The Effect of Flashcache and Bcache on I/O Performance

Christopher Hollowell <hollowec@bnl.gov>
RHIC/ATLAS Computing Facility
Brookhaven National Laboratory

Co-authors: Jason Smith, Richard Hogue, Alexandr Zaytsev, William Strecker-Kellogg, Tony Wong

October 15, 2013

Introduction

- x86_64 server CPU core counts are continuing to increase
  - *Intel server processor logical core counts:*

Generation	***Name***	***Max Logical Cores Per Physical CPU***
Current	Sandy Bridge	16
Near Future	Ivy Bridge	24 (30 rumored with Ivy Bridge-EX)
Future	Haswell	16-20 physical cores expected = 32-40 logical cores

- Typical HEP/NP model: 1 batch job slot per logical core
  - More jobs per system have lead to a growing demand for local scratch and remote storage random I/O performance
  - Primarily interested in scratch storage performance for this study
- Solid state drives (SSDs) provide excellent random I/O performance when compared to traditional rotating drives
  - Also very expensive when compared to traditional SATA/SAS drives
    - 1 TB SSDs typically retail near $1,000 USD
      - Can be over $2,500 for enterprise drives from server vendors
      - Enterprise drives needed in high write volume environments
    - 1 TB SATA drives typically retail under $100

Introduction (Cont.)

- Increasing spindle count helps with random I/O:
  - Large software RAID0 arrays have excellent random I/O performance characteristics, and are well suited for local scratch use
  - Need to buy many traditional drives, which can also be fairly costly

- Hybrid SSD drives?
  - SSD/flash memory cache in front of traditional rotating media
  - Commodity hardware hybrid drives available from Seagate for reasonable prices
    - Unfortunately, not currently sold or supported by large server vendors such as Dell
  - Software defined hybrid SSD devices for Linux
    - Flashcache
    - Bcache

Flashcache

Linux kernel module which provides software devices with block caching on SSDs or other fast storage. Cache structure is a set associative hash

Uses Linux Device Mapper (DM) layer
- Devices appear as /dev/mapper/CACHENAME

Developed by Facebook in 2010

Not integrated into the upstream kernel source: need to compile separately
- ELRepo offers packages for SL/RHEL6
  - Haven't tried these
  - Built stable version 2.0 from source for our tests (2.1 recently out)
    - Builds against the SL/RHEL6 X86_64 2.6.32-358.6.2.el6 kernel without any issues

Supports writethrough, writearound and writeback caching
- FIFO and LRU cache replacement policies supported

Many tunables via sysctl interface

Building/Configuring Flashcache

Get the source

Release 2.1 now available:
https://github.com/facebook/flashcache/releases/tag/2.1

2.x wasn't available as a tarball when we started our tests, so we manually created one:

```
$ git clone https://github.com/facebook/flashcache.git
$ cd flashcache
$ git archive -o /tmp/fc2.tar.gz stable_v2 .
```

Install the SL/RHEL6 kernel-devel package

Build the flashcache kernel module and utilities against the kernel-devel headers:

```
$ tar -xzvf fc2.tar.gz
$ cd flashcache
$ make KERNEL_TREE=/usr/src/kernels/2.6.32-358.6.2.el6.x86_64
$ cd src
```

Building/Configuring Flashcache (Cont.)

Insert the kernel module:

```
# insmod ./flashcache.ko
```

Create and format the flashcache device (using sdb1 [SSD] and sda8):

```
# ./utils/flashcache_create -p back fc1 /dev/sdb1 /dev/sda8
```

If the Flashcache device already existed, one would instead issue:

```
# ./utils/flashcache_load /dev/sdb1
```

```
# mkfs.ext4 /dev/mapper/fc1
```

Check/modify systctl parameters (we used defaults):

```
# sysctl -a | grep flashcache
...
dev.flashcache.sdb1+sda8.reclaim_policy = 0
dev.flashcache.sdb1+sda8.fallow_delay = 900
dev.flashcache.sdb1+sda8.skip_seq_thresh_kb = 0
...
```

Bcache

Another Linux kernel module which provides software devices with block caching on SSDs or other fast storage. Utilizes a btree cache structure

Created devices appear as /dev/bcacheX

Packages not available for SL/RHEL6 in standard repositories (SL, EPEL, ELRepo, etc.)

Integrated as stable software into the upstream vanilla kernel starting in 3.10

- Integrated source cannot be easily extracted and compiled against the SL/RHEL6 kernel source
- Therefore, all testing performed with vanilla 3.11.1 kernel

Also provides writethrough, writearound and writeback caching

- Supports FIFO, LRU, and random cache replacement policies

Configuration via a sysfs (/sys) interface: many options available

Building/Configuring Bcache

Copy SL/RHEL6 kernel SRPM X86_64 config to .config in a vanilla kernel 3.11.1 tree, run make olddefconfig, and build/install (See “Backup Slide” for more information)

Edit /boot/grub/grub.conf to make 3.11.1 the default kernel and reboot

Load the module:

```
# modprobe bcache
```

Obtain and build bcache-tools source:

```
$ git clone http://evilpiepirate.org/git/bcache-tools.git
$ cd bcache-tools
$ make
```

Building/Configuring Bcache (Cont.)

Create bcache device (using sdb1 [SSD] and sda8):

```
# ./make-bcache -B /dev/sda8
# ./make-bcache -C /dev/sdb1
# echo /dev/sdb1 > /sys/fs/bcache/register
# echo /dev/sda8 > /sys/fs/bcache/register
# cd /sys/block/bcache0/bcache
# echo CACHESET_UUID_FROM_MAKE-BCACHE_CMD > attach
```

On reboot, it is only necessary to run the above echo commands to reassemble the device. Udev rules are also available which make manual registration unnecessary for assembly of pre-existing devices

Change default parameters and format the bcache device:

```
# cd /sys/block/bcache0/bcache
# echo writeback > cache_mode
# echo 0 > sequential_cutoff
# mkfs.ext4 /dev/bcache0
```

Evaluation Hardware/Configuration

Single SATA hard drive, SSD, Flashcache, and Bcache benchmarks

- Dell PowerEdge R410
- 2 6-core Xeon X5660@2.80 GHz CPUs (HT on: 24 logical cores total)
- 48 GB DDR3 1333 MHz RAM
- SAS 6/IR disk controller
- 64-bit Scientific Linux 6.4 (kernel 2.6.32-358.6.2.el6.x86_64, 3.11.1 for bcache)

1. Hard drive used during single SATA, Flashcache and Bcache benchmarks:
   - Seagate ST32000644NS 3.5" drive
     - 2 TB, SATA 3.0 Gbps
     - 64 MB cache
     - 7200 RPM
     - Firmware release KA06
2. SSD drive used during single SSD, Flashcache and Bcache benchmarks:
   - Dell MZ-5EA2000-0D3 (Samsung SM825) 2.5" Enterprise SSD
     - 200 GB, SATA 3.0 Gbps, eMLC
     - Firmware release 7D3Q
     - Kernel I/O scheduler set to "deadline" for all tests

Evaluation Hardware/Configuration (Cont.)

- Flashcache Configuration
  - Used defaults:
    - Writeback cache (set at device creation time)
    - Clean idle dirty blocks after 15 minutes without use. Inconsistent benchmark results with this parameter set to 0 (no idle cleaning)
      - fallow_delay = 900
    - FIFO cache reclaim policy – default
      - reclaim_policy = 0
    - Disable sequential cutoff
      - skip_seq_thresh_kb = 0
- Bcache Configuration
  - Used defaults, with a few exceptions:
    - Writeback caching enabled (writethrough the default)
      - cache_mode = “writeback”
    - Sequential cutoff disabled
      - sequential_cutoff = 0 (4 MB the default)
    - LRU cache reclaim policy – default
      - cache_replacement_policy = “lru”
  - Default initial writeback_delay of 30 seconds – somewhat different meaning than Flashcache's fallow_delay parameter

Evaluation Hardware/Configuration (Cont.)

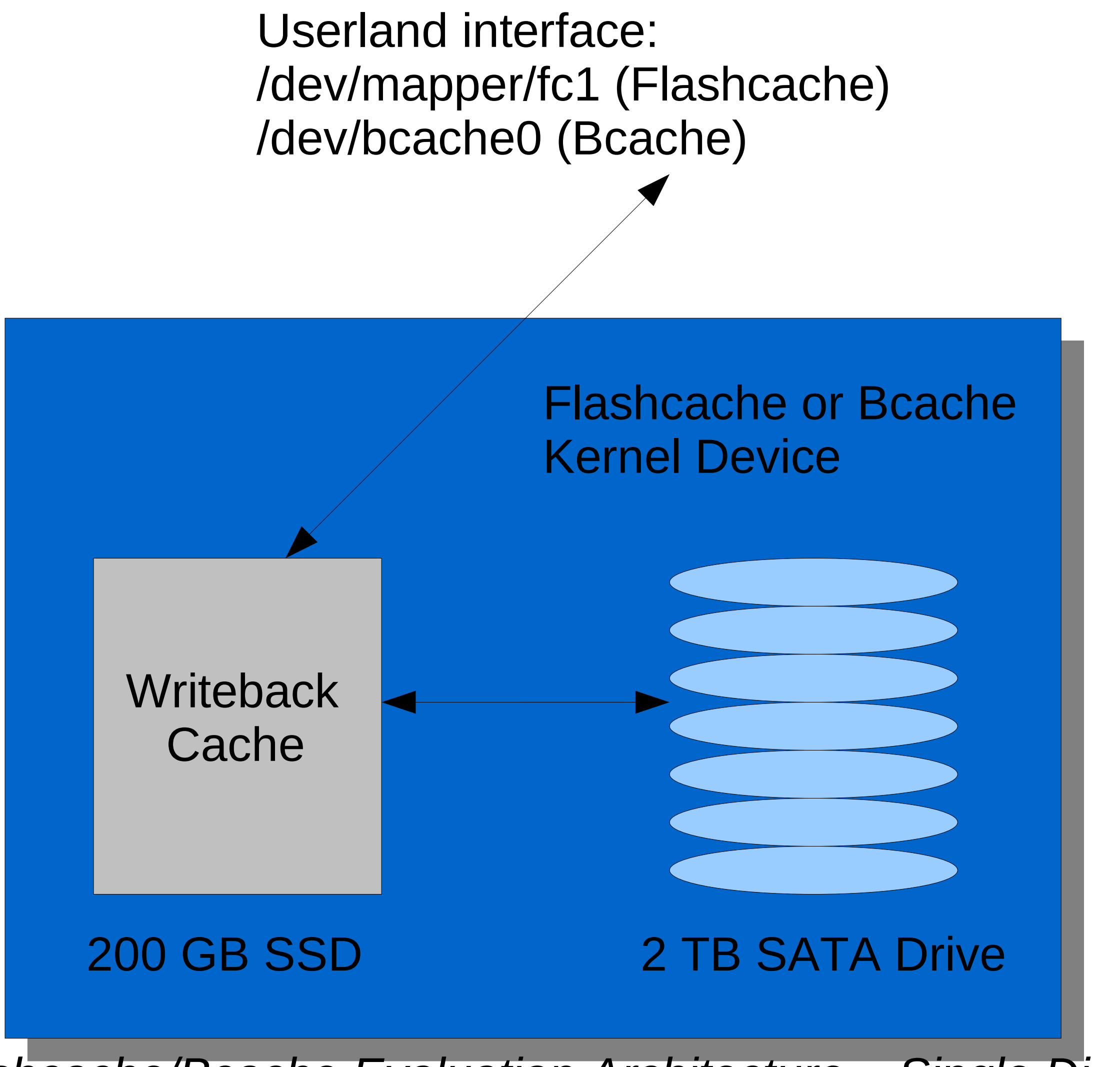
- Software RAID0, Flashcache and Bcache Benchmarks
  - Dell PowerEdge R620
  - 2 8-core Xeon E5-2660@2.20 GHz CPUs (HT on: 32 logical cores total)
  - 48 GB DDR3 1600 MHz RAM
  - PERC H310 disk controller
  - 64-bit Scientific Linux 6.4 (kernel 2.6.32-358.6.2.el6.x86_64, 3.11.1 for Bcache tests)
  - 8 2.5” SATA hard drives in a software RAID0 configuration
    - Only 7 spindles used in the array for Flashcache and Bcache tests
    - Seagate ST9500620NS 2.5” drive
    - 500 GB, SATA 3.0 Gbps
    - 64 MB cache
    - 7200 RPM
    - Firmware release AA09

- SSD TRIM (“discard” mount option) not enabled. EXT4 used in all tests

- Tested both “clean” and “dirty” Flashcache and Bcache configurations
  - Clean - no data written besides filesystem metadata before benchmark
  - Dirty – benchmark run multiple times in succession before final test

Evaluation Hardware/Configuration (Cont.)

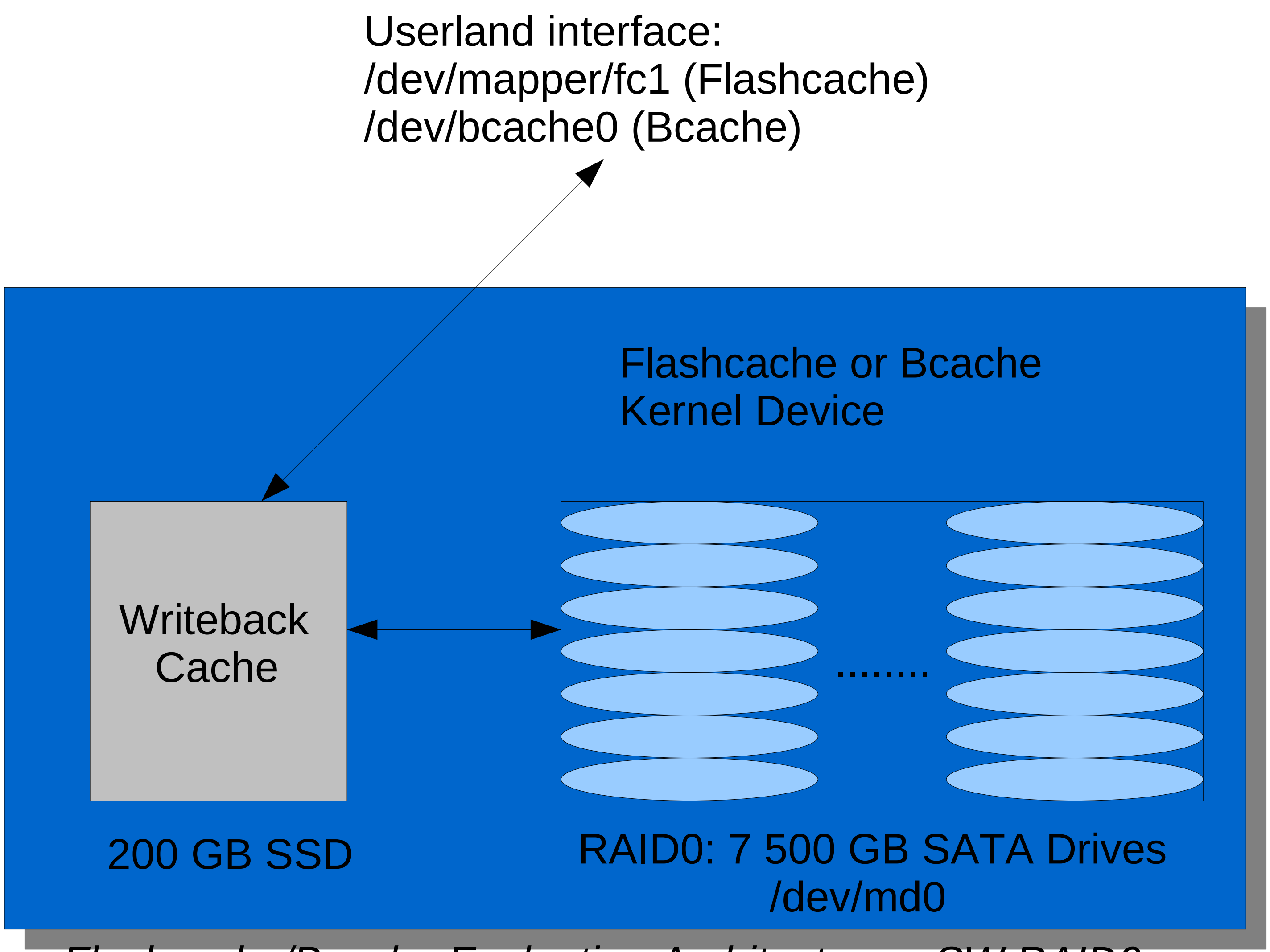
Single SSD cache in front of a single SATA drive, as used in both the Flashcache and Bcache evaluation configurations



Flashcache/Bcache Evaluation Architecture – Single Disk

Evaluation Hardware/Configuration (Cont.)

Single SSD cache in front of a 7 SATA drive software RAID0 (md) array, as used in both the Flashcache and Bcache evaluation configurations



Flashcache/Bcache Evaluation Architecture – SW RAID0

Benchmarks

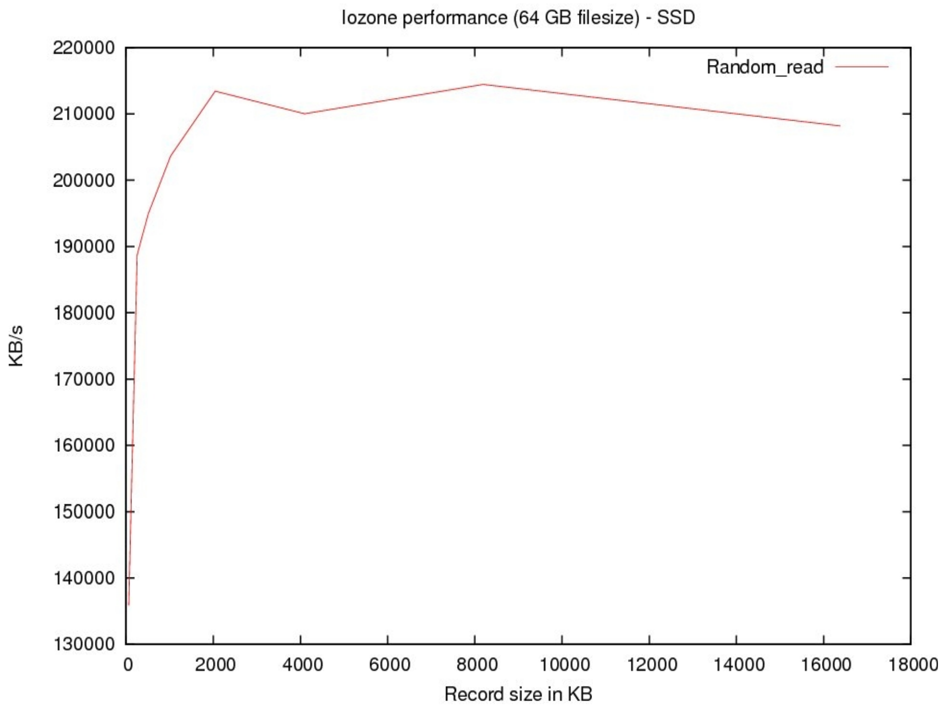
- Iozone
  - Ran version 3.420, with variable record sizes
  - Runs a number of I/O tests, including both random and sequential
  - Supports single-threaded, or multi-threaded (throughput) modes
    - Only the single-threaded test (automatic mode) was run in this study

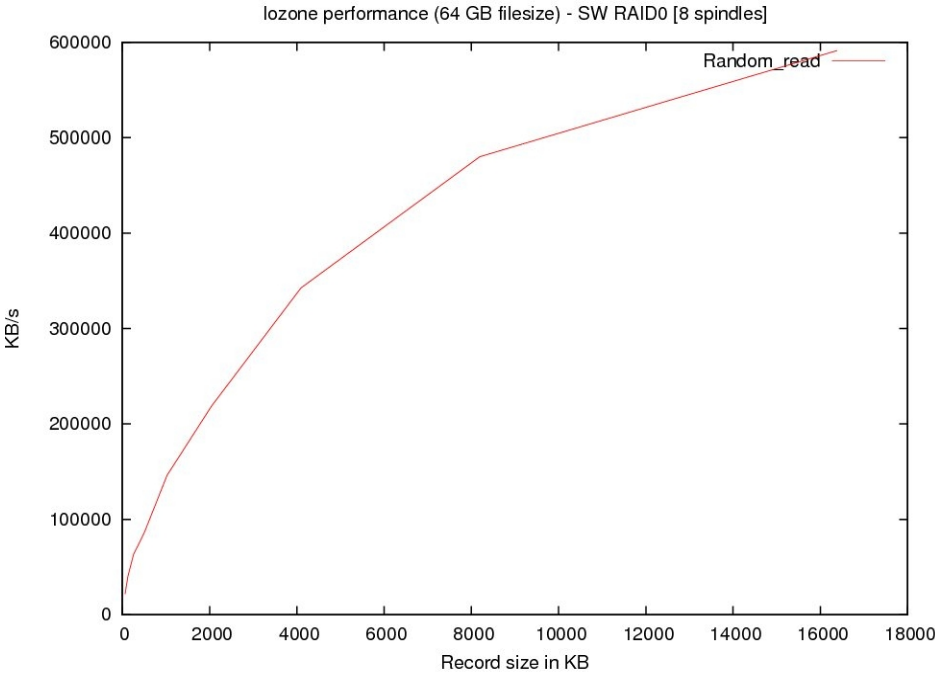
- bonnie++
  - Ran version 1.03e, with 8k and single byte/character record sizes
  - Supports single-process, or synchronized multi-process execution
  - Input/output tests are sequential
    - Using multiple processes, one can effectively create a random workload
    - Multiple processes performing sequential I/O is likely a good model for real world scratch access patterns on HEP/NP worker nodes
      - Aggregated parallel results

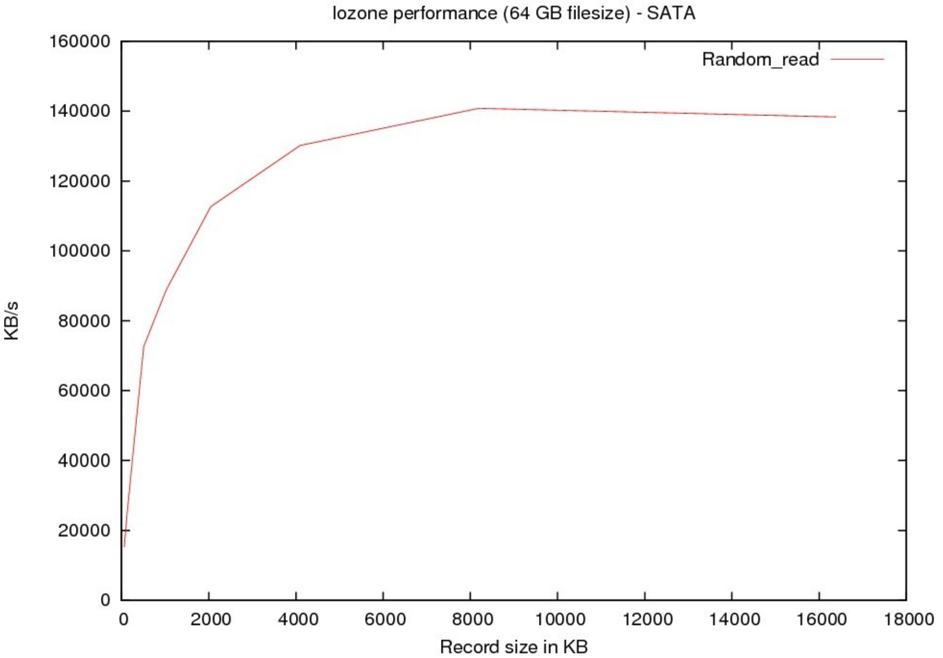
- Importantly, test filesize was set larger than memory size in all benchmarks

Iozone (1 Thread) – Random Read

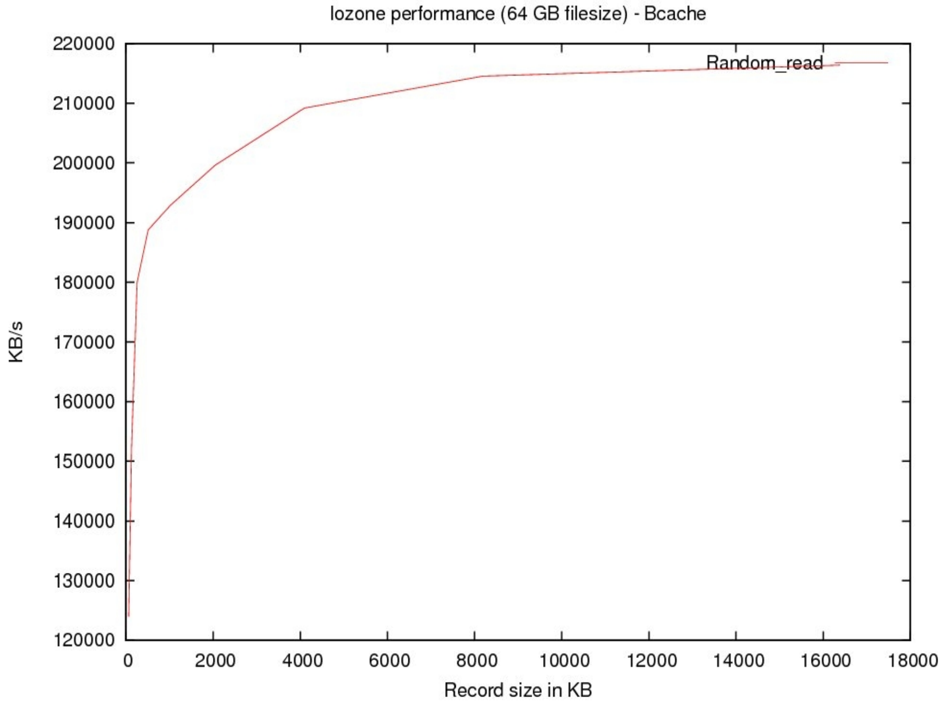
iozone -a -g 120G

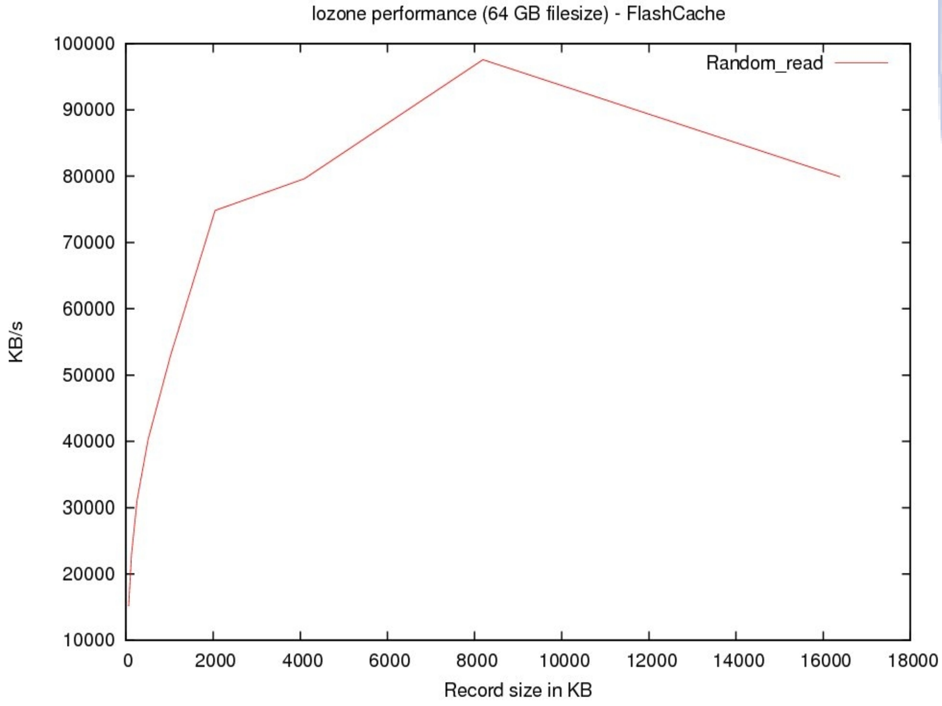






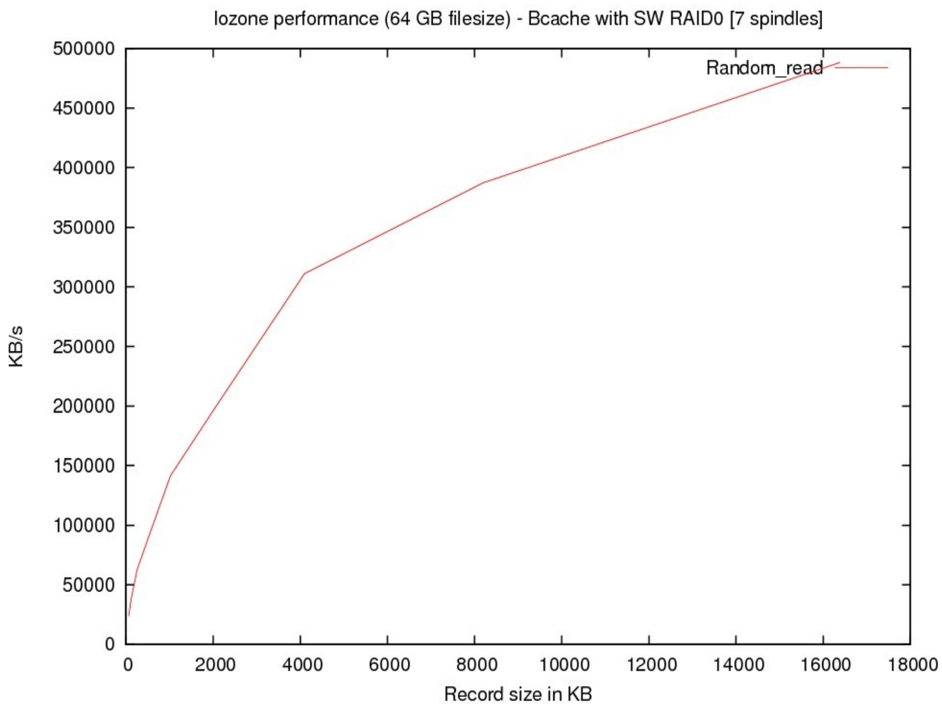
Iozone (1 Thread) – Random Read (Cont.)

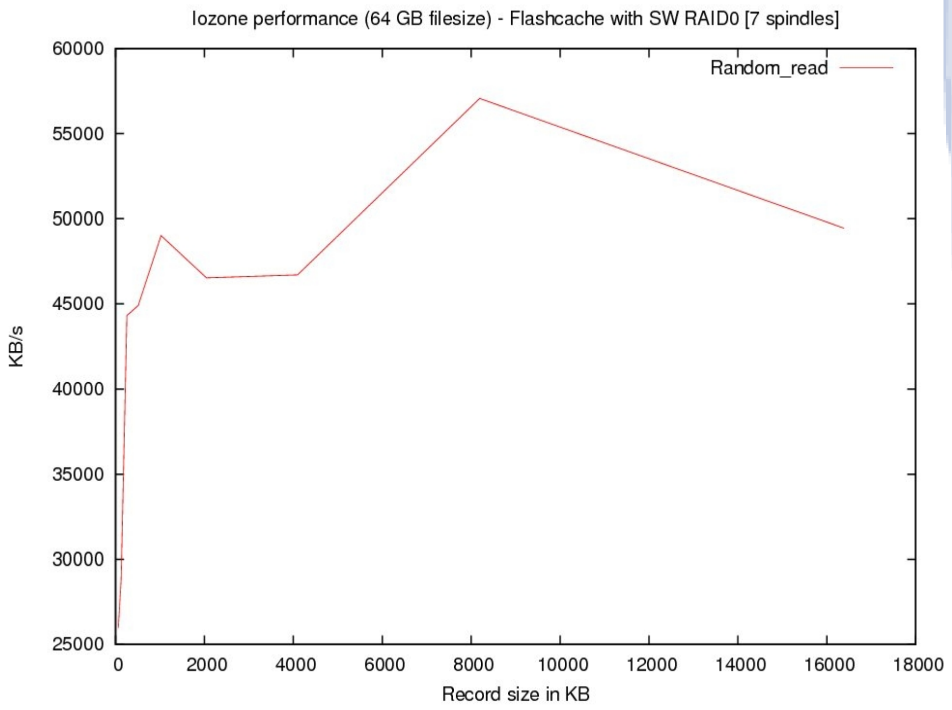




Dirty cache in both test cases

Iozone (1 Thread) – Random Read (Cont.)

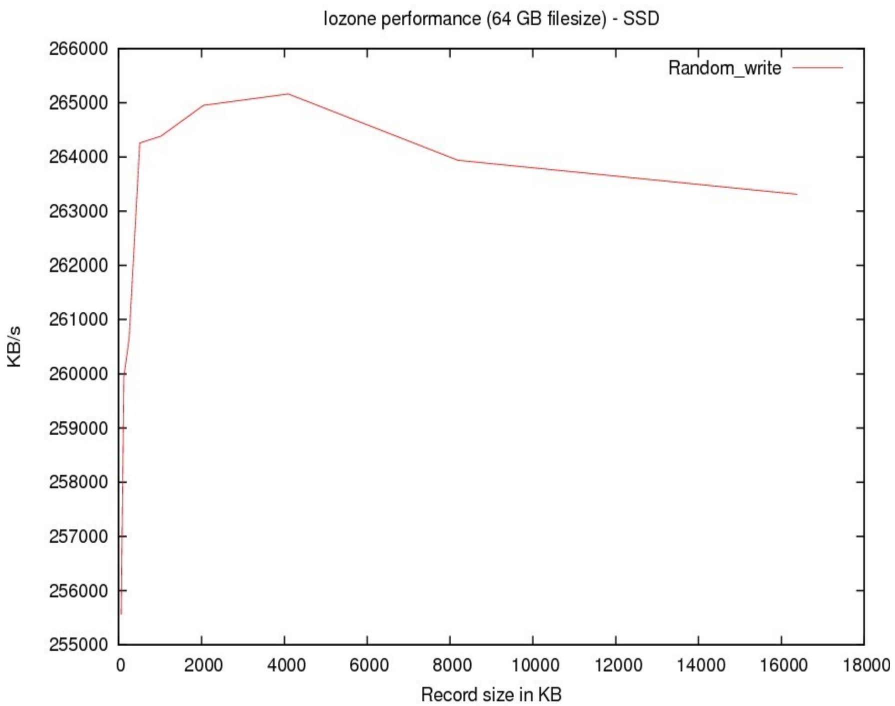


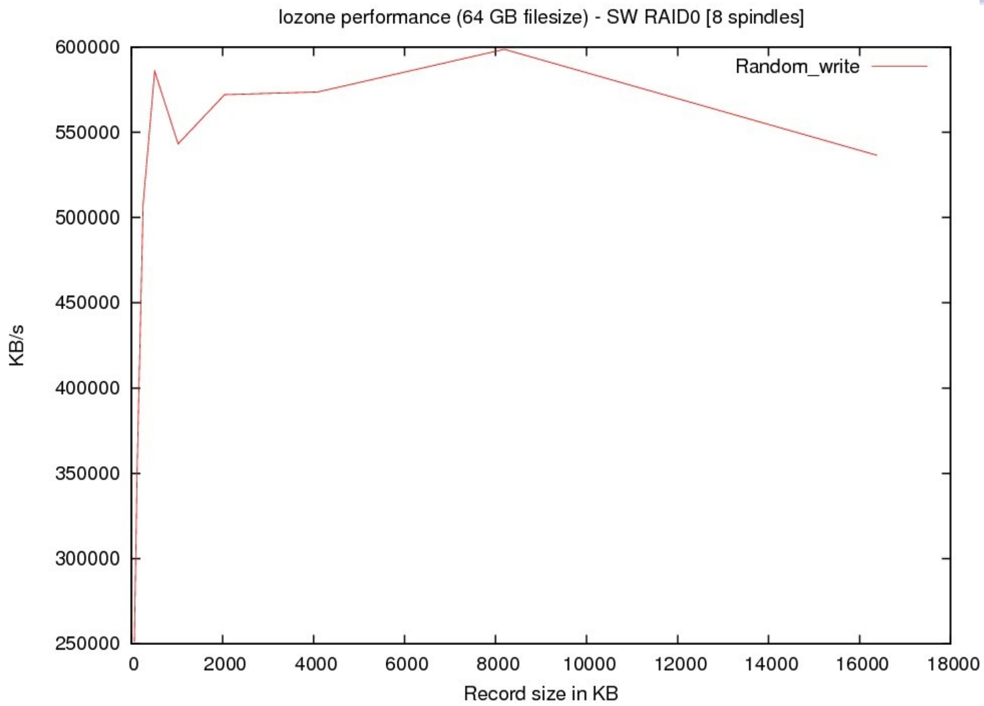


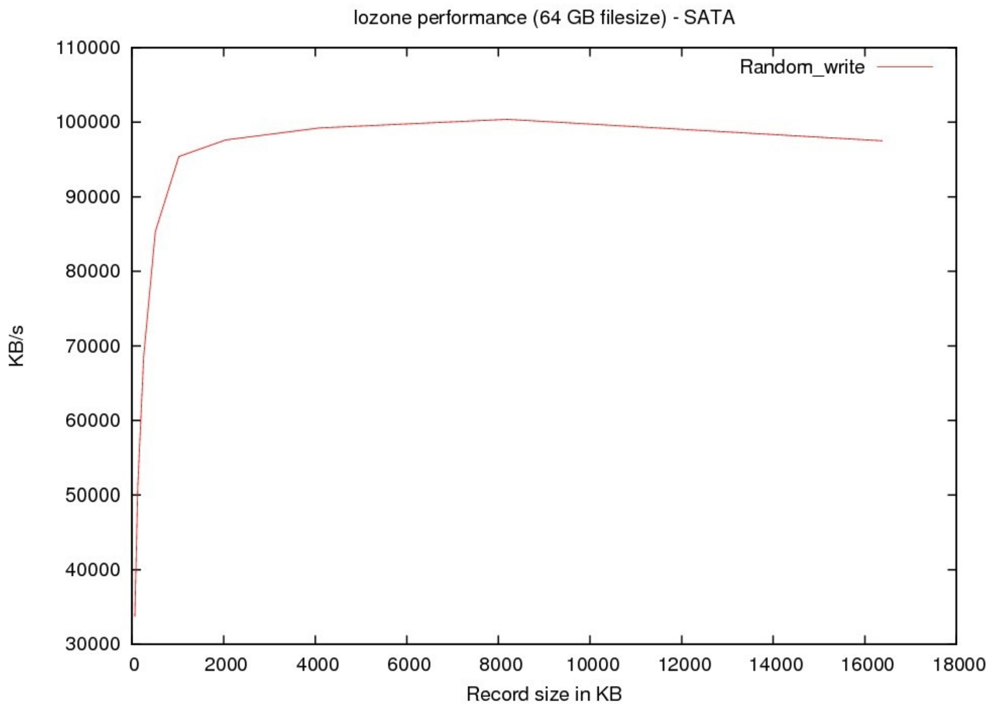
Dirty cache in both test cases

Iozone (1 Thread) – Random Write

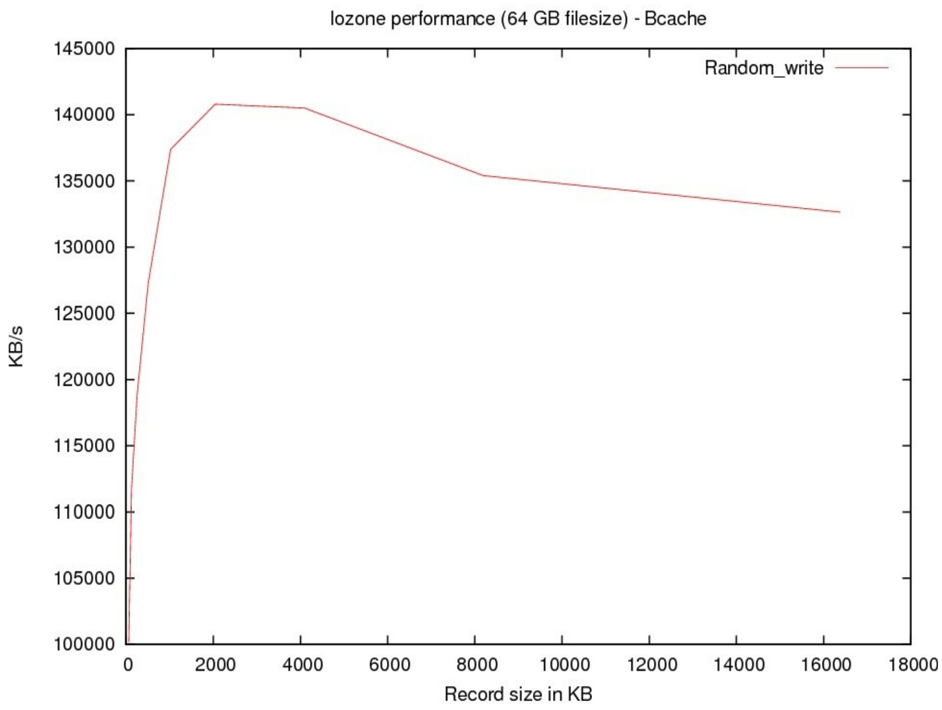
iozone -a -g 120G

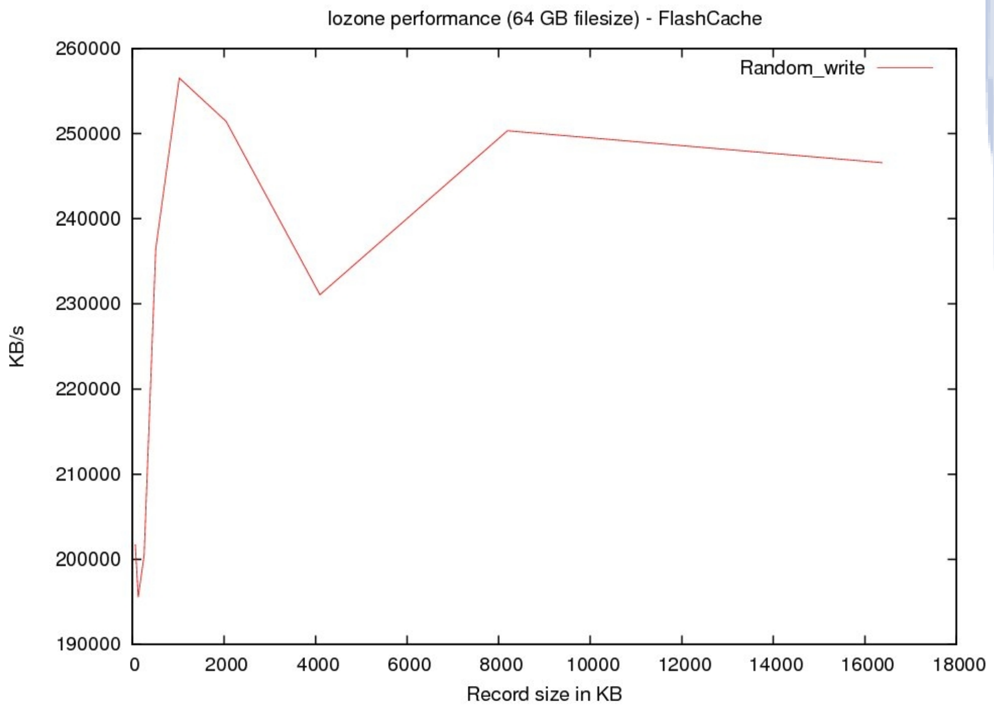






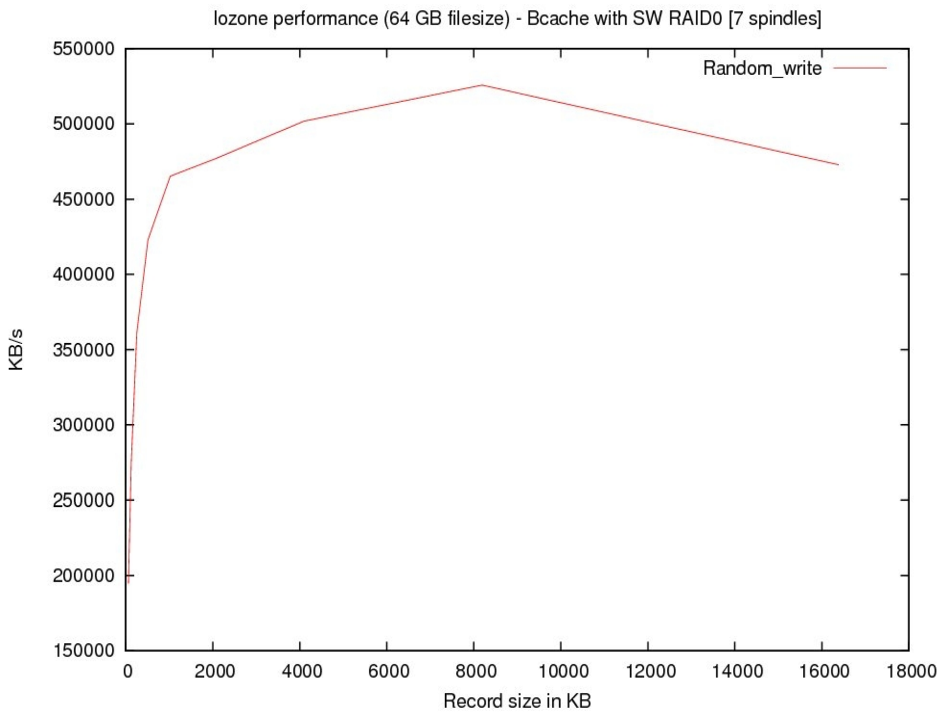
Iozone (1 Thread) – Random Write (Cont.)

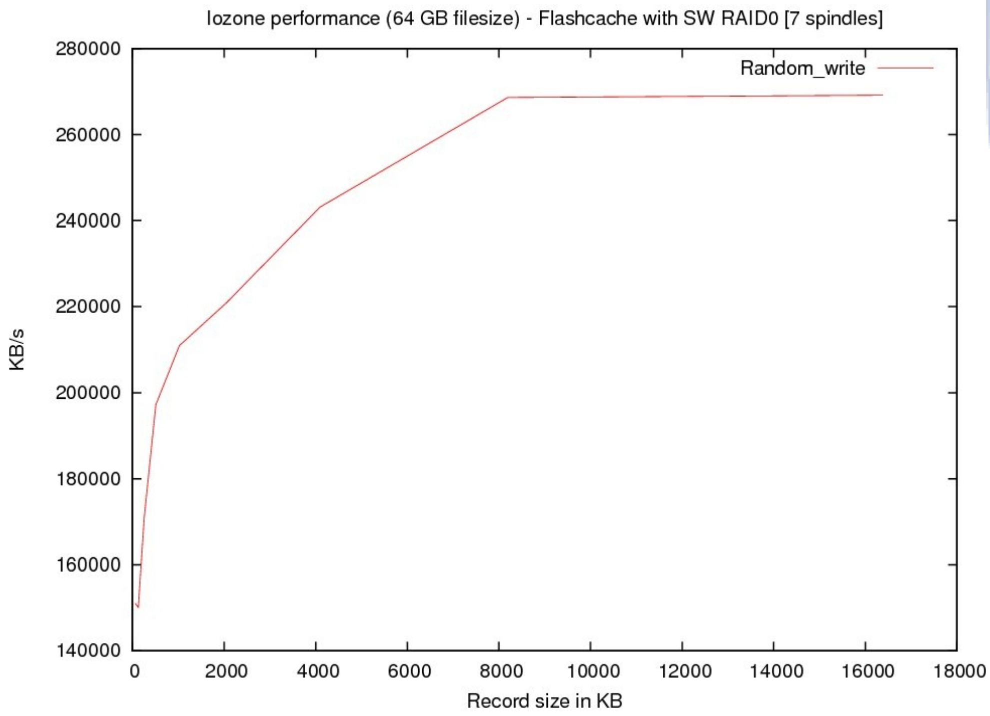




Dirty cache in both test cases

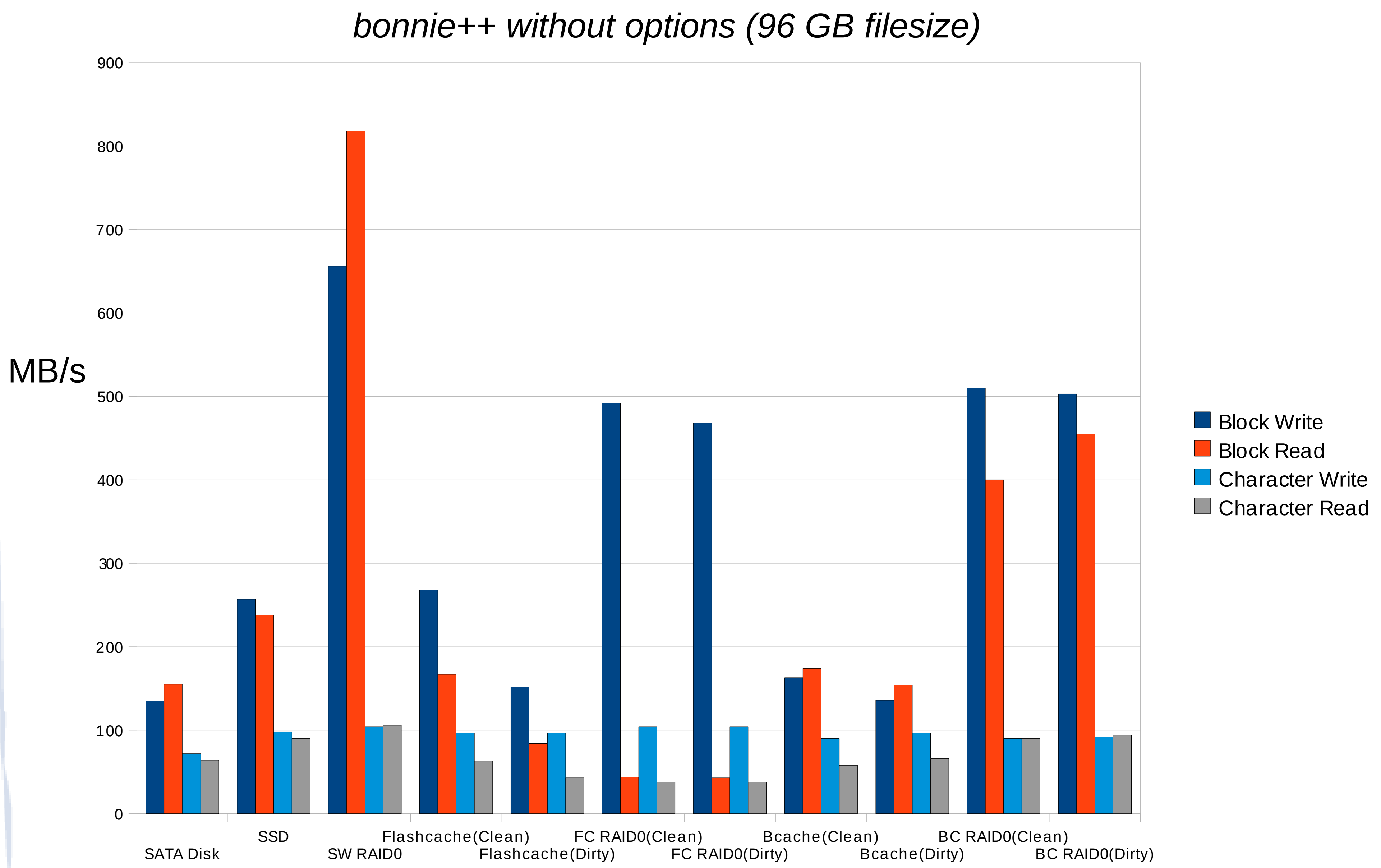
Iozone (1 Thread) – Random Write (Cont.)





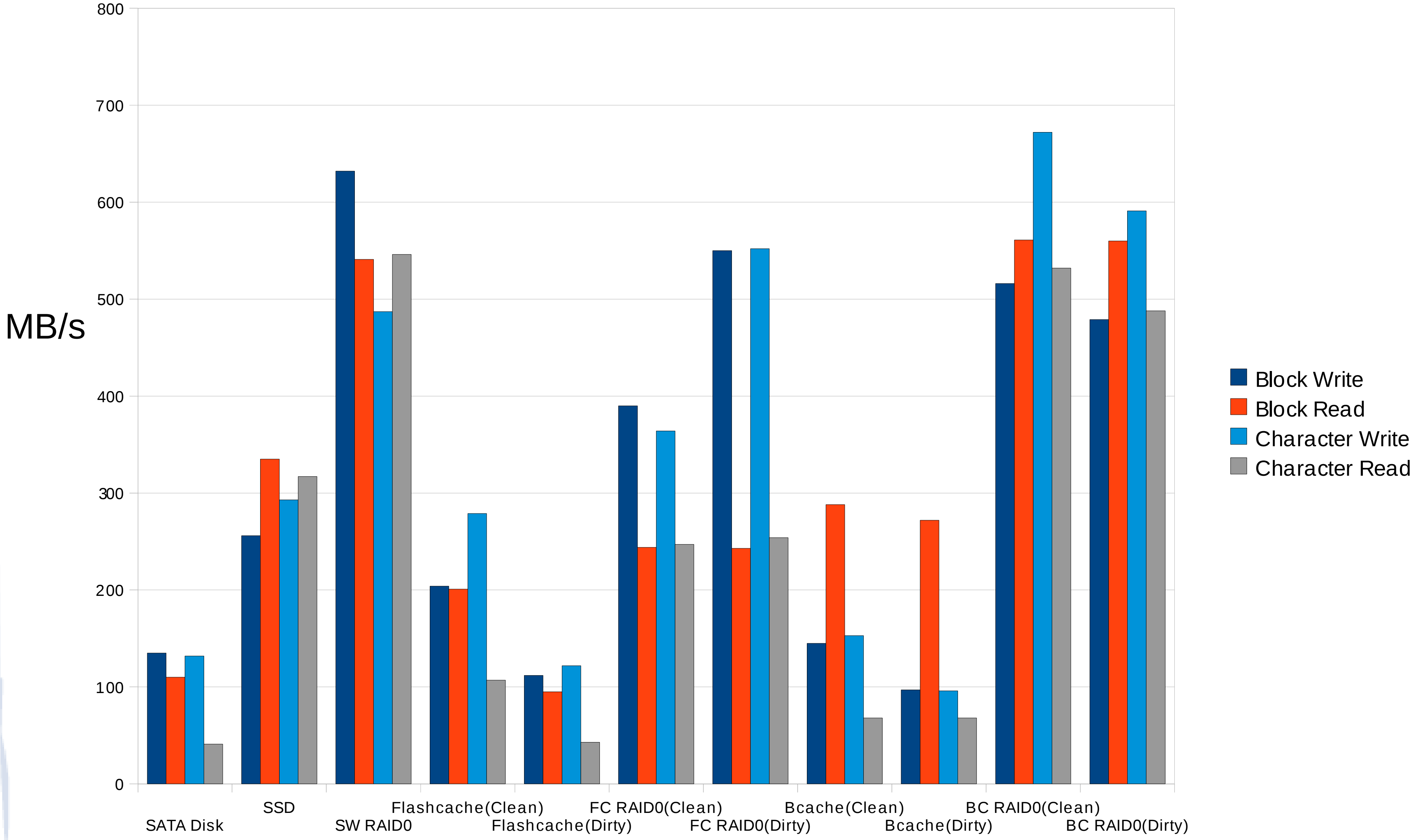
Dirty cache in both test cases

Sequential bonnie++ (1 Process)



Parallel/Random bonnie++

Multiprocess (24) sequential, synchronized, aggregate: bonnie++ -y -r 2560 -s 5120 (122 GB total)
Effectively creates a random workload



Conclusions

The single SSD tested provided excellent random I/O characteristics, particularly for small record sizes, but did not provide the performance of a multi-spindle software RAID0 configuration for larger record sizes

- The software RAID0 configuration provided roughly double the random I/O performance compared to the SSD for large records and for parallel workloads
  - But it consisted of 8 times the number of drives
- Single SSD random I/O performance was significantly better than a single SATA drive

Flashcache and Bcache with an SSD cache generally augmented the I/O performance of a single SATA disk for files that fit within the cache

- Generally true for both random and sequential I/O
- Smaller gains, or performance losses, were typically seen when the cache was preloaded with dirty data during bonnie++ testing
- Probably not suitable for scratch space utilization, since in this use case we're likely dealing with large files that are only written and/or read once
- Would likely benefit database, webserver, or other applications where a set of relatively small files are repeatedly read/written

Conclusions (Cont.)

In the single SATA disk configurations tested, Flashcache typically yielded better random write performance, while Bcache typically yielded better random read performance

Fronting a 7-spindle software RAID0 array with a single SSD cache via Flashcache or Bcache, instead of using an 8-spindle standard RAID0 array (without SSD cache), generally reduced the performance of the array

- Bcache/Flashcache may improve the performance of RAID5/6 arrays – we did not test this setup since we were primarily interested in scratch storage performance

As mentioned, there are a large number of Bcache, Flashcache, kernel and filesystem configuration parameters (as well as many alternative filesystem types). There are so many tunables/variables, that testing variations on each was impossible due to time constraints. However, if the application data access pattern is well known, it is possible that improved Flashcache and Bcache performance can be obtained through additional modification of these parameters

Questions?

Backup Slide

Building a test 3.11.1 kernel from the SL/RHEL6 x86_64 2.6.32 kernel configuration:

```
$ rpm -ivh kernel-2.6.32-358.6.2.el6.src.rpm
$ rpmbuild -bp rpmbuild/SPECS/kernel.spec
$ cp rpmbuild/BUILD/kernel-2.6.32-358.6.2.el6/
   linux-2.6.32-358.6.2.el6.x86_64/configs/kernel-
   2.6.32-x86_64.config linux-3.11.1/.config
$ cd linux-3.11.1
$ make olddefconfig
$ make menuconfig
   Select “Device Drivers” ->  “Multiple devices driver
   support (RAID and LVM)”
   Select “Block device as cache” as a module “<M>”
$ make
# cp arch/x86_64/boot/bzImage /boot/vmlinuz-3.11.1
# make modules_install
# depmod -a 3.11.1
# mkinitrd /boot/initramfs-3.11.1.img 3.11.1
```