ΜΥΕ029
Προσομοίωση και Μοντελοποίηση Υπολογιστικών Συστημάτων

Διδάσκων: Γεώργιος Καππές

Αναφορά 1ης Εργαστηριακής Άσκησης:

Ανάλυση της απόδοσης του συστήματος αρχείων EXT4
σε περιβάλλον εικονικοποίησης

Ομάδα:

Κονδυλία Βέργου, ΑΜ 4325

Παναγιώτης Βουζαλής, ΑΜ 2653

Εαρινό Εξάμηνο 2022

Περιεχόμενα

Εισαγωγή

Σε αυτήν την εργαστηριακή άσκηση ασχοληθήκαμε με την ανάλυση της απόδοσης του ημερολογιακού (journaled) συστήματος αρχείων EXT4 στο περιβάλλον εικονικοποίησης VMware Workstation Player 16. Ειδικότερα, συγκρίναμε την απόδοση 3 μεθόδων λειτουργίας του ημερολογίου που προσφέρει το EXT4, εξετάζοντας μία εικονική μηχανή που προορίζεται να εκτελείται σε περιβάλλον εξυπηρέτησης δυναμικών ιστοσελίδων.

Οι 3 μέθοδοι λειτουργίας που τέθηκαν υπό εξέταση ηταν οι εξής:

1. ext4 - journal
2. ext4 - ordered
3. ext4 - writeback

Για την σύγκριση αυτή χρησιμοποιήθηκε το πρόγραμμα benchmarking *filebench (github.com/filebench)*. Το πείραμα αποτελούνταν από εκτελέσεις (runs) με διάφορα workloads και modes λειτουργίας του ημερολογίου του ext4.

Host & VM system specifications

Η μελέτη έγινε στα παρακάτω συστήματα χρησιμοποιώντας τον root λογαριασμό (admin στην περίπτωση των Windows).

(Η εφαρμογή του VMware Player βρίσκεται εγκατεστημένη στον SSD και το VM υπό εξέταση βρίσκεται εγκατεστημένο στον HDD. Τα πειράματα έλαβαν χώρα όσο το host σύστημα είχε ανοιχτό μόνο τον Firefox με περίπου 2GB απαιτήσεων σε RAM)

Host system specifications

- CPU: AMD Ryzen 5 5600X 6core clocked @ 4.2GHz all-core
- RAM: Kingston DDR4 16GB 3600MHz
- SSD: Samsung 860 Evo 500GB
- HDD: Western Digital Black 1TB 7200rpm
- OS: Windows 10 Enterprise LTSC version 1809 build 17763.2867
- VMware Workstation Player 16

VM system specifications

- CPU: 1 core assigned
- RAM: 2GB assigned
- HDD: Western Digital Black 1TB 7200rpm
- OS: Debian 11
- Filebench 1.5-alpha3
- Disk partition μεγέθους 5.1GB στο οποίο έγινε η πειραματική μελέτη.

Quality of life improvements

- Ως workspace χρησιμοποιήσαμε το φάκελο /home/user/ του Debian.
- Τοποθετήσαμε τα workloads υπό εξέταση στο φάκελο /home/user/workloads.
- Το script για την εκτέλεση των πειραμάτων (autobench.py) και το filebench καλούνταν από το terminal μέσα από το path /home/user/ με τα κατάλληλα ορίσματα.
- Αλλάξαμε το shebang των scripts του φακέλου /root/scripts σε #!/bin/bash αντί #/bin/bash
- Το autobench.py έχει γραφεί με τέτοιο τρόπο ώστε να νομίζει πως εκτελείται πάντα στο path /home/user

Διαχείριση κρυφής μνήμης

Πριν γίνει οποιαδήποτε μελέτη, έγιναν οι παρακάτω τροποποιήσεις στην διαχείρηση κρυφής μνήμης του Debian:

- /proc/sys/vm/dirty_expire_centisecs:
  Αλλαγή της default τιμής των 30sec σε 3sec υπό το σκεπτικό πως τα runs του benchmark μας θα έχουν διάρκεια 30 ή 60sec, άρα θέλουμε να αναγκάσουμε το λειτουργικό να στείλει τα δεδομένα της κρυφής μνήμης στο δίσκο τουλάχιστον 10 (ή 20) φορές ανά run, μιας και ήταν σημαντικό να εξετάσουμε την απόδοση του δίσκου και όχι της μνήμης.
- /proc/sys/vm/dirty_writeback_centisecs: Αλλαγή της default τιμής των 5sec σε αυτή του 1sec όπως προστάζει η εκφώνηση.
- /proc/sys/vm/drop_caches: Πριν απο κάθε run του benchmark μας γράφονταν η τιμή 3 στο συγκεκριμένο αρχείο ώστε να καθαρίσουν οι κρυφές μνήμες του συστήματος αρχείων.

Αυτοματοποίηση των πειραμάτων με Python

Η Python βρίσκεται πάντα εκεί για εμάς για αυτό και εμείς με τη σειρά μας αξιοποιήσαμε τη βοήθειά της για να αυτοματοποιήσουμε τη διαδικασία των πειραμάτων.

(Παρακάτω παρουσιάζεται επιγραμματικά το scriptάκι μας χωρίς περαιτέρω εξήγηση ή σχολιασμό στον κώδικα. Προσπαθήσαμε να είναι όσο το δυνατόν self explanatory αν και μπορεί να το παρακάναμε λιγάκι όσον αφορά τα σχόλια.)

```
# Benchmark run time of 60 seconds means dirty_expire will be called at least 20 times
# Benchmark run time of 30 seconds means dirty_expire will be called at least 10 times
dirty_expire = 300 # centisecs
dirty_writeback = 100 # centisecs

def setcaches():
    # Clean filesystem caches
    print('Cleaned filesystem caches.')
    subprocess.run(['echo', '3'], stdout = open('/proc/sys/vm/drop_caches', 'wb'))
    # Set dirty_expire
    print('Set dirty_expire to %i seconds.' % (dirty_expire/100))
    try:
        with open('/proc/sys/vm/dirty_expire_centisecs', 'w') as file:
            file.write(str(dirty_expire))

    except FileNotFoundError as e:
        print('Cannot open dirty_expire_centisecs. Quitting...')
        sys.exit()

    # Set dirty_writeback
    print('Set dirty_writeback to %i seconds.' % (dirty_writeback/100))
    try:
        with open('/proc/sys/vm/dirty_writeback_centisecs', 'w') as file:
            file.write(str(dirty_writeback))

    except FileNotFoundError as e:
        print('Cannot open dirty_writeback_centisecs. Quitting...')
        sys.exit()
```

setcaches.py - Καλείται στην αρχή της πειραματικής διαδικασίας

```
def parse_cpustats(path):
def parse_memorystats(path):
def parse_diskstats(path):
def parse_filebenchstats(path):
def parse_perfdata(path):
def parse_perfstat(path):
```

parsefiles.py – Καλείται για την πρόσβαση στα παραχθέντα αρχεία αποτελεσμάτων

```
##### IMPORTANT NOTES #####
# start-disk.sh: Changed first line shebang to #!/bin/bash instead of #/bin/bash
# stop-disk.sh: Changed first line shebang to #!/bin/bash instead of #/bin/bash
# Redirected output of mpstat, iostat, vmstat, to directories in the same location as
autoexp.py
# Redirected output of sar to directory in the same location as autoexp.py
# Redirected output of perf record to directory in the same location as autoexp.py
# Redirected output of perf stat to directory in the same location as autoexp.py
# Redirected output of filebench to directory in the same location as autoexp.py

import subprocess
from datetime import datetime
import time
import sys
import os
import math
import setcaches
import parsefiles
def start_usage_monitoring(dirName, interval):
    print('\nStarting resource usage monitors...')

    # Open mpstat, vmstat, iostat, perf in background with Popen
    #subprocess.Popen(['mpstat', interval], stdout = open(dirName + '/mpstat.out', 'wb'))
    #subprocess.Popen(['iostat', '-d', interval], stdout = open(dirName + '/iostat.out', 'wb'))
    #subprocess.Popen(['vmstat', interval], stdout = open(dirName + '/vmstat.out', 'wb'))
    subprocess.Popen(['sar', interval], stdout = open(dirName + '/cpu_stats.txt', 'w'))
    subprocess.Popen(['sar', '-r', interval], stdout = open(dirName + '/memory_stats.txt',
'w'))
    subprocess.Popen(['sar', '-d', interval], stdout = open(dirName + '/disk_stats.txt', 'w'))
    #subprocess.Popen(['sar', '-B', interval], stdout = open(dirName + '/paging_stats.txt',
'w'))
    subprocess.Popen(['perf', 'record', '-o', './' + dirName + '/perf.data'])
    subprocess.Popen(['perf', 'stat', '-ddd', '-o', './' + dirName + '/perf_stat.txt'])

def execute_benchmark(dirName):
    print('\nExecuting stress test with workload %s on ext4 journal mode %s' % (workload_path,
ext4_dataMode))
    subprocess.run(['filebench', '-f', workload_path], stdout = open(dirName +
'/filebench_stats.txt', 'w'))

def stop_usage_monitoring():
    print('\nStopping resource usage monitors...\n')
    #subprocess.run(['pkill', '-2', 'mpstat'])
    #subprocess.run(['pkill', '-2', 'iostat'])
    #subprocess.run(['pkill', '-2', 'vmstat'])
    subprocess.run(['pkill', '-2', 'sar'])
    subprocess.run(['pkill', '-2', 'perf'])
    time.sleep(2) # Give some time to filebench perf in order to print the stderr on screen

def perf_data_to_txt(dirName):
```

```
def calculate_statistics():
def print_statistics():
def calculate_standard_deviation(operations_tput_avg, opsPerSec_tput_avg, reads_tput_avg,
writes_tput_avg):
def calculate_95conf_interval(sd_ops, sd_opsPerSec, sd_reads, sd_writes, operations_tput_avg,
opsPerSec_tput_avg, reads_tput_avg, writes_tput_avg):

alpha = 0.05 # 95% confidence interval
ext4_dataMode = 'ext4-journal' # ext4-journal, ext4-ordered, ext4-writeback

workload = 'webserver.f'
workload_path = './workloads/' + workload

polling_interval = '2' # Polling time in seconds for the usage monitors
print_detailed_run_statistics = False # Set to True to print each run's statistics in detail

runs = 0 # experiment repetitions
moreRunsNeeded = True # flag used to stop experiment

print('////////// Autobench started //////////\n')
setcaches.setcaches() # Clean caches, set dirty_expire to 5 sec, set dirty_writeback to 1 sec
print('\nExt4 Journal mode =', ext4_dataMode)
print('\nUsage monitors polling interval = %s seconds' % polling_interval)
print('\nExecuting experiments...\n')

# Make sure that /dev/sda3 is unmounted before running experiments

while (moreRunsNeeded == True) and (runs < 20):
    # Update number of runs

    # Clean filesystem caches

    # Start a clean filesystem

    # Create a directory for the output files of the run

    # Start resource usage monitors and write output to files

    # Run benchmark and write output to file

    # Stop resource usage monitors

    # Print perf.data to perf_data.txt

    # Update statistical metrics

    # cpu_usr_avg, cpu_sys_avg, cpu_iowait_avg, cpu_idle_avg
    # mem_avg, commit_avg

    # disk_read_tput_avg, disk_write_tput_avg
```

```
    # filebench stats

    # perf.data

    # perf_stat

    # Update result metrics
    # total_tput_avg
    # read_tput_avg
    # write_tput_avg
    # ...

    # Metric chosen: operations
    print('\nOperation-centric throughputs across %i runs:' % runs)
    operations_tput_avg = total_ops / runs
    opsPerSec_tput_avg = total_ops_per_sec / runs
    reads_tput_avg = total_reads / runs
    writes_tput_avg = total_writes / runs

    # Choose one metric and calculate
    # 1 - Standard deviation
    # 2 - 95% confidence interval

    print('\nStandard deviations across %i runs:' % runs)
    print('Total ops standard deviation: %.2f' % sd_ops)
    print('Ops/sec standard deviation: %.2f' % sd_opsPerSec)
    print('Reads standard deviation: %.2f' % sd_reads)
    print('Writes standard deviation: %.2f' % sd_writes)

    # Check if more runs are needed
    # if 95_conf_interval / avg_of_chosen_metric < alpha: moreRunsNeeded = False

    if runs > 1:
        # Check if more runs are needed
        #if conf95_ops / operations_tput_avg < alpha: moreRunsNeeded = False
        if conf95_opsPerSec / opsPerSec_tput_avg < alpha: moreRunsNeeded = False
        #if conf95_reads / reads_tput_avg < alpha: moreRunsNeeded = False
        #if conf95_writes / writes_tput_avg < alpha: moreRunsNeeded = False

    # Stop the filesystem

    # Be a helpful little script

# Print statistics and results
```

autobench.py – Αυτοματοποιεί την πειραματική διαδικασία

Ο δρόμος για την επιλογή κατάλληλου workload

Το filebench προσφέρει πληθώρα απο τεχνητά φορτία εργασίας. Σκεπτόμενοι πως εξετάζουμε έναν web-server δυναμικών ιστοσελίδων, 3 ήταν τα βασικά workloads που δοκιμάσαμε. Αυτά ηταν το *oltp* που προσομοιώνει μια βάση δεδομένων βασισμένη στο μοντέλο E/E Oraclie 9i, το *videoserver* που προσομοιώνει έναν εξυπηρετητή video, και το *webserver* που προσομοιώνει εναν εξυπηρετητή ιστοσελίδων. Άλλα workloads που πέρασαν από το μυαλό μας αλλα δεν εξετάστηκαν καθόλου ήταν τα *fileserver, varmail,* και *webproxy.*

Runtime και Activity monitors

Όλα τα workloads εξετάστηκαν με runtime κάθε run 30sec, η παρατήρηση του CPU έγινε με την εντολή $sar, η παρατήρηση της μνήμης με την εντολή $sar -r, και η παρατήρηση του δίσκου έγινε με την εντολή $sar -d με παράμετρο τα 2sec σαν polling interval.

Database: oltp.f workload

```
set $dir=/mnt
set $eventrate=0
set $runtime=30
set $iosize=2k
set $nshadows=200
set $ndbwriters=10
set $usermode=200000
set $filesize=200m # was 10m
set $nfiles=20 # was 10
```

```
set $memperthread=1m
set $workingset=0
set $cached=0
set $logfilesize=10m
set $nlogfiles=1
set $directio=0
```

Οι μόνες αλλαγές που έγιναν στο συγκεκριμένο workload ήταν να αλλαχτεί το $filesize σε 200m και το $nfiles σε 10 έτσι ώστε να δημιουργηθεί φόρτος περίπου 4GB στη διαμέριση 5GB που είχαμε στη διάθεση μας.

Ενδεικτικά παρουσιάζονται τα στατιστικά του filebench απο 1 run εξετάζοντας τα 3 modes του ημερολογίου του ext4 (χωρίς να γίνεται λόγος σε αυτό το σημείο για τις εμφανείς διαφορές μεταξύ τους):

- ext4-journal:

```
IO Summary: 833219 ops 27651.981 ops/s 14164/13351 rd/wr  53.0mb/s   0.8ms/op
```

  Καθόλη τη διάρκεια του run παρατηρήθηκε CPU utilization (user) μέχρι 80% (value when idle 0%), RAM utilization memused μέχρι 25% (value when idle 25%) και commit μέχρι 125% (value when idle 55%). Το utilization του δίσκου έφτασε μέχρι 90% κατά τη διάρκεια δημιουργίας του fileset απο το filebench αλλά κατά τη διάρκεια του benchmark πάνω σε αυτό το fileset η χρήση του δεν ξεπέρασε το 45%.

- ext4-ordered:

```
IO Summary: 614418 ops 20401.741 ops/s 10551/9751 rd/wr  38.7mb/s   0.8ms/op
```

(μείωση 26% στα ops σε σχέση με το ext4-journal)

Στη διάρκεια δημιουργίας του fileset απο το filebench παρατηρήθηκε CPU utilization (user) μέχρι 90% (value when idle 0%), RAM utilization memused μέχρι 30% (value when idle 25%) και commit μέχρι 125% (value when idle 55%). Το utilization του δίσκου έφτασε μέχρι 100% κατά τη διάρκεια δημιουργίας του fileset απο το filebench. Κατά τη διάρκεια του benchmark πάνω σε αυτό το fileset η χρήση του δίσκου δεν ξεπέρασε το 65%, τα επίπεδα χρήσης της RAM παρέμειναν τα ίδια, και η χρήση του CPU μοιράστηκε σχεδόν ισομερώς μεταξυ user και system.

- ext4-writeback:

```
IO Summary: 611208 ops 20275.915 ops/s 10484/9692 rd/wr  38.4mb/s   0.9ms/op
```

(μείωση 26% στα ops σε σχέση με το ext4-journal)

Στη διάρκεια δημιουργίας του fileset απο το filebench παρατηρήθηκε CPU utilization (user) μέχρι 90% (value when idle 0%), RAM utilization memused μέχρι 30% (value when idle 25%) και commit μέχρι 125% (value when idle 55%). Το utilization του δίσκου έφτασε μέχρι 100% κατά τη διάρκεια δημιουργίας του fileset απο το filebench. Κατά τη διάρκεια του benchmark πάνω σε αυτό το fileset η χρήση του δίσκου δεν ξεπέρασε το 65%, τα επίπεδα χρήσης της RAM παρέμειναν τα ίδια, και η χρήση του CPU μοιράστηκε σχεδόν ισομερώς μεταξυ user και system.

Videoserver: videoserver.f workload

```
set $dir=/mnt
set $filesize=200m # was 10g
set $nthreads=48
set $eventrate=96
set $numactivevids=10 # was 32
```

```
set $numpassivevids=10 # was 194
set $reuseit=false
set $readiosize=256k
set $writeiosize=1m
```

Οι μόνες αλλαγές που έγιναν στο συγκεκριμένο workload ήταν να αλλαχτεί το $filesize σε 200m και τα $numactivevids/$numpassivevids σε 10 έτσι ώστε να δημιουργηθεί φόρτος περίπου 4GB στη διαμέριση 5GB που είχαμε στη διάθεση μας.

Ενδεικτικά παρουσιάζονται τα στατιστικά του filebench απο 1 run εξετάζοντας τα 3 modes του ημερολογίου του ext4:

- ext4-journal:

  ```
  IO Summary: 11596 ops 386.364 ops/s 386/0 rd/wr 116.1mb/s   0.3ms/op
  ```

  Στη διάρκεια δημιουργίας του fileset το CPU utilization ηταν μοιρασμένο μεταξύ system και iowait, με το δευτερο να λαμβάνει σταθερά περίπου 70%. Η χρήση του CPU ηταν μηδενική κατά τη διάρκεια του benchmark πάνω σε αυτό το fileset. Η χρήση της RAM δεν ξεπέρασε το 45% memused και 88% commit καθόλη τη διάρκεια του run. Ο δίσκος χρησιμοποιήθηκε μέχρι 95% μόνο κατά τη δημιουργία του fileset και στο benchmark πάνω σε αυτό το fileset δε χρησιμοποιήθηκε καθόλου εκτός από μόνο 2 spikes στο 20% που τα θεωρούμε outliers και τα αγνοούμε.

- ext4-ordered:

  ```
  IO Summary: 11596 ops 386.223 ops/s 386/0 rd/wr 116.0mb/s   0.3ms/op
  ```

  Παρόμοια utilizations με αυτά του ext4-journal.

- ext4-writeback:

  ```
  IO Summary: 11596 ops 386.253 ops/s 386/0 rd/wr 116.0mb/s   0.3ms/op
  ```

  Παρόμοια utilizations με αυτά του ext4-journal.

Webserver: webserver.f workload

```
# Παράγοντες
set $filesize=cvar(type=cvar-gamma,parameters=mean:2170552;gamma:1.5,min=524288,max=31457280)
set $nthreads=1
# Σταθερές
set $nfiles=2400 # was 1000
set $meandirwidth=20
set $iosize=1m # also tried 4k, 16k, 64k (that lead to larger ms/op values)
set $meanappendsize=16k
define process name=filereader,instances=1
{
  thread name=filereaderthread,memsize=10m,instances=$nthreads
```

Στο συγκεκριμένο workload αλλάξαμε το $filesize σε μέγεθος *αντιπροσωπευτικό μιας σύγχρονης ιστοσελίδας* και τον αριθμό των νημάτων που χρησιμοποιεί κάθε instance του filereader σε 1 σε μια προσπάθεια να πετύχουμε περισσότερα operations κατά τη διάρκεια κάθε run. Το I/O size αφέθηκε στη default τιμή του καθώς με μικρότερες τιμές επιτυγχάνονταν λιγότερα operations. Αλλαγή έγινε επίσης στον αριθμό των αρχείων ώστε να δημιουργηθεί φόρτος περίπου 4.8GB στη διαμέριση 5GB που είχαμε στη διάθεση μας.

Το κατάλληλο workload

Το workload oltp.f φαίνεται πολλά υποσχόμενο καθώς από ένα run και μόνο βλέπουμε διαφορές μεταξύ των 3 διαφορετικών mode λειτουργίας του ημερολογίου.

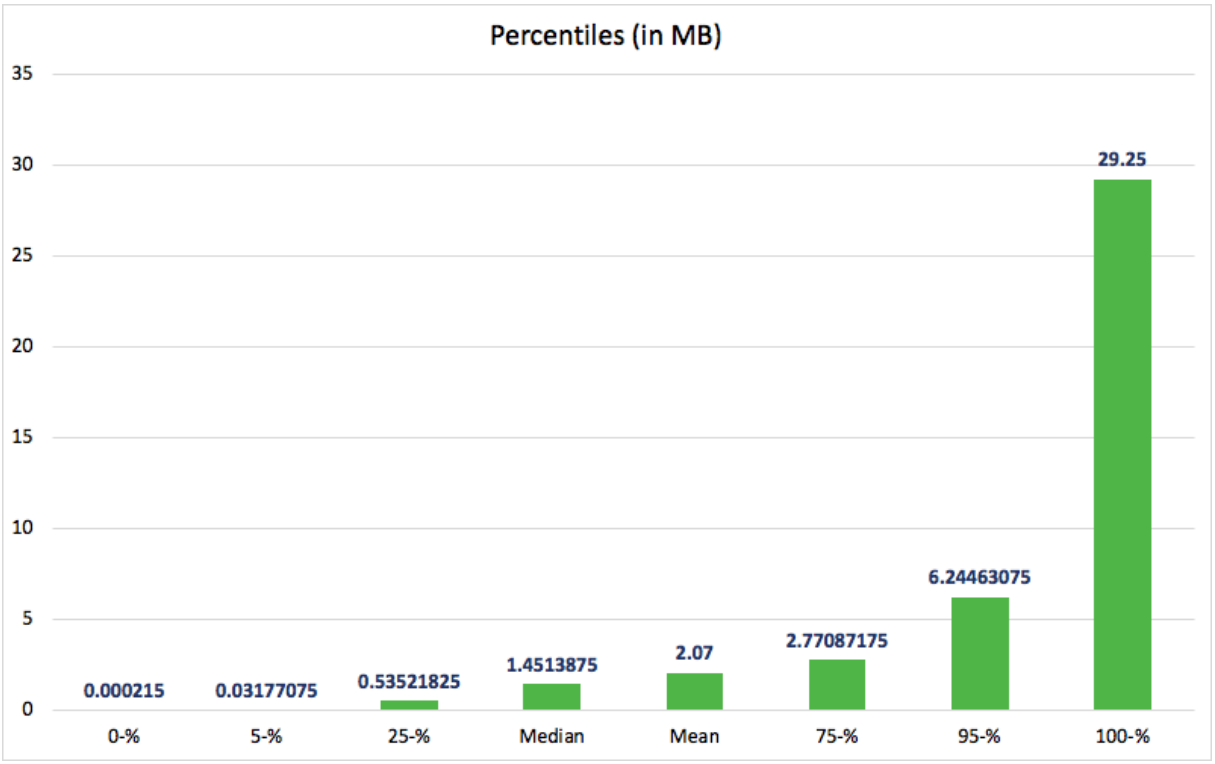
Προς το παρόν όμως, και ίσως περισσότερο λόγω του ονόματος, θα επικεντρωθούμε στο workload webserver.f και εαν δεν μας αρέσουν τα αποτελέσματα ή δεν μπορούμε να εξάγουμε κάποιο συμπέρασμα τότε θα επιστρέψουμε στο workload oltp.f.

Το workload videoserver.f μας απογοήτευσε καθώς δεν πετύχαινε το stress που επιθυμούμε στο δίσκο και αγνοήθηκε.

Webserver: Παράγοντες - Τι μέγεθος έχει μια σύγχρονη ιστοσελίδα?

Κάποιες ιστοσελίδες φορτώνουν ταχύτατα, άλλες “σέρνονται”. Κάποιες είναι lightweight, κάποιες τρέχουν τόσα scripts με το που τις επισκεφθείς που νομίζεις πως πρόκειται για cryptominers. Με αυτή τη σκέψη λοιπόν στο νου κάναμε μια αναζήτηση στο google που δεν είχαμε κάνει ποτέ μέχρι τώρα: “Πόσο μεγάλη είναι μια σύγχρονη ιστοσελίδα δυναμικού περιεχομένου?”

Για καλή μας τύχη, στο πρώτο αποτέλεσμα βρήκαμε μια ενδιαφέρουσα έρευνα που πραγματεύεται ακριβώς αυτό το ερώτημα. Θα βασιστούμε σε αυτή λοιπόν για να ορίσουμε ακριβώς το μέγεθος των αρχείων που θα δημιουργηθούν από το filebench.



Percentile groups for top 1000 web page total size in Megabytes – results data sheet here

Συμβουλευόμενοι λοιπόν το παραπάνω γράφημα θέτουμε το μέσο μέγεθος αρχείων σε 2.07MBs, το ελάχιστο σε 0.5MBs, και το μέγιστο σε 30MBs. *(Οι μετατροπές σε bytes έγιναν με το εξής εργαλείο.)*

```
set $filesize=cvar(type=cvar-gamma,parameters=mean:2170552;gamma:1.5,min=524288,max=31457280)
```

Webserver: Παράγοντες - Έχει σημασία ο αριθμός των threads?

```
set $nthreads=1
define process name=filereader,instances=1
{
  thread name=filereaderthread,memsize=10m,instances=$nthreads
```

Η απάντηση σε αυτό το ερώτημα είναι πως ναι, ασφαλώς και έχει (περισσότερα threads per filereader instance αυξάνουν τη μετρική ms/op). Εμείς όμως λόγω απλότητας αλλά και σε μια αναζήτηση να μειώσουμε το χρόνο που έπαιρνε το benchmark για να εξυπηρετήσει κάθε operation, θέσαμε αυτήν την τιμή σε 1.

Στόχος μας ήταν να έρθουμε πιο κοντά στις απαιτήσεις ενός σύγχρονου εξυπηρετητή ιστοσελίδων, όπως φαίνεται και στις διαφάνειες του μαθήματος.

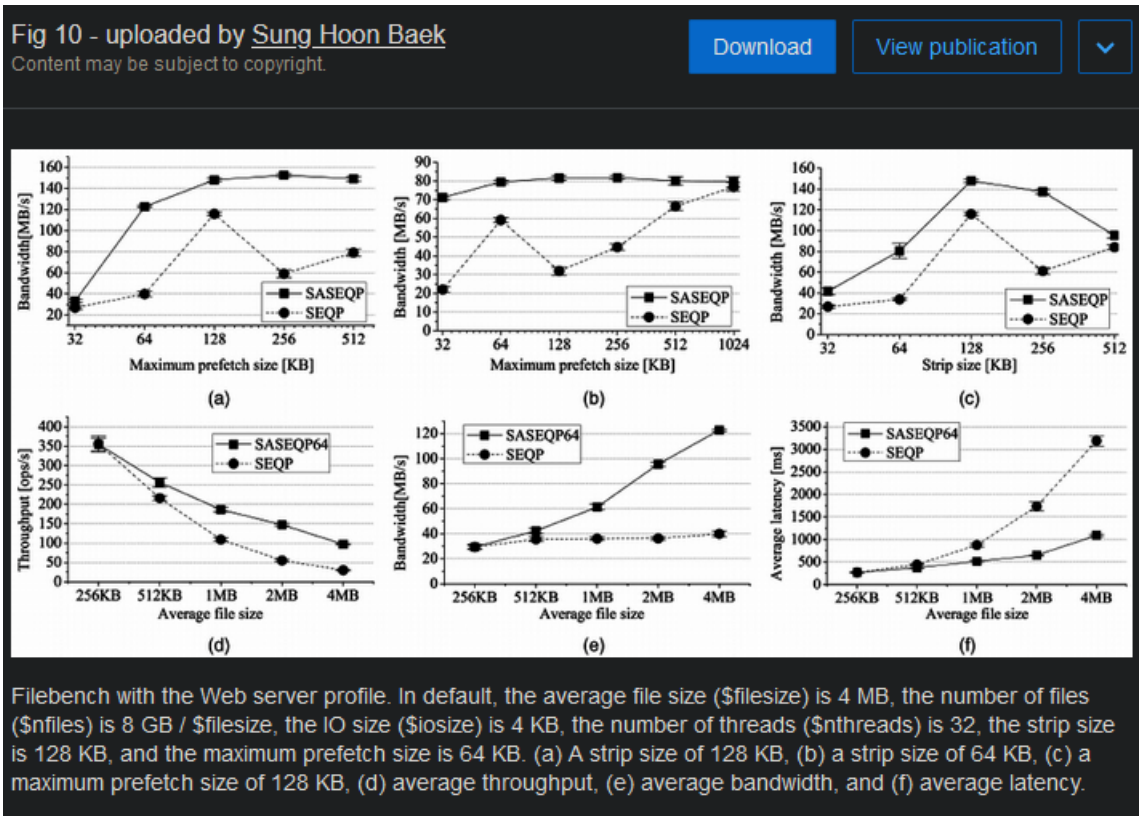
Με αυτό το σκεπτικό, πειράξαμε τα filereader_instances στο workload. Επιθυμία μας ήταν να δούμε πρώτον τι θα γίνει, και δεύτερον, πόσο κοντά μπορούμε να έρθουμε σε έναν πραγματικό εξυπηρετητή ιστοσελίδων που εξυπηρετεί εκατοντάδες και χιλιάδες πελάτες. Εμείς θεωρήσαμε πως κάθε filereader instance είναι και ένας πελάτης και κάναμε μετρήσεις για την τιμή του workload filereader_instances = 1 και filereader_instances = 100.

Απαιτήσεις εξυπηρετητή ιστοσελίδων

- Ο χρόνος απόκρισης πρέπει να είναι < 5 sec
- Η χρησιμοποίησή του πρέπει να είναι ~70%
- Υποστήριξη μέχρι και 1000 ταυτόχρονων πελατών με συνολικό ρυθμό 12.000 requests/sec

Webserver: Σταθερές

Στόχος μας ήταν να φτάσουμε τη διαμέριση των 5GB που είχαμε στη διάθεσή μας στα άκρα καθώς πειραματιζόμασταν με διάφορες τιμές.



Filebench with the Web server profile. In default, the average file size ($filesize) is 4 MB, the number of files ($nfiles) is 8 GB / $filesize, the IO size ($iosize) is 4 KB, the number of threads ($nthreads) is 32, the strip size is 128 KB, and the maximum prefetch size is 64 KB. (a) A strip size of 128 KB, (b) a strip size of 64 KB, (c) a maximum prefetch size of 128 KB, (d) average throughput, (e) average bandwidth, and (f) average latency.

https://www.researchgate.net/figure/Filebench-with-the-Web-server-profile-In-default-the-average-file-size-filesize-is-4_fig10_220330745

Workload	Dataset size	Files	R/W/M ratio	I/O Size
File-server	2.0GB	20,000	1/2/3	WF
Web-server	1.6GB	100,000	10/1/0	WF
DB-server	2.0GB	10	10/1/0	2KB
Mail-server	24.0GB	120	1/2/0	32KB

For the File-server and Web-server workloads, most of the writes happen in 4KB and 64KB I/O sizes. The 4KB read size is dominant in all workloads because this is the guest file system block size. However, many of the File-server's reads were merged into larger requests by the I/O scheduler and then later split into 64KB sizes by the NFS client. This happens because the average file size for the File-server is 128KB, so whole-file reads can be merged. For the Web-server workload, the average file size is only 16KB, so there are no 64KB reads at all. For the same reason, the Web-server workload exhibits many reads around 16KB (some files are slightly smaller, others are slightly larger, in accordance with Filebench's gamma distribution [47]). In-

https://www.usenix.net/system/files/conference/fast13/fast13-final84.pdf
Virtual Machine Workloads: The Case for New Benchmarks for NAS
Vasily Tarasov, Dean Hildebrand, Geoff Kuenning, Erez Zadok
Stony Brook University, IBM Research—Almaden, Harvey Mudd College

Συμβουλευόμενοι τα παραπάνω publications αλλά και τα δικά μας πειράματα, επιλέξαμε τις παρακάτω τιμές για τις σταθερές:

```
# Σταθερές
set $nfiles=2400 # was 1000
set $meandirwidth=20
set $iosize=1m # also tried 4k, 16k, 64k (that lead to larger ms/op values)
set $meanappendsize=16k
```

Webserver: Παρουσίαση αποτελεσμάτων για filereader_instances = 1

Και στα 3 modes οι κορυφαίες 2 διεργασίες σε κάθε run ήταν οι εξής:

```
Top 2 processes of each run:
~~~~~~~~~~~~~~
    48.10%  filebench       [kernel.kallsyms]          [k] copy_user_enhanced_fast_string
     9.76%  filebench       [kernel.kallsyms]          [k] clear_page_erms
~~~~~~~~~~~~~~
```

- ext4 - journal

```
Average CPU util percentage (user): 0.35 / Average CPU util percentage (sys): 58.70
Average CPU iowait percentage: 39.06 / Average CPU idle percentage: 1.89
Average memory usage percentage: 20.04 / Average memory commit percentage: 65.82
Average disk tranfers per second (tps): 1611.84 / Average disk read kBs: 497139.75
Average disk write kBs: 152985.01 / Average disk util percentage: 94.95
Average ops: 50159 / Average ops per sec: 1671.08 / Average reads: 539 / Average writes: 53
Average mb/s: 1095.08 / Average ms/op: 1.69
Average context switches: 82700 / Average page faults: 10827
Average L1 dcache misses percentage: 2.64 / Average L1 icache misses percentage: 1.49
```

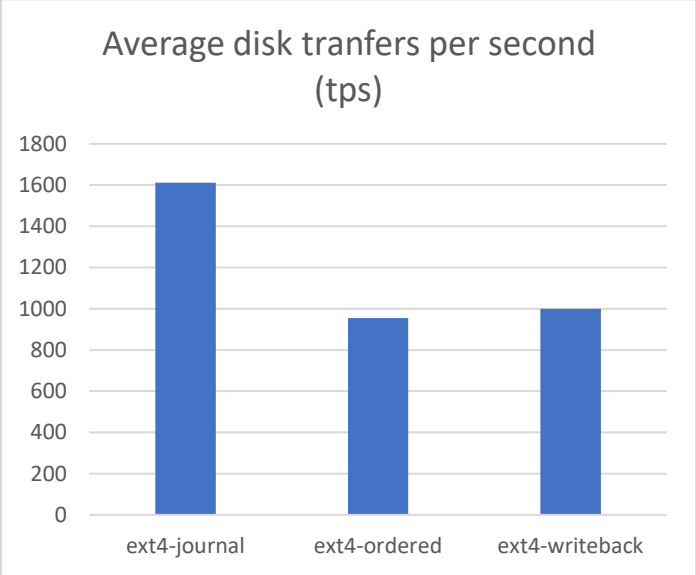
- ext4 - ordered

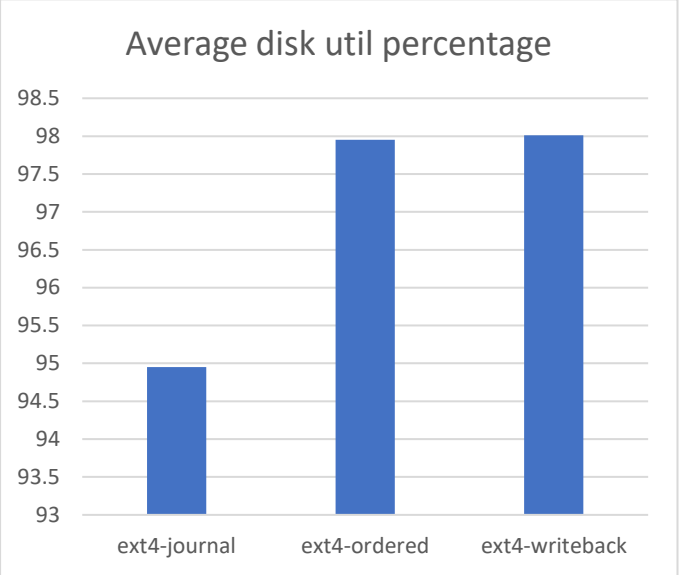
```
Average CPU util percentage (user): 0.74 / Average CPU util percentage (sys): 53.19
Average CPU iowait percentage: 44.88 / Average CPU idle percentage: 1.19
Average memory usage percentage: 20.70 / Average memory commit percentage: 70.26
Average disk tranfers per second (tps): 954.11 / Average disk read kBs: 535440.85
Average disk write kBs: 86418.76 / Average disk util percentage: 97.95
Average ops: 47930 / Average ops per sec: 1596.25 / Average reads: 515 / Average writes: 51
Average mb/s: 1046.09 / Average ms/op: 1.78
Average context switches: 78598 / Average page faults: 9384
Average L1 dcache misses percentage: 2.43 / Average L1 icache misses percentage: 1.19
```

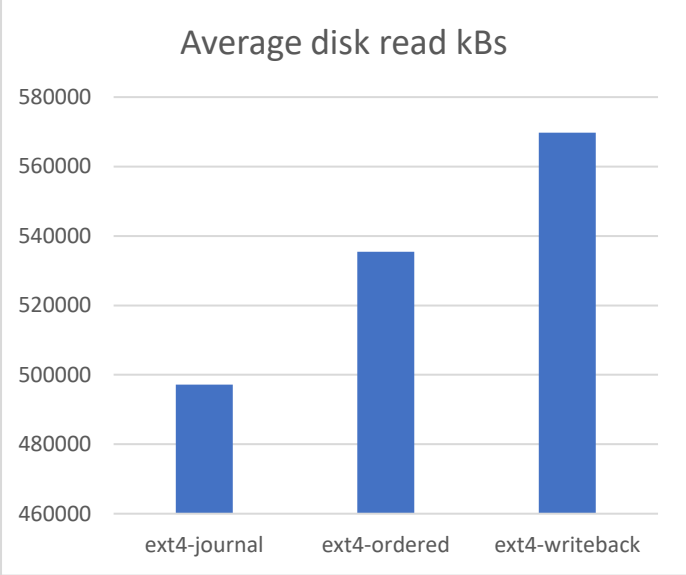
- ext4 - writeback

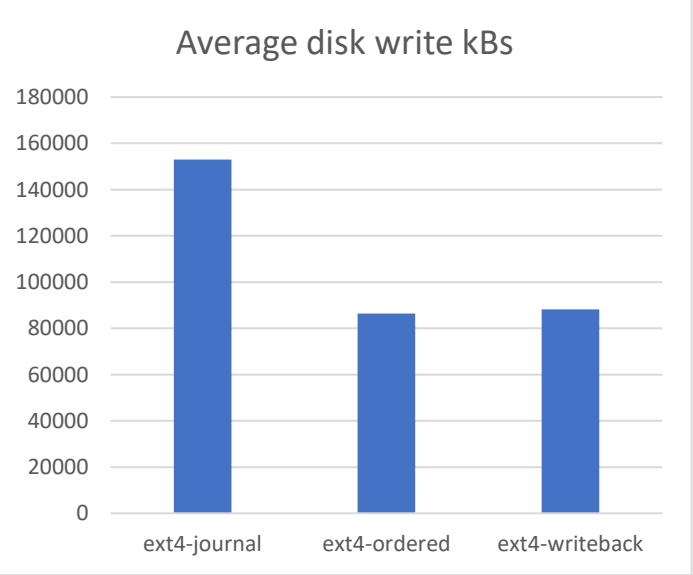
```
Average CPU util percentage (user): 0.35 / Average CPU util percentage (sys): 55.41
Average CPU iowait percentage: 42.74 / Average CPU idle percentage: 1.49
Average memory usage percentage: 19.99 / Average memory commit percentage: 70.82
Average disk tranfers per second (tps): 1000.21 / Average disk read kBs: 569774.79
Average disk write kBs: 88255.25 / Average disk util percentage: 98.01
Average ops: 50308 / Average ops per sec: 1676.31 / Average reads: 540 / Average writes: 54
Average mb/s: 1099.21 / Average ms/op: 1.70
Average context switches: 74082 / Average page faults: 6307
Average L1 dcache misses percentage: 2.40 / Average L1 icache misses percentage: 1.17
```

Webserver: Σύγκριση αποτελεσμάτων για filereader_instances = 1



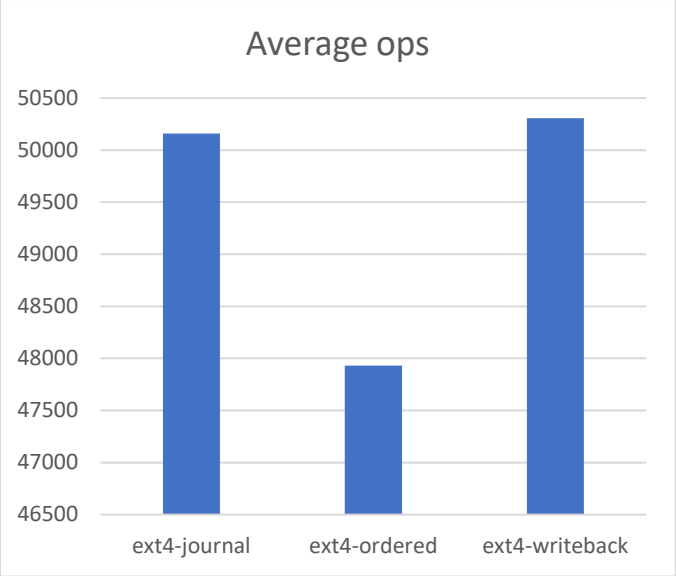


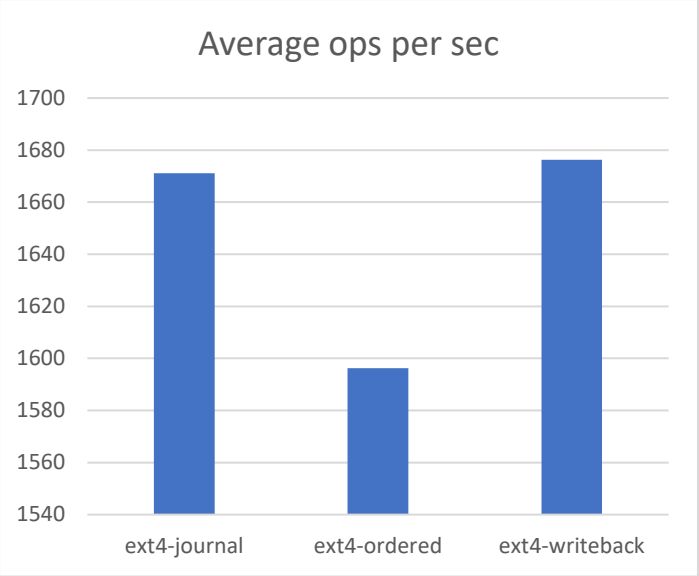


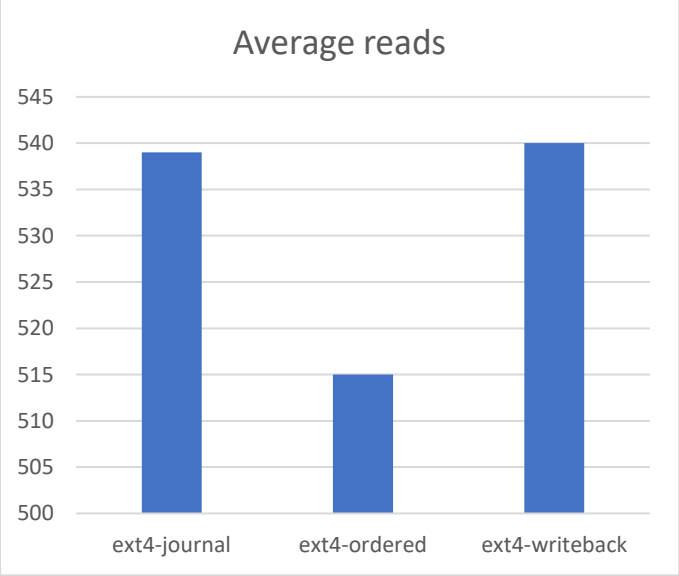


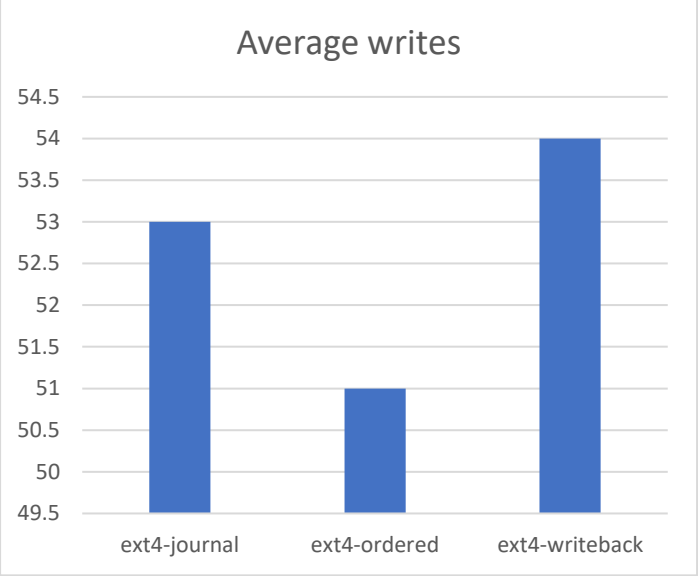
Παρατηρήσεις:

- Το ext4-journal εκτελεί περισσότερα transfers per second
- Το ext4-journal διαβάζει λιγότερα kBs απο το ext4-ordered το οποίο διαβάζει λιγότερα kBs από το το ext4-writeback
- Το ext4-journal γράφει περισσότερα κBs από τα ext4-ordered και ext4-writeback τα οποία γράφουν περίπου το ίδιο



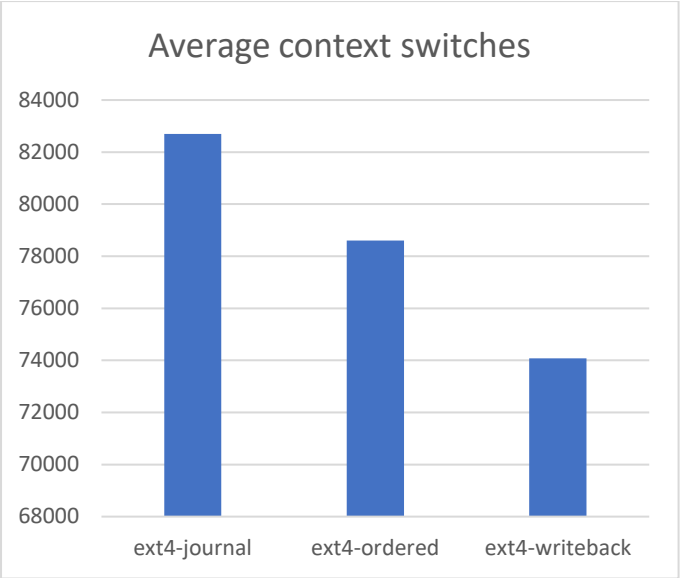


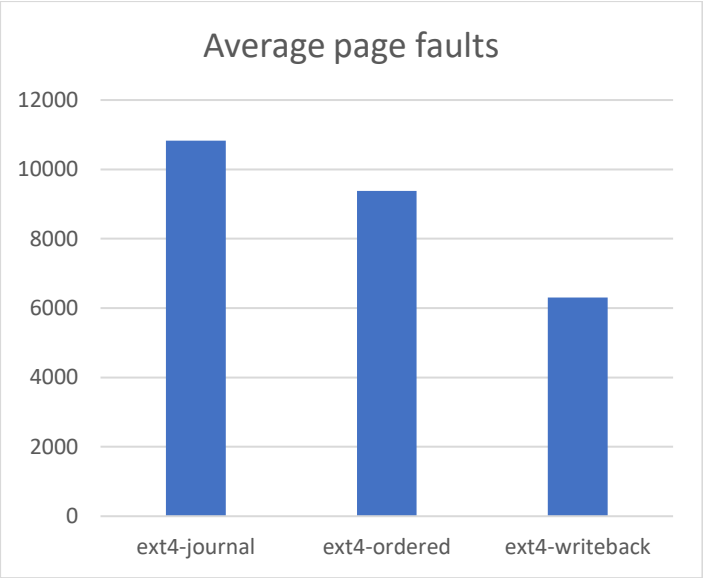




Παρατηρήσεις:

- Το ext4-ordered έχει αισθητά μειωμένο αριθμό operations και ρυθμό operations/sec. Αυτό αντικατοπτρίζεται και στον αριθμό reads και writes.





Παρατηρήσεις:

- Το ext4-journal έχει το μεγαλύτερο αριθμό context switches καθώς και page faults και το ext4-writeback το μικρότερο.

Webserver: Συμπεράσματα για filereader_instances = 1

Όσον αφορά το δίσκο, παρατηρούμε πως το ext4-journal εκτελεί τα περισσότερα transfers per sec και γράφει τα περισσότερα kBs στο δίσκο. Η συμπεριφορά αυτή είναι φυσιολογική και συνάδει με όσα ξέρουμε για το ext4-journal: Η καταγραφή κάθε λειτουργίας με τα data και τα metadata της στο ημερολόγιο επηρεάζει τη μετρική tps και write kBs. Αυτό γίνεται με ένα overhead της τάξης του 40% περίπου σε tps kai write kBs. Αυτό σημαίνει πως το ext4-journal θα στρεσσάρει περισσότερο το δίσκο από τα υπόλοιπα modes σε ένα δεδομένο χρονικό διάστημα, μειώνοντας πιθανώς τη διάρκεια ζωής του. Σκεπτόμενοι όμως την ταχύτητα των σημερινών δίσκων καθώς και την ασφάλεια που προσφέρει το ext4-journal σαν απλοί χρήστες, δεν υπάρχει λόγος να μην χρησιμοποιήσουμε το ext4-journal μόνο και μόνο για να κερδίσουμε λίγες μονάδες τοις εκατό πιο γρήγορο υπολογιστή. Στον enterprise χώρο, που το stability είναι αυτό για το οποίο πληρώνουν οι πελάτες, seems like a no brainer η χρήση του ext4-journal έναντι των άλλων modes. Φυσικά αυτό αλλάζει ανάλογα τις απαιτήσεις του κάθε πελάτη καθώς και το πόσο θέλει να ρισκάρει στο κυνήγι μεγαλύτερης απόδοσης.

Όσον αφορα τα average ops, average ops per sec, average reads, average writes, παρατηρούμε πως το ext4-journal έχει πάνω κάτω τα ίδια αποτελέσματα με το ext4-writeback (το οποίο θα περίμενε κανείς να είναι αυτό με το μεγαλύτερο αριθμό ops και ρυθμό ops/sec μιας που δεν προσφέρει κάποια εγγύηση σε περίπτωση κατάρρευσης). Αξιοσημείωτη είναι η μειωμένη απόδοση του ext4-ordered, η οποία πιστεύουμε πως οφείλεται στο γεγονός ότι το ext4-ordered εξασφαλίζει τη σωστή διάταξη των δεδομένων σε περίπτωση κατάρρευσης.

Όσον αφορα τα context switches και τα page faults, παρατηρούμε πως το ext4-journal έχει το μεγαλύτερο αριθμό context switches καθώς και page faults και το ext4-writeback το μικρότερο. Αυτή η συμπεριφορά θεωρούμε πως είναι φυσιολογική καθώς το ext4-journal θα πρέπει να προσπελάσει το δίσκο (με ο,τι συνεπάγεται αυτό) πολλές περισσότερες φορές από όσες θα το κάνει το ext4-writeback.

Για filereader_instances = 1 παρατηρήσαμε συμπεριφορά που συμφωνεί με τα όσα ξέρουμε για το ext4-journal.

Webserver: Παρουσίαση αποτελεσμάτων για filereader_instances = 100

Και στα 3 modes οι κορυφαίες 2 διεργασίες σε κάθε run ήταν οι εξής:

```
Top 2 processes of each run:
~~~~~~~~~~~~~~~
    27.19%  filebench       [kernel.kallsyms]           [k] count_shadow_nodes
    20.90%  filebench       [kernel.kallsyms]           [k] copy_user_enhanced_fast_string
~~~~~~~~~~~~~~~
```

- ext4 - journal

```
Average CPU util percentage (user): 0.56 / Average CPU util percentage (sys): 59.76
Average CPU iowait percentage: 36.51 / Average CPU idle percentage: 3.18
Average memory usage percentage: 39.31 / Average memory commit percentage: 89.00
Average disk tranfers per second (tps): 1529.85 / Average disk read kBs: 283373.61
Average disk write kBs: 148450.57 / Average disk util percentage: 92.44
Average ops: 29066 / Average ops per sec: 967.26 / Average reads: 312 / Average writes: 32
Average mb/s: 632.99 / Average ms/op: 286.43
Average context switches: 119987 / Average page faults: 307088
Average L1 dcache misses percentage: 5.82 / Average L1 icache misses percentage: 1.66
```

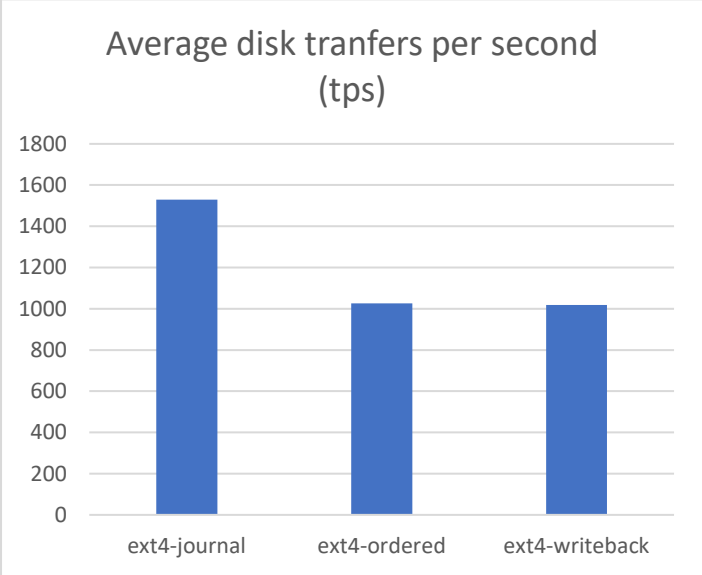
- ext4 - ordered

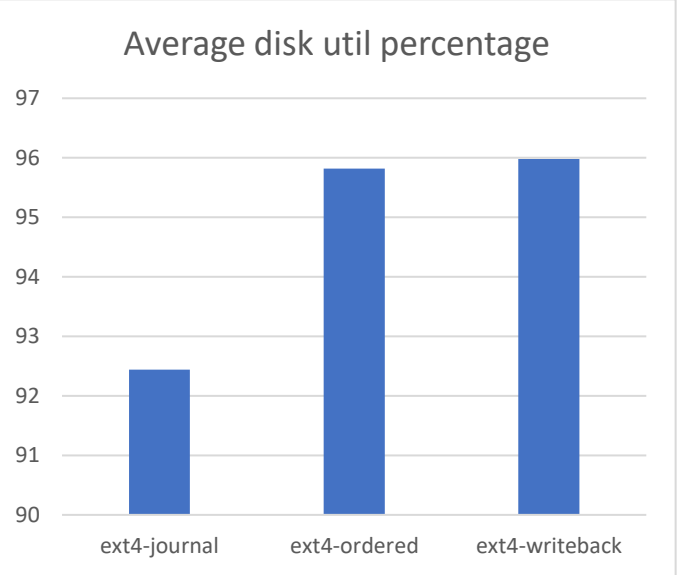
```
Average CPU util percentage (user): 0.54 / Average CPU util percentage (sys): 57.93
Average CPU iowait percentage: 38.77 / Average CPU idle percentage: 2.76
Average memory usage percentage: 44.00 / Average memory commit percentage: 105.42
Average disk tranfers per second (tps): 1026.07 / Average disk read kBs: 307796.58
Average disk write kBs: 86516.98 / Average disk util percentage: 95.82
Average ops: 27340 / Average ops per sec: 910.82 / Average reads: 293 / Average writes: 31
Average mb/s: 596.10 / Average ms/op: 304.34
Average context switches: 115452 / Average page faults: 300860
Average L1 dcache misses percentage: 4.13 / Average L1 icache misses percentage: 1.18
```

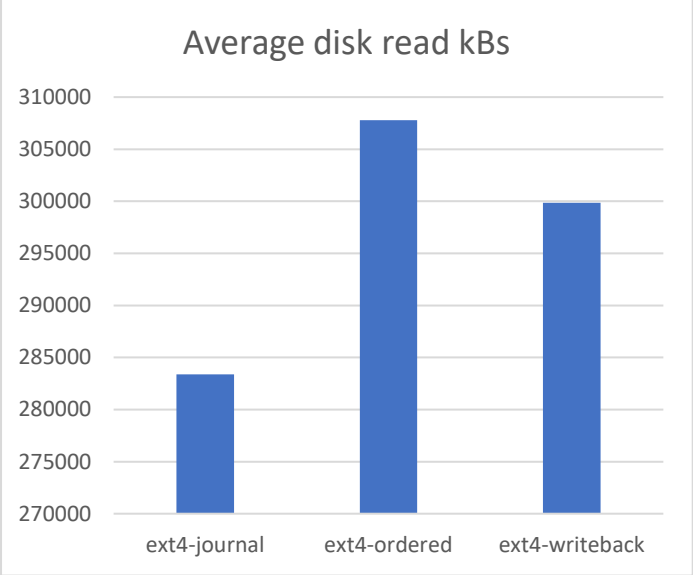
- ext4 - writeback

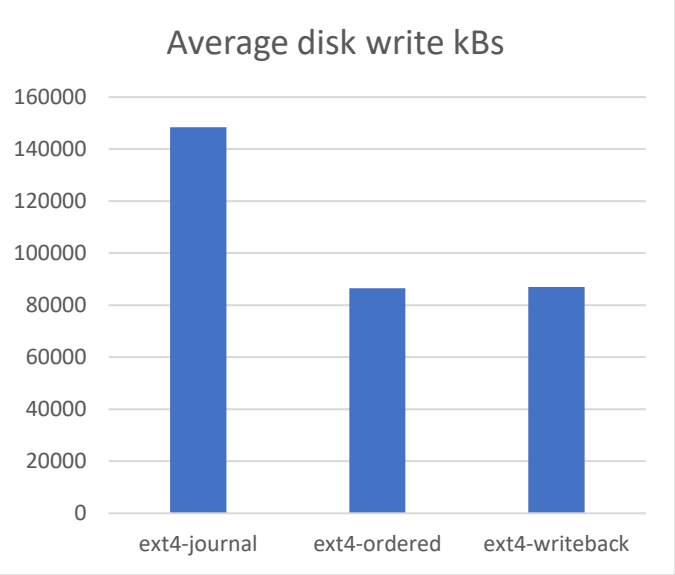
```
Average CPU util percentage (user): 0.57 / Average CPU util percentage (sys): 58.17
Average CPU iowait percentage: 38.72 / Average CPU idle percentage: 2.55
Average memory usage percentage: 45.36 / Average memory commit percentage: 105.75
Average disk tranfers per second (tps): 1017.72 / Average disk read kBs: 299859.59
Average disk write kBs: 87049.50 / Average disk util percentage: 95.98
Average ops: 26499 / Average ops per sec: 882.85 / Average reads: 284 / Average writes: 29
Average mb/s: 577.28 / Average ms/op: 313.11
Average context switches: 113477 / Average page faults: 303544
Average L1 dcache misses percentage: 4.15 / Average L1 icache misses percentage: 1.19
```

Webserver: Σύγκριση αποτελεσμάτων για filereader_instances = 100



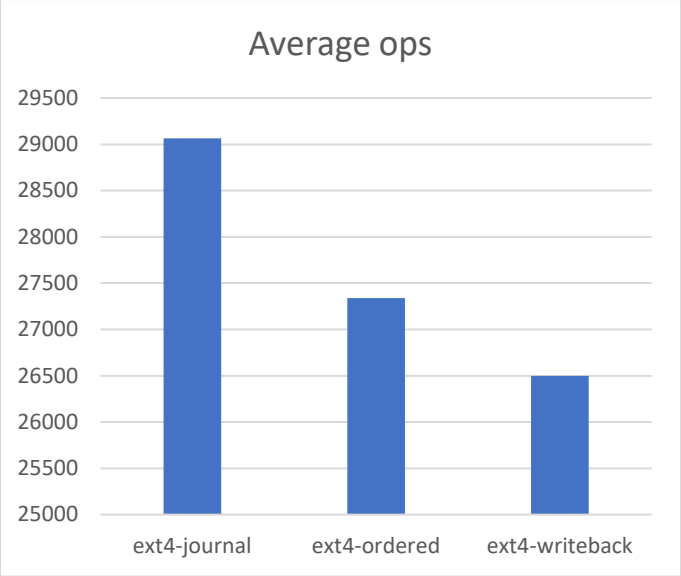


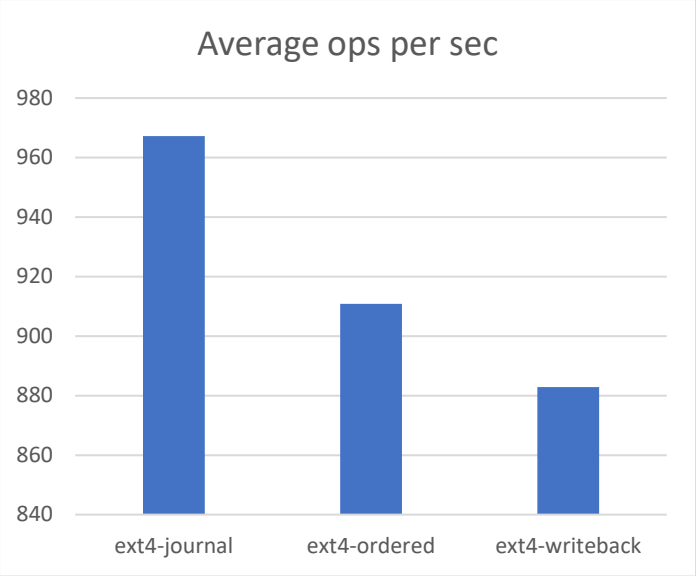


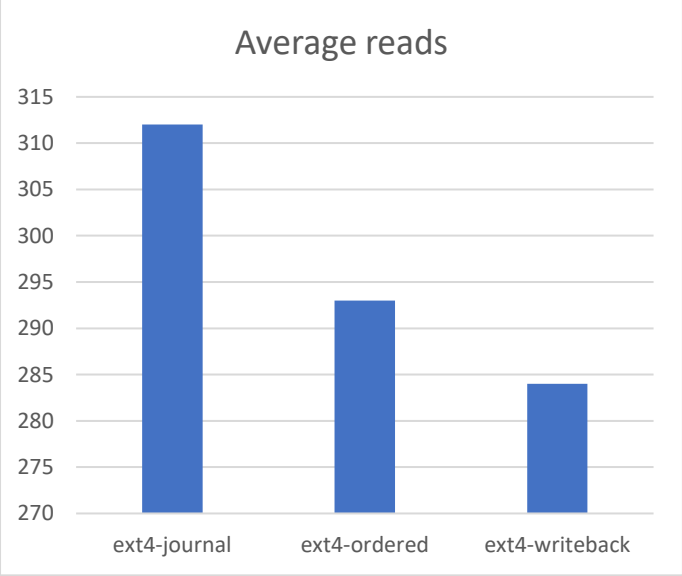


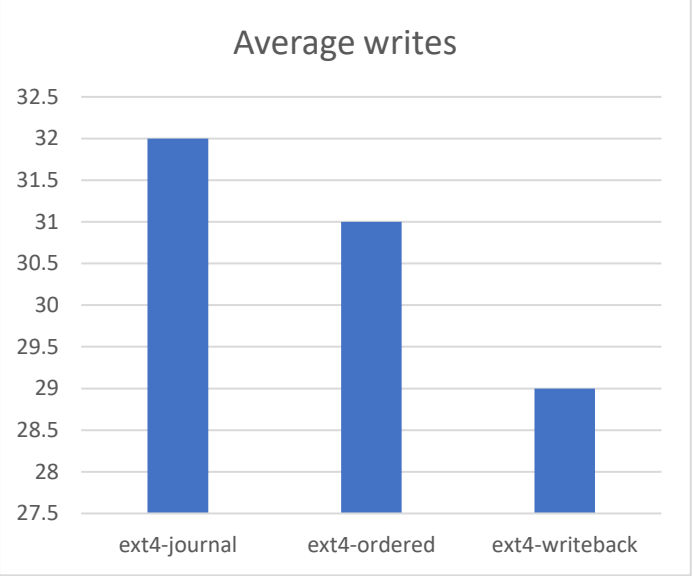
Παρατηρήσεις:

- Το ext4-journal εκτελεί σχεδόν 50% παραπάνω transfers per second από τα άλλα 2 modes ενώ παράλληλα έχει αισθητά μειωμένο συνολικό αριθμό read kBs και αισθητά αυξημένο συνολικό αριθμό write kBs, γεγονός που συνάδει με τις λειτουργίες που εκτελεί.



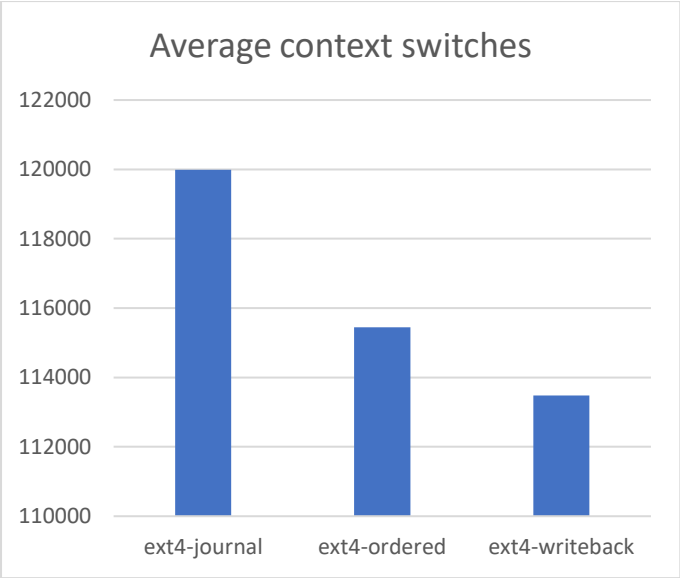


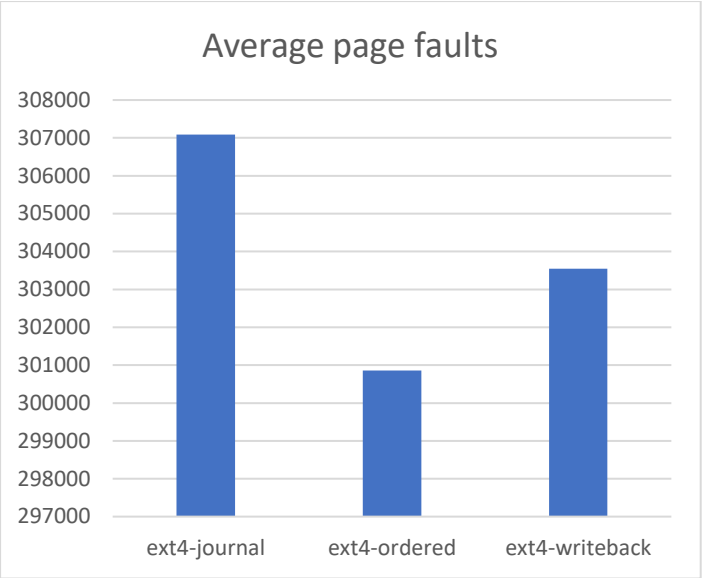




Παρατηρήσεις:

- Το ext4-journal εκτελεί παραπάνω operations από τα άλλα 2 modes και έχει το μεγαλύτερο ρυθμό ops per sec ενώ παράλληλα έχει αισθητά αυξημένο συνολικό αριθμό reads και writes, γεγονός που συνάδει με τις λειτουργίες που εκτελεί.





Παρατηρήσεις:

- Το ext4-journal κατά μέσο όρο έχει αισθητά αυξημένο αριθμό context switches και page faults, γεγονός που συνάδει με τις λειτουργίες που εκτελεί.

Webserver: Συμπεράσματα για filereader_instances = 100

Έχοντας θέσει τον αριθμό νημάτων σε 1 στην αρχή του πειράματος και τροποποιώντας το workload έτσι ώστε να ανοίγει 100 filereader instances αντί για 1, περιμένουμε πως θα υπάρχει διαφορά με τα όσα είδαμε για firereader_instances = 1, κυρίως στον αριθμό ops και ops per sec.

Όσον αφορά τον δίσκο και τα tranfers per sec, disk read kBs, disk writes kBs παρατηρούμε συμπεριφορά παρόμοια με αυτή που είδαμε για 1 instance filereader.

Όσον αφορα τα average ops, average ops per sec, average reads, average writes, μας εξέπληξε που είδαμε το ext4-writeback να έρχεται τελευταίο σε όλες τις μετρικές σύγκρισης. Εδώ περιμέναμε να το δούμε να έρχεται τουλάχιστον στα ίσα με το ext4-journal όπως είδαμε για 1 instance filereader.

Όσον αφορα τα context switches και τα page faults, περιμέναμε τα αποτελέσματα να είναι όμοια με αυτά που είδαμε για 1 instance filereader, και για το ext4-journal είναι. Όπως και για 1 instance filereader, το ext4-journal έχει το μεγαλύτερο αριθμό context switches και page faults εξ' αιτίας των περισσότερων προσπελάσεων που κάνει στο δίσκο. Αυτό που δεν περιμέναμε να δούμε είναι το ext4-writeback να έχει περισσότερα page faults από το ext4-ordered. Αυτό είναι κάτι που δεν ερευνήσαμε περισσότερο.

Database: Παράγοντες και σταθερές

TODO

Database: Παρουσίαση αποτελεσμάτων για filereader_instances = 1

TODO

Database: Σύγκριση αποτελεσμάτων για filereader_instances = 1

TODO

Database: Συμπεράσματα για filereader_instances = 1

TODO

Database: Παρουσίαση αποτελεσμάτων για filereader_instances = 100

TODO

Database: Σύγκριση αποτελεσμάτων για filereader_instances = 100

TODO

Database: Συμπεράσματα για filereader_instances = 100

TODO

Παραδοτέα αρχεία

turnin.zip/

- benchmark_results/webserver/filereader_instances_1/
  - ext4-journal/
    - benchmark_results.txt , run_1/ ~ run_20/
  - ext4-ordered/
    - benchmark_results.txt , run_1/ ~ run_20/
  - ext4-writeback/
    - benchmark_results.txt , run_1/ ~ run_20/
- benchmark_results/webserver/filereader_instances_100/
  - ext4-journal/
    - benchmark_results.txt , run_1 ~ run_20/
  - ext4-ordered/
    - benchmark_results.txt , run_1 ~ run_20/
  - ext4-writeback/
    - benchmark_results.txt , run_1 ~ run_20/
- workloads/
  - oltp.f, videoserver.f, webserver.f
- autobench.py, parsefiles.py, setcaches.py
- mye029-lab1.pdf
- report.pdf

Βιβλιογραφία

1. https://www.cs.uoi.gr/~gkappes/mye029
2. https://www.cse.uoi.gr/~gkappes/files/mye029/mye029-lecture1.pdf
3. https://www.cse.uoi.gr/~gkappes/files/mye029/mye029-lecture2.pdf
4. https://www.cse.uoi.gr/~gkappes/files/mye029/t-Dist1.pdf

5. https://github.com/filebench/filebench

6. https://www.pingdom.com/blog/webpages-are-getting-larger-every-year-and-heres-why-it-matters/
7. https://drive.google.com/file/d/1JqniM-gDXpgP6etdXesFoY_BD3l3bRxH/view

8. https://www.flightpedia.org/convert-megabytes-to-bytes.html

9. https://www.researchgate.net/figure/Filebench-with-the-Web-server-profile-In-default-the-average-file-size-filesize-is-4_fig10_220330745
10. https://www.usenix.net/system/files/conference/fast13/fast13-final84.pdf

11. https://www.linuxtechi.com/generate-cpu-memory-io-report-sar-command/
12. https://www.baeldung.com/linux/monitor-disk-io
13. https://stackoverflow.com/questions/10082517/simplest-tool-to-measure-c-program-cache-hit-miss-and-cpu-time-in-linux

14. https://www.statisticshowto.com/probability-and-statistics/confidence-interval/
15. https://www.scribbr.com/statistics/standard-deviation/
16. https://stackoverflow.com/questions/1174984/how-to-efficiently-calculate-a-running-standard-deviation
17. https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance#Welford's_online_algorithm