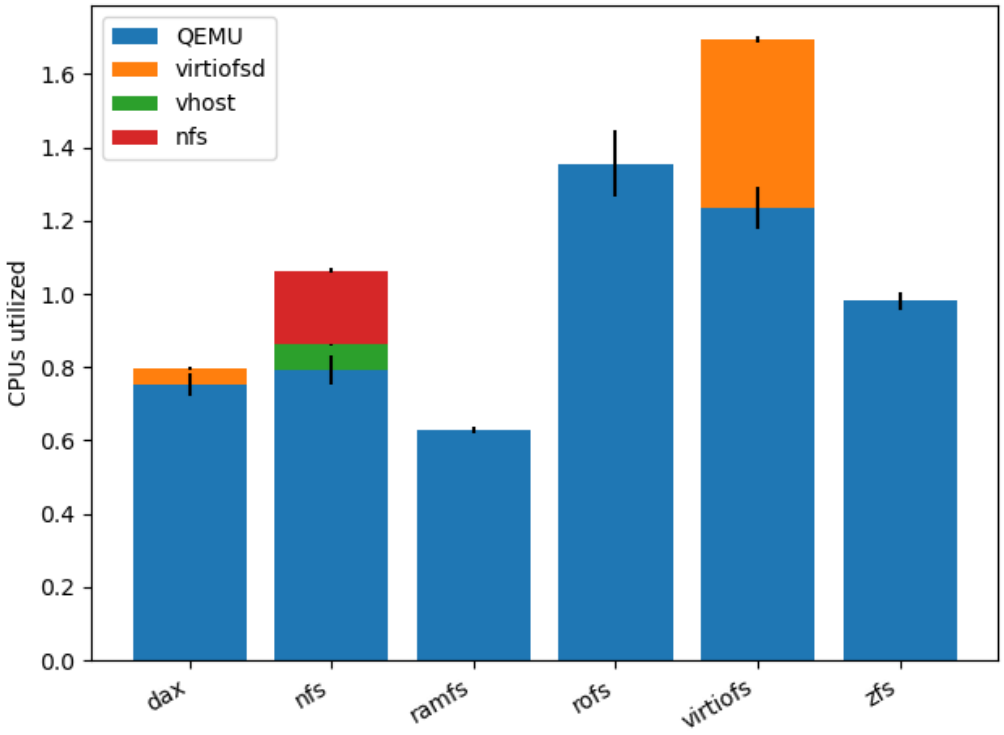


Σχήμα 4.2: fio, ένα αρχείο, τυχαία ανάγνωση

(α′) Throughput

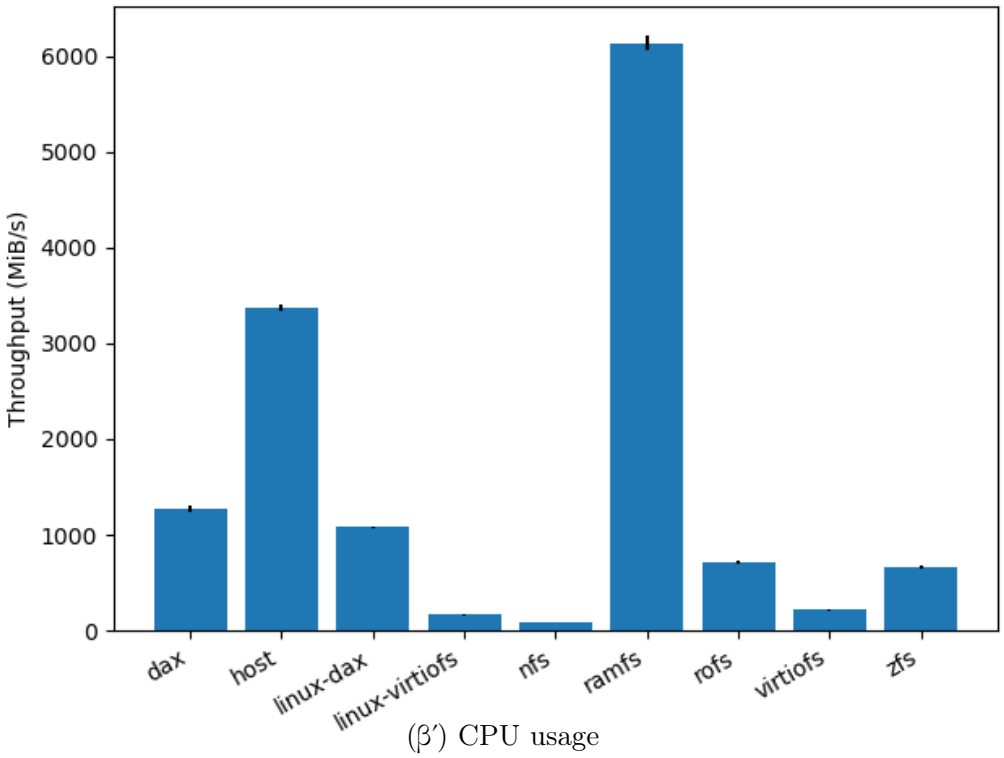


(β′) CPU usage

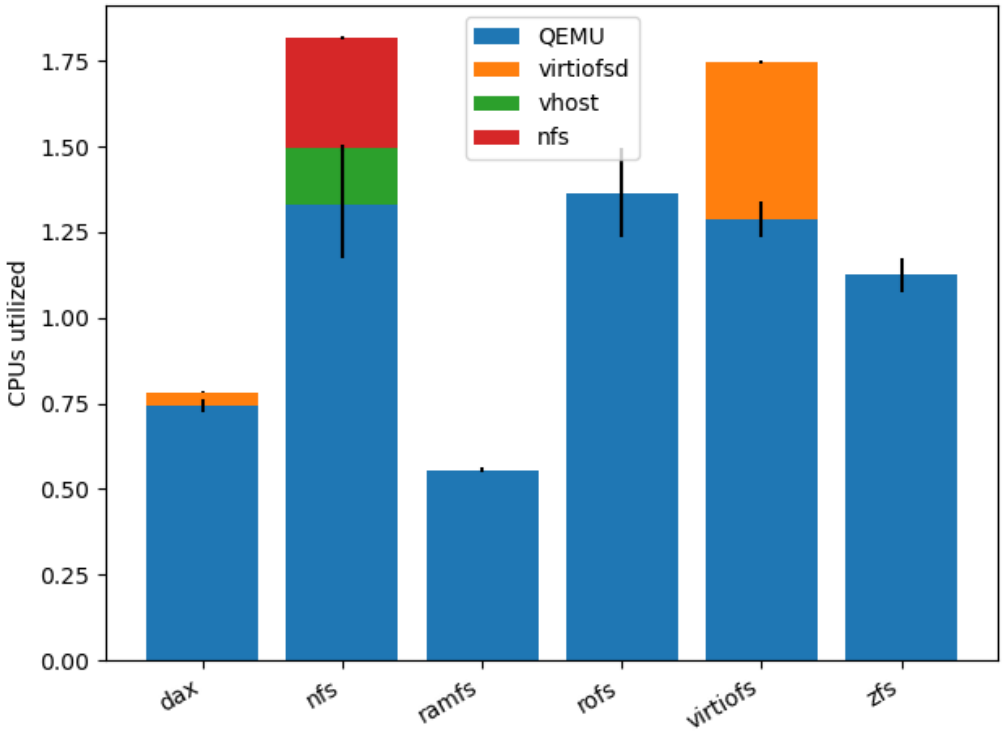


Σχήμα 4.3: fio, πολλαπλά αρχεία, σειριακή ανάγνωση

(α′) Throughput



(β′) CPU usage



Σχήμα 4.4: fio, πολλαπλά αρχεία, τυχαία ανάγνωση

4.3 Χρόνος εκκίνησης

4.3.1 Περιγραφή

Προκειμένου να αξιολογήσουμε τον χρόνο εκκίνησης στο virtio-fs επιλέξαμε μία εφαρμογή από αυτές που έχουν ήδη μεταφερθεί στο OSv ως παραδείγματα. Συγκεκριμένα, επελέγη το spring-boot-example, μια απλή web εφαρμογή από το χτισμένη με το spring boot framework, στην έκδοση του 2.3.4.[3] Ως Java runtime επιλέξαμε και πάλι από την ίδια συλλογή εφαρμογών το openjdk8-zulu-full. Η επιλογή της συγκεκριμένης εφαρμογής έγινε καθώς θεωρήθηκε αντιπροσωπευτική, ως stateless web app, εφαρμογής που θα γινόταν deploy ως unikernel, σε ένα πλαίσιο cloud.

Στις δοκιμές συμμετείχαν όλα τα συστήματα αρχείων του OSv τα οποία μπορούν να χρησιμοποιηθούν ως root file systems: ZFS, rofs, ramfs και virtio-fs.

Η διαδικασία περιελάμβανε την εκκίνηση του OSv με το εκάστοτε σύστημα αρχείων να χρησιμοποιείται ως root file system. Αυτό αφηνόταν να τρέξει για ένα εύλογο χρονικό διάστημα κάποιων δευτερολέπτων, μέσα στο οποίο ολοκληρωνόταν η πλήρης αρχικοποίηση της εφαρμογής και στη συνέχεια τερματιζόταν από το μηχανισμό (script) ενορχήστρωσης των δοκιμών, τερματίζοντας τη διεργασία του QEMU.

4.3.2 Αποτελέσματα

Στο σχήμα 4.5 απεικονίζεται μία ανάλυση του συνολικού χρόνου εκκίνησης ως εξής:

OSv boot είναι ο χρόνος εκκίνησης (boot) του συστήματος του OSv, δηλαδή του μέρους που είναι ανεξάρτητο της εκάστοτε εφαρμογής.

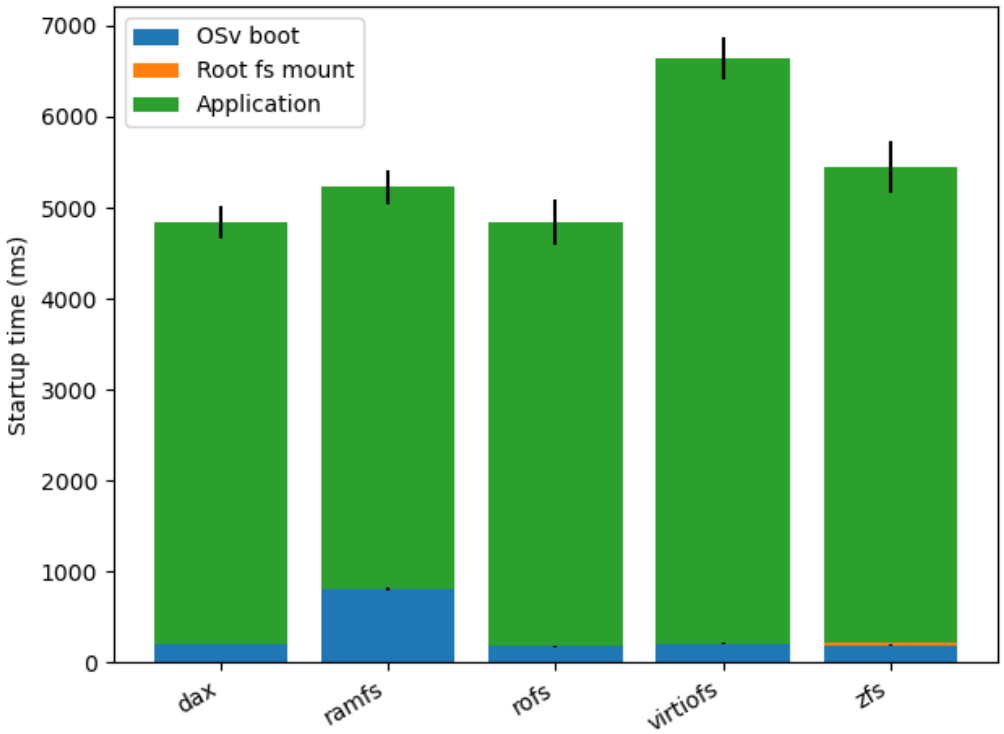
Root fs mount είναι ο χρόνος προσάρτησης (mount) του root file system και της αλλαγής της 'ρίζας' (/) του εικονικού συστήματος αρχείων σε αυτό (pivot). Επισημαίνεται ότι αυτό είναι ένα στάδιο του προηγούμενου, αλλά εν προκειμένω έχει αφαιρεθεί από εκείνο και παρουσιάζεται χωριστά.

Application είναι ο χρόνος αρχικοποίησης της ίδιας της εφαρμογής, όπως καταγράφεται από αυτήν.

Όπως βλέπουμε καλύτερα στον πίνακα 4.4, με όρους συνολικού χρόνου εκκίνησης, το virtio-fs *χωρίς* DAX window υστερεί έναντι των υπολοίπων, ενώ το virtio-fs *με* DAX window έχει την καλύτερη επίδοση, μαζί με το rofs (τα ramfs και ZFS είναι ελαφρώς πιο αργά).

Όσον αφορά τον χρόνο προσάρτησης (mount time), για όλα τα συστήματα αρχείων είναι πρακτικά αμελητέος (< 2% του OSv boot time), *εκτός από το ZFS*. Στην περίπτωση του τελευταίου, ο χρόνος προσάρτησης αποτελεί περίπου το 14% του

[3] Όλες οι αλλαγές που έγιναν για τις δοκιμές μας βρίσκονται στο 'virtiofs-tests' branch του git repository `https://github.com/foxeng/osv-apps`.



Σχήμα 4.5: Spring boot example, χρόνοι εκκίνησης

ZFS	rofs	ramfs	virtio-fs	virtio-fs DAX
1,13	1,00	1,08	1,37	1,00

Πίνακας 4.4: Κανονικοποιημένος συνολικός χρόνος εκκίνησης spring-boot-example στο OSv.

χρόνου εκκίνησης του συστήματος, κάτι αναμενόμενο δεδομένης της πολυπλοκότητας του συγκεκριμένου συστήματος αρχείων, η οποία μεταφράζεται σε μία σαφώς πιο μακρά διαδικασία αρχικοποίησης.

Τέλος, αξίζει να αναφερθούμε στον αισθητά αυξημένο χρόνο εκκίνησης του OSv στην περίπτωση του ramfs. Αυτή η διαφοροποίηση δικαιολογείται εάν αναλογιστούμε ότι στην περίπτωση του ramfs, το root file system *ταυτίζεται* με το boot file system (σε ορολογία OSv, το αντίστοιχο initramfs στο linux). Αυτό είναι μέρος του ELF object που περιέχει το kernel, το οποίο φορτώνεται και αποσυμπιέζεται κατά τα πρώιμα στάδια του boot, σε μια διαδικασία με χαμηλό throughput, λόγω του περιορισμένου αρχικού περιβάλλοντος. Έτσι, όταν στην περίπτωση του ramfs, το εν λόγω ELF object είναι έως και μία τάξη μεγέθους μεγαλύτερο από τα υπόλοιπα, αυτό είναι υπαίτιο για την αύξηση του χρόνου εκκίνησης. Βέβαια, εν προκειμένω έχουμε κάνει κατάχρηση του ramfs, το οποίο δεν είναι προορισμένο για τόσο μεγάλα images.

4.4 Application benchmark

4.4.1 Περιγραφή

Για να αξιολογήσουμε το virtio-fs με όρους μιας ολοκληρωμένης, πραγματικής εφαρμογής επιλέξαμε και πάλι από το πεδίο των stateless εφαρμογών σε πλαίσιο cloud το σενάριο ενός στατικού web εξυπηρετητή (server). Συγκεκριμένα, χρησιμοποιήσαμε τον nginx (στην έκδοση του 1.19.2), έναν από τους πλέον δημοφιλείς web servers ελεύθερου λογισμικού, ο οποίος είναι επίσης διαθέσιμος στη συλλογή εφαρμογών του OSv.[4]

Στις δοκιμές συμμετείχαν τα όλα τα συστήματα αρχείων του OSv (στην περίπτωση του virtio-fs, ως root file system χρησιμοποιήθηκε το ramfs) *εκτός από το NFS*, με το οποίο το σενάριο μας αποτύγχανε. Συγκεκριμένα, κατά την εξυπηρέτηση του πρώτου αιτήματος (request) από τον server, μετά την αποστολή λίγων αρχικών δεδομένων της απάντησης (response), η διαδικασία σταματούσε και ο guest φαινόταν 'παγωμένος'. Αυτό διαπιστώθηκε ότι δεν οφείλονταν στον nginx, καθώς την ίδια ακριβώς συμπεριφορά επιδείκνυε το σύστημα με τον lighttpd (επίσης περιλαμβάνεται στη συλλογή του 'osv-apps') στη θέση του. Περαιτέρω διερεύνηση απαιτείται για να εντοπιστεί και πιθανώς να διορθωθεί η αιτία, η οποία από την εμπειρία μας φαίνεται να μοιάζει με κάποιου είδους 'αδιέξοδο' που πιθανώς να εντοπίζεται στην υλοποίηση του NFS ή της στοίβας δικτύωσης του OSv.

Τα αρχεία τα οποία εξέθετε ο web server ήταν συνολικά 10, με μέγεθος που κυμαίνονταν από περίπου 500 KiB μέχρι και 12 MiB. Το μέγεθος κάθε αντίστοιχου αρχείου ήταν σταθερό σε όλες τις δοκιμές.

Για την διεξαγωγή των δοκιμών, που είχαν τη μορφή δοκιμών φορτίου HTTP (HTTP load tests), από τη μεριά του πελάτη (HTTP client) χρησιμοποιήσαμε το

[4] Όλες οι αλλαγές που έγιναν για τις δοκιμές μας βρίσκονται στο 'virtiofs-tests' branch του git repository `https://github.com/foxeng/osv-apps`.

vegeta, ένα δημοφιλές εργαλείο γι΄ αυτό το σκοπό, στην έκδοση του 12.8.3. Η διαδικασία (αυτοματοποιημένη από το script ενορχήστρωσης) είχε ως εξής:

1. Αρχικά εκκινούνταν ο OSv guest, στον οποίο δίνονταν ένα δευτερόλεπτο περιθώριο προκειμένου να ολοκληρωθεί η αρχικοποίηση του nginx server).

2. Στον host εκκινούσε το vegeta, ρυθμισμένο να παράγει HTTP 1.1 GET requests για όλα τα αρχεία του server, με το μέγιστο δυνατό ρυθμό, χρησιμοποιώντας 20 'εργάτες' και έως 4 CPUs, με την επαναχρησιμοποίηση (keepalive) των συνδέσεων TCP ενεργοποιημένη.

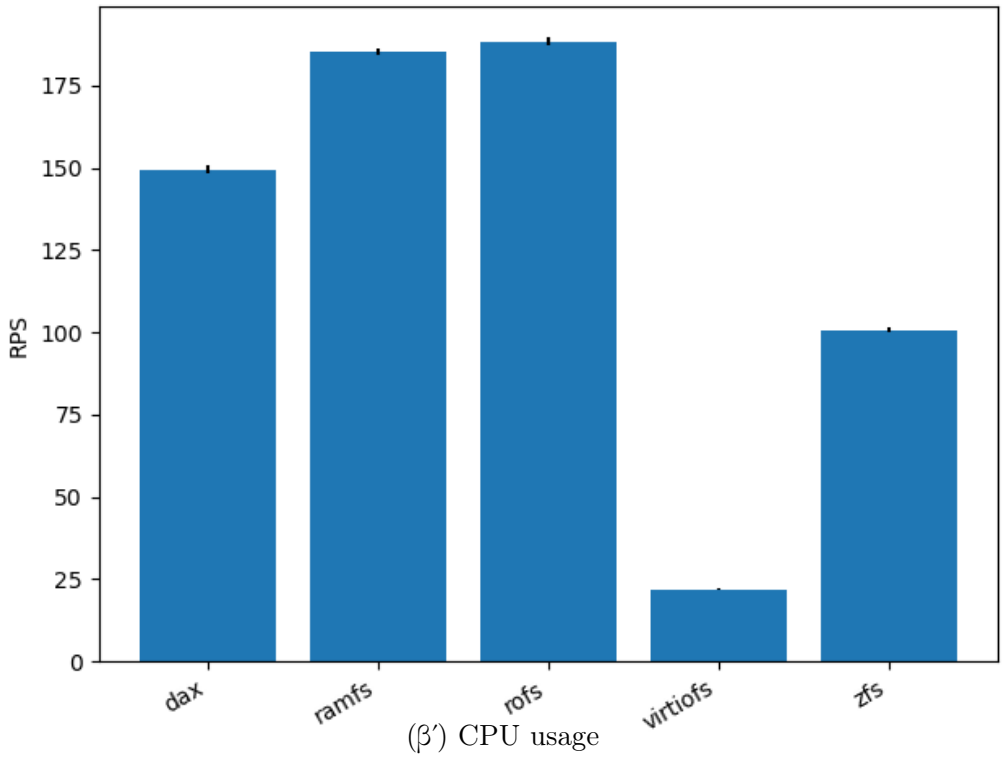
3. Μετά από δέκα δευτερόλεπτα, το vegeta ολοκλήρωνε το έργο του και τερματιζόταν, οπότε και ο αυτοματισμός τερμάτιζε τον guest μέσω της διεργασίας του QEMU.

4.4.2 Αποτελέσματα

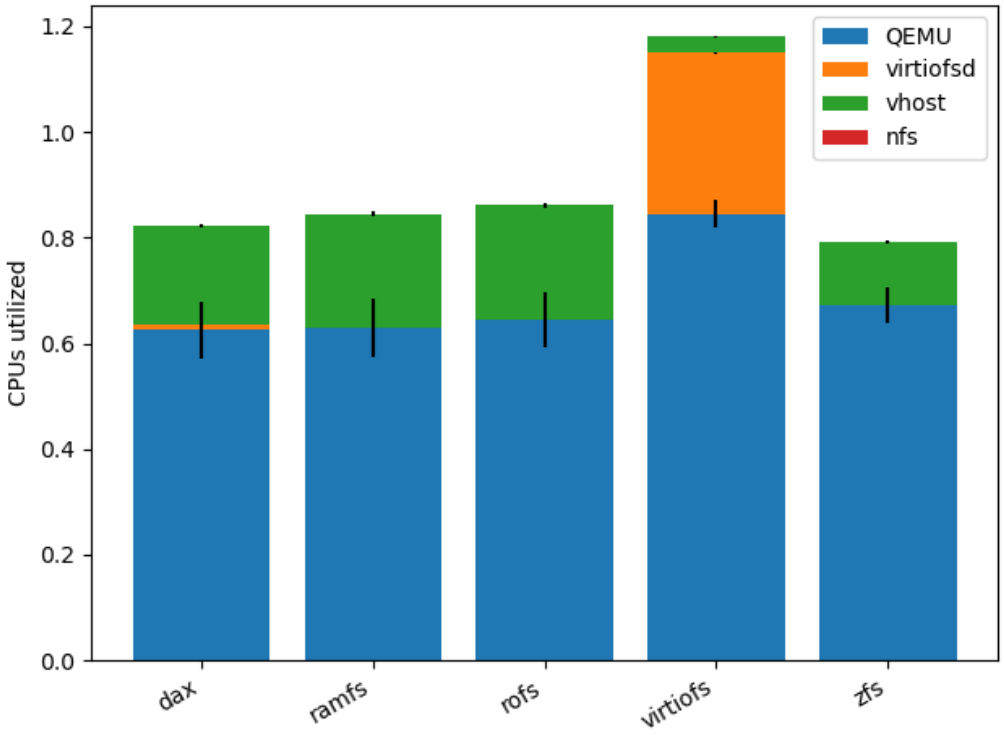
Όπως βλέπουμε στο σχήμα 4.6α΄, το virtio-fs με DAX window προσφέρει throughput (σε όρους εξυπηρετούμενων αιτημάτων ανά δευτερόλεπτο) που υπολείπεται αυτού των rofs και ramfs (τα οποία προηγούνται) κατά $\sim 20\%$. Δεδομένου ότι το CPU usage είναι οριακά χαμηλότερο από των άλλων δύο, αυτή η διαφορά χρήζει περαιτέρω μελλοντικής διερεύνησης (μέσω profiling), με εστίαση στην υλοποίηση του virtio-fs με DAX read datapath στο OSv, ως κύριο ύποπτο.

Όπως είναι αναμενόμενο σε αυτό το σενάριο χρήσης, το virtio-fs χωρίς DAX υστερεί με διαφορά έναντι όλων των υπόλοιπων συστημάτων αρχείων. Αυτό διότι είναι το μόνο που δεν χρησιμοποιεί οποιασδήποτε μορφής κρυφή μνήμη (cache) και κάθε λειτουργία ανάγνωσης συνεπάγεται έξοδο και επεξεργασία στον host (όπως φαίνεται και από το υψηλό CPU usage), ενώ αυτές οι λειτουργίες είναι πολύ συχνές, πολλαπλασιάζοντας τις συνέπειες αυτού του overhead.

(α′) Requests ανά δευτερόλεπτο



(β′) CPU usage



Σχήμα 4.6: nginx HTTP load test

Κεφάλαιο 5

Συμπεράσματα και επεκτάσεις