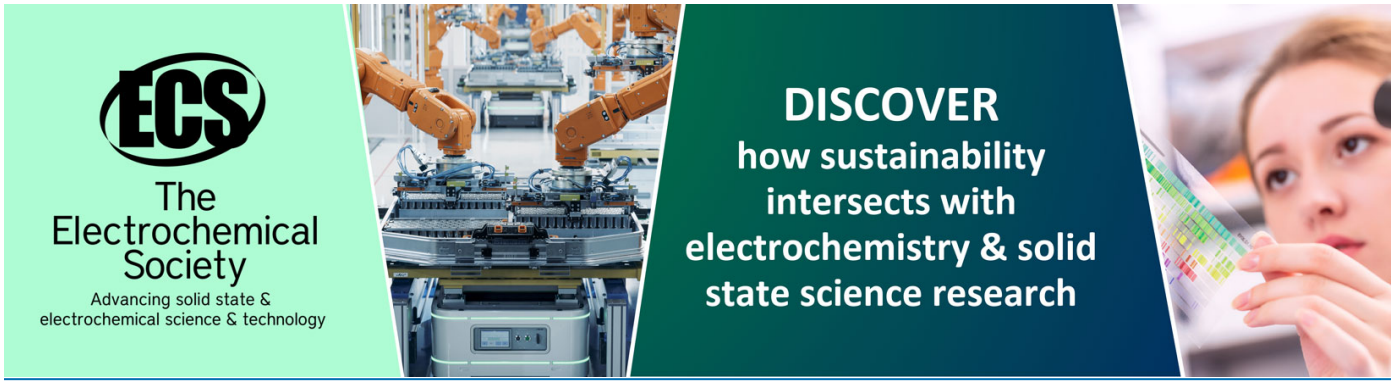
The Effect of Flashcache and Bcache on I/O Performance



Journal of Physics: Conference Series **513** (2014) 062023 doi:10.1088/1742-6596/513/6/062023

The Effect of Flashcache and Bcache on I/O Performance

Christopher Hollowell[1], Richard Hogue[1], Jason Smith[1], William Strecker-Kellogg[1], Antonio Wong[1], Alexandr Zaytsev[1]

[1]Brookhaven National Laboratory, Upton, NY 11973, USA

E-mail: hollowec@bnl.gov, rhogue@bnl.gov, smithj4@bnl.gov, willsk@bnl.gov, tony@bnl.gov, alezayt@bnl.gov

Abstract. Solid state drives (SSDs) provide significant improvements in random I/O performance over traditional rotating SATA and SAS drives. While the cost of SSDs has been steadily declining over the past few years, high density SSDs continue to remain prohibitively expensive when compared to traditional drives. Currently, 1 TB SSDs generally cost more than USD $1,000, while 1 TB SATA drives typically retail for under USD $100. With ever-increasing x86_64 server CPU core counts, and therefore job slot counts, local scratch space density and random I/O performance have become even more important for HEP/NP applications.

Flashcache and Bcache are Linux kernel modules which implement caching of SATA/SAS hard drive data on SSDs, effectively allowing one to create hybrid SSD drives using software. In this paper, we discuss our experience with Flashcache and Bcache, and the effects of this software on local scratch storage performance.

1. Introduction

Over the past few years, x86_64 server processor core counts have continued to increase. Sandy Bridge processors used in servers today already commonly provide 16 logical cores per physical CPU. In the not so distant future, Haswell-based server CPUs are expected to provide up to 40 logical cores in a single physical package [1]. In the traditional High Energy Physics and Nuclear Physics (HEP/NP) batch processing model, one job slot is allocated for each logical core in a system. This has lead to a constant increase in the number of jobs per host, and therefore in demand for local scratch and remote storage I/O performance and density. There has been some success in the development of multithreaded HEP/NP batch processing applications and frameworks, which may help alleviate increasing random I/O performance requirements, potentially through the use of scatter/gather I/O models [2]. However, as of the time of writing, multithreaded batch processing has not yet been widely adopted by the experiments utilizing our computing center, the RHIC/ATLAS Computing Facility (RACF) at Brookhaven National Laboratory (BNL).

The current method we've adopted for improving scratch I/O performance on our worker nodes has been to populate these systems with multiple commodity drives, and combine them into software RAID0 arrays. This can be somewhat expensive, however, as it requires that many drives be purchased for each host. Furthermore, the increase in core density is outpacing the increase in available spindle count in typical worker node server hardware, and therefore, this solution may not scale at some point in the near future.

Table 1. Hybrid device operations.

Operation	**Flashcache**	**Bcache**
Insert Module	`modprobe flashcache`	`modprobe bcache`

Create Device	`flashcache_create -p back fc1 /dev/SSD /dev/HDD`	`make-bcache -B /dev/SSD`
		`make-bcache -C /dev/HDD`
		`cd /sys/fs/bcache`
		`echo /dev/SSD > register`
		`echo /dev/HDD > register`
		`cd /sys/block/bcache0/bcache`
		`echo CACHESET_UUID > attach`

Load Existing	`flashcache_load /dev/SSD`	`cd /sys/fs/bcache`
		`echo /dev/SSD > register`
		`echo /dev/HDD > register`

Format Device	`mkfs.ext4 /dev/mapper/fc1`	`mkfs.ext4 /dev/bcache0`

SSDs provide excellent random I/O performance characteristics, and we considered adopting them for scratch storage in our worker nodes. However, they are extremely expensive for the capacities we require. For instance, as of September 2013, 1 TB SSD drives typically cost near or above USD $1,000, with enterprise models exceeding USD $2,500. In contrast, a commodity 1 TB SATA drive retails for less than USD $100. Therefore, we decided to look into the possibility of using hybrid hard drives, which have a flash memory cache in front of traditional rotating media. While Seagate offers such products close to the price of traditional SATA drives, they have not yet been validated for use in servers offered by large scale manufacturers such as Dell. Therefore, this technology was not a viable option for us.

Hybrid devices can also be implemented in software, and there are currently a number of Linux kernel drivers available which implement this functionality, two of which are Flashcache and Bcache. We tested both of these driver implementations to determine if they could be used to eliminate the need for multi-spindle RAID0 array scratch storage on worker nodes, or increase the performance of these arrays.

2. Flashcache and Bcache

While Flashcache and Bcache provide similar functionality, there are a number of important differences between the two projects. Bcache has been integrated into the Linux kernel as stable software since the 3.10 release. Flashcache, on the other hand, is available only as separate software, which is compiled as a module outside of the kernel source. Whereas Flashcache is controlled via the kernel's sysctl interface, Bcache uses Sysfs (/sys) for configuration parameter modification. Finally, Bcache utilizes a btree cache structure [3], while Flashcache's cache is structured as a set associative hash [4]. The steps we used to setup and configure our Flashcache and Bcache devices are available in Table 1.

Table 2. Hybrid device parameters.

Flashcache	**Bcache**
dev.flashcache.SSD+HDD.reclaim_policy = 0 (FIFO)	cache_replacement_policy = "lru"
dev.flashcache.SSD+HDD.fallow_delay = 900	writeback_delay = 30
dev.flashcache.SSD+HDD.skip_seq_thresh_kb = 0	sequential_cutoff = 0
Writeback cache type set at creation time	cache_mode = "writeback"

There are number of configuration options available for both drivers. Most importantly, all of our tests were performed with the Flashcache and Bcache device SSD caches in writeback mode. A listing of some additional relevant parameter settings for our tests are present in Table 2. The number of available options is quite large: testing variations on each was not possible given our time constraints. For simplicity, in general, configuration defaults were untouched, and only altered when it was believed that significant performance gains could be achieved, or when such modifications were recommended in documentation.

3. Hardware and Operating System Configuration

Details on the hardware configuration used during our benchmarking is available in Table 3. 64-bit Scientific Linux (SL) 6.4 was used as the operating system in all tests. However, the use of a custom configured/compiled vanilla upstream 3.11.1 kernel was necessary during Bcache testing. This was because it was not possible (at least not without heavy modification) to extract the Bcache source from the upstream 3.x kernel tree, and integrate it into the stock SL6 2.6.32 kernel source.

For the Flashcache and Bcache benchmarks, two hardware configurations were tested: one with a single SSD cache in front of a single SATA drive, and one with a single SSD cache fronting a 7-spindle software RAID0 array. For comparison, the same benchmarks were run on a single SSD, a single SATA drive, and an 8-spindle software RAID0 device. Due to the fact that SSDs are not subject to the I/O timings required for efficient access to rotating media, the "deadline" or "noop" kernel I/O schedulers should be used with these drives. During our tests, we set the kernel I/O scheduler to "deadline" for the SSD, and TRIM was not enabled at mount time.

Only one SSD model was tested in our benchmarks due to time and available hardware constraints. It's possible that better performance could have been obtained from Flashcache and Bcache through the use of a higher performing SSD, or multiples thereof. However, the performance of the SSD utilized in this study was in line with the results of other SSD makes/models we've benchmarked in the past.

4. Benchmarks

For performance testing, we utilized both Bonnie++ [5] and Iozone [6]. When these benchmarks were run, we were careful to ensure that filesize was set larger than total system memory size. Bonnie++ only tests sequential I/O performance, but it supports a synchronized multiprocess/parallel execution mode. In this way, the benchmark can be used to create a random workload, albeit not heavily randomized due to kernel I/O buffering and scheduling optimizations that attempt to sequentialize all I/O to disk. In particular, this benchmark, with multiple processes performing sequential I/O, is likely a good model for HEP/NP batch process scratch access. This is especially the case at RACF where some of the experiments we support stream input/output files to/from local scratch storage at the beginning and end of their batch jobs.

Table 3. Evaluation hardware configuration.

Component	**Single SATA & SSD Tests**	**SW RAID0 Tests**
Server	Dell PowerEdge R410	Dell PowerEdge R620
CPU	2 6-core Xeon X5660@2.80 GHz	2 8-core Xeon E5-2660@2.20 GHz
Total Logical Cores	24	32
Memory	48 GB DDR3 1333 MHz	48 GB DDR3 1600 Mhz
Disk Controller	SAS 6/IR	PERC H310
Hard Drives	1 Seagate ST32000644NS 3.5" 2 TB	8 [7 for Flashcache/Bcache] Seagate ST9500620NS 2.5" 500 GB
HDD Attributes	7200 RPM SATA 3.0 Gbps	7200 RPM SATA 3.0 Gbps
SSD Drive	Samsung SM825 2.5" 200 GB	Samsung SM825 2.5" 200 GB
SSD Attributes	Enterprise eMLC SATA 3.0 Gbps	Enterprise eMLC SATA 3.0 Gbps

For this study, we ran Bonnie++ as both a single process for sequential I/O testing, and as 24 synchronized processes in parallel for random I/O testing. Iozone can run in multithreaded (throughput mode), or single-threaded modes, and performs several I/O tests, including both sequential and random. Only single-threaded Iozone tests were run, and we were primarily interested in the random I/O benchmark results. Iozone was run in "automatic mode," where record size was varied multiple times during execution. Bonnie++ was run with only two record sizes: 8 KB, and single byte (character).

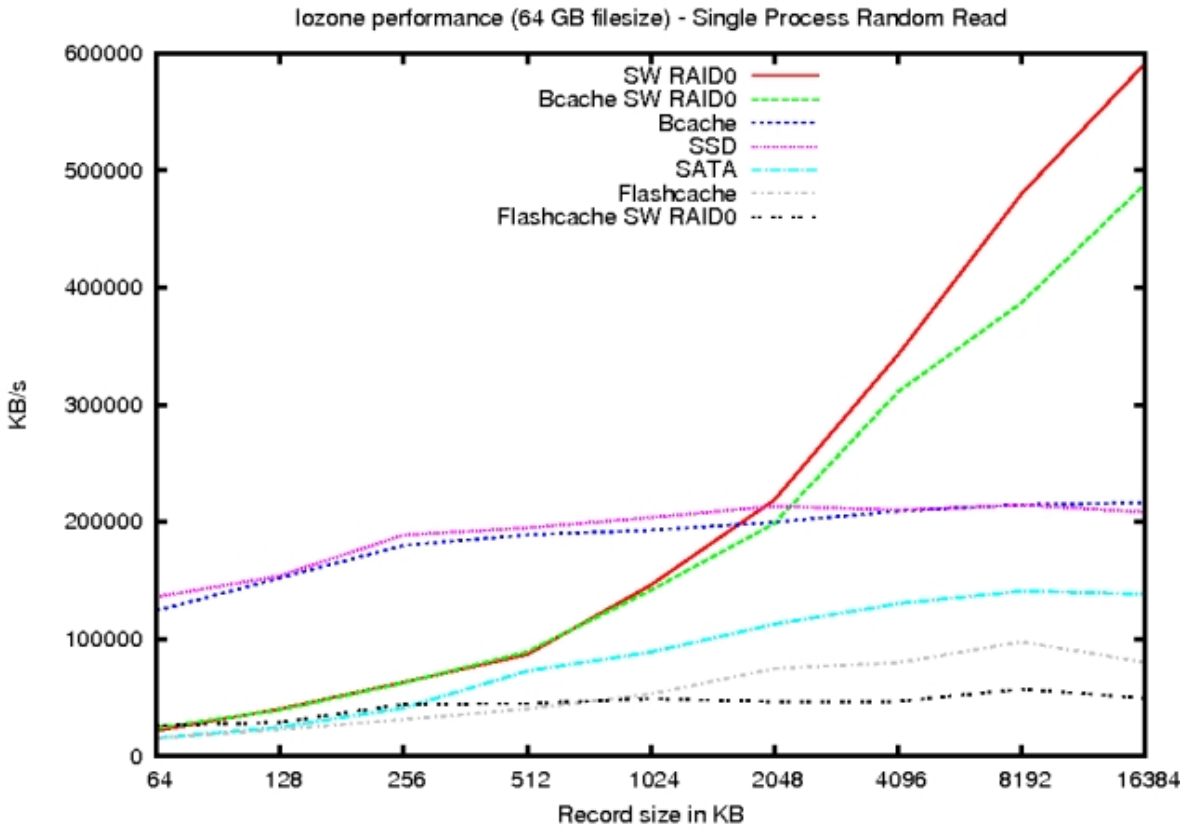


Figure 1. Iozone random read performance.

As the parallel Bonnie++ results were most relevant to our use case, the Bonnie++ benchmarks were run with both clean and dirty Flashcache and Bcache SSD caches. Dirty test results were obtained by running the benchmark a number of times in succession before the final run, where results were collected. Clean tests were first runs of the benchmark on a newly created device and filesystem. The Iozone results presented can be considered dirty, since the benchmark was run multiple times with varying filesizes before the final 64 GB file tests.

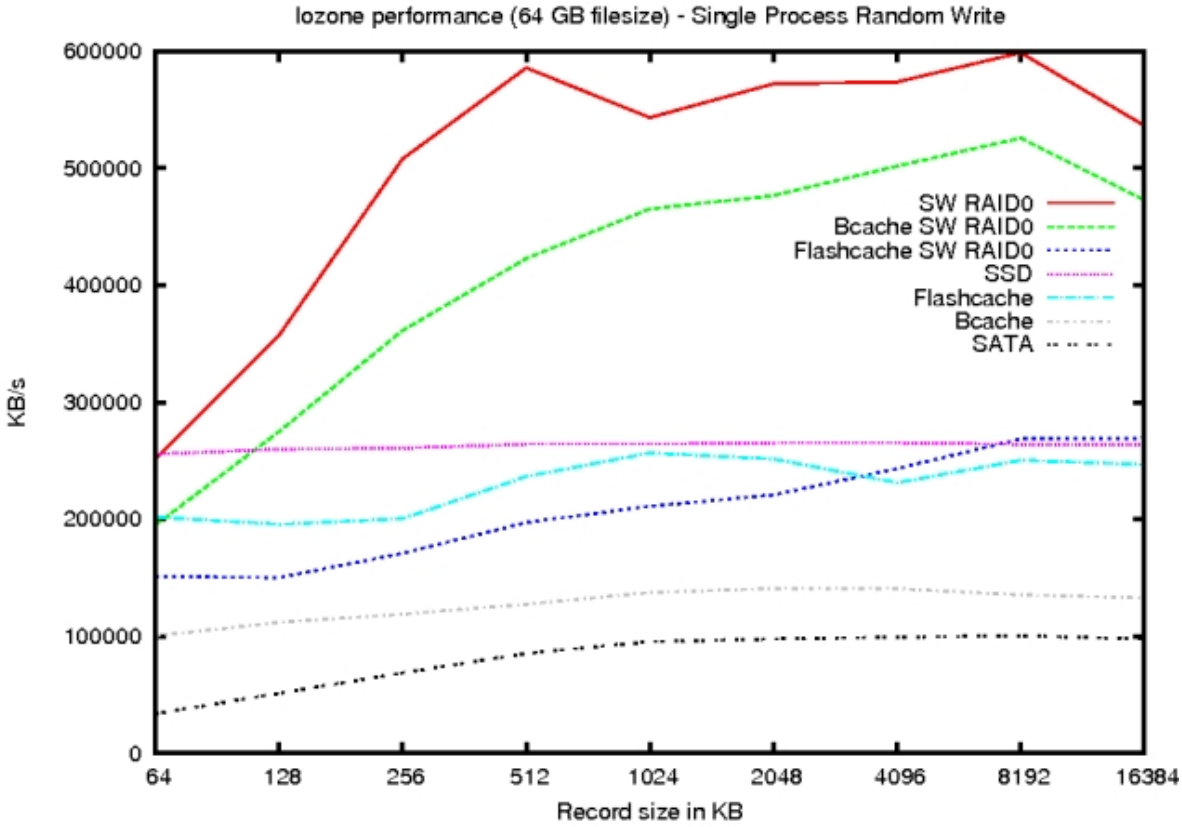


Figure 2. Iozone random write performance.

5. Results

While the single SSD provided excellent random I/O benchmark results, particularly for small record sizes, it did not provide the performance of the multi-spindle software RAID0 configuration for larger records. The software RAID0 configuration provided roughly double the random I/O performance, when compared to the SSD for large records, and for parallel workloads. However, it consisted of 8 times the number of drives. As expected, 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. Flashcache usually yielded better random write performance in the single disk configuration, while Bcache provided for better overall random read performance. Smaller gains, or performance losses were typically seen when the cache was preloaded with dirty data during Bonnie++ testing.

Fronting a 7-spindle software RAID0 array with a single SSD cache via Flashcache and Bcache generally reduced the performance of the array, when compared to an 8-spindle RAID0 device without an SSD cache. It's possible that Flashcache and Bcache may improve the performance of RAID5/6 devices. However, we did not test these RAID levels, as we were mainly interested in the performance of local scratch storage, where they are not commonly used. Bcache backed by a

7-spindle software RAID0 array came close to the performance of the 8-spindle cacheless software RAID0 device, and in some tests surpassed it. This is likely due to Bcache's monitoring of SSD congestion, and its default automatic cache bypass capability during periods of high latency SSD access.

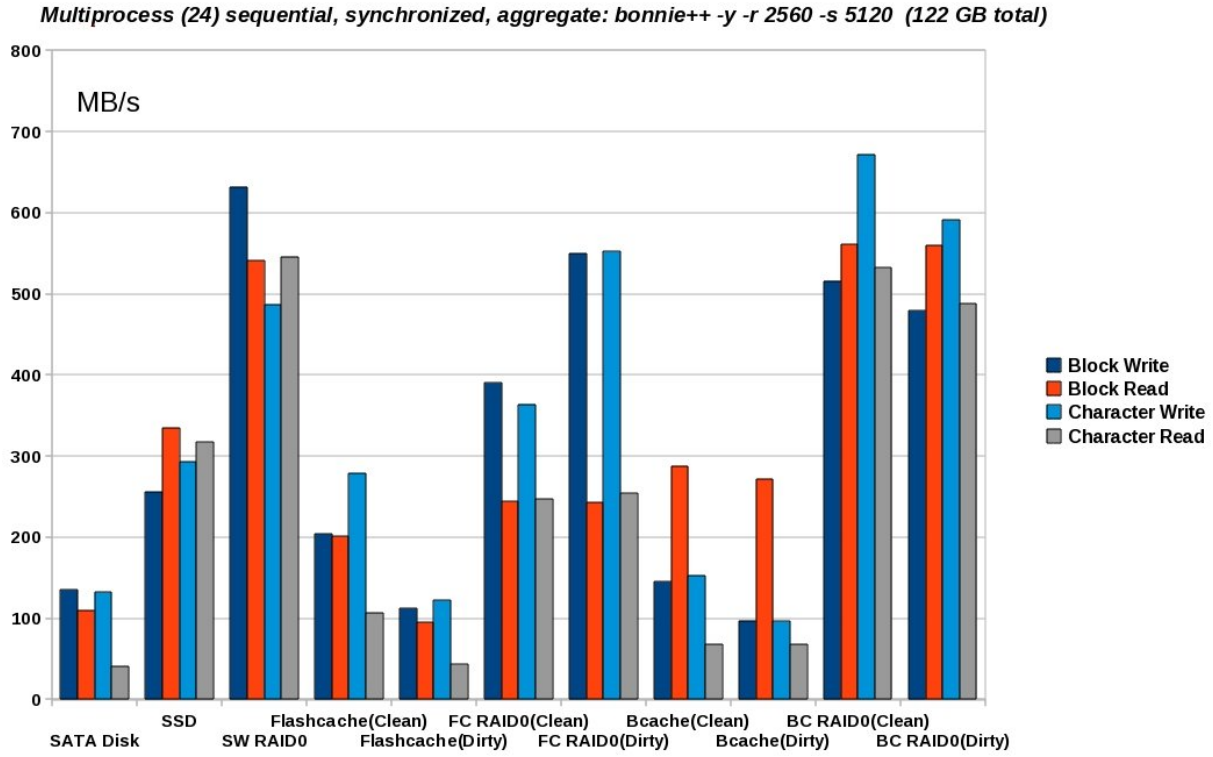


Figure 3. Bonnie++ parallel sequential read/write performance - generates a randomized workload.

6. Conclusions

We tested Bcache and Flashcache, two software hybrid device implementations for the Linux kernel, to determine if they could be used to address the continued increase in I/O demands from HEP/NP jobs running on worker nodes with progressively larger core/slot counts. We were primarily interested in increasing local worker node scratch storage performance, and/or reducing the cost of this storage. While it appears that Flashcache and Bcache can be used to improve the performance of a single SATA drive, these kernel drivers, with a single SSD cache, cannot provide the overall performance of multi-spindle software RAID0 devices, or be used to greatly improve the performance of these arrays. Therefore, while Flashcache and Bcache may be well suited for use in situations where sets of relatively small files are repeatedly and randomly accessed on a single drive or slow redundant storage, this software does not appear to be particularly well suited for use in HEP/NP worker node local scratch storage environments. Until the cost of high capacity SSDs significantly drops, or the I/O performance and/or density requirements for HEP/NP jobs is reduced (i.e. through the widespread adoption of multithreaded batch processing models), the use of software RAID0 arrays consisting of many traditional rotating drives appears to be the best option for local scratch storage.

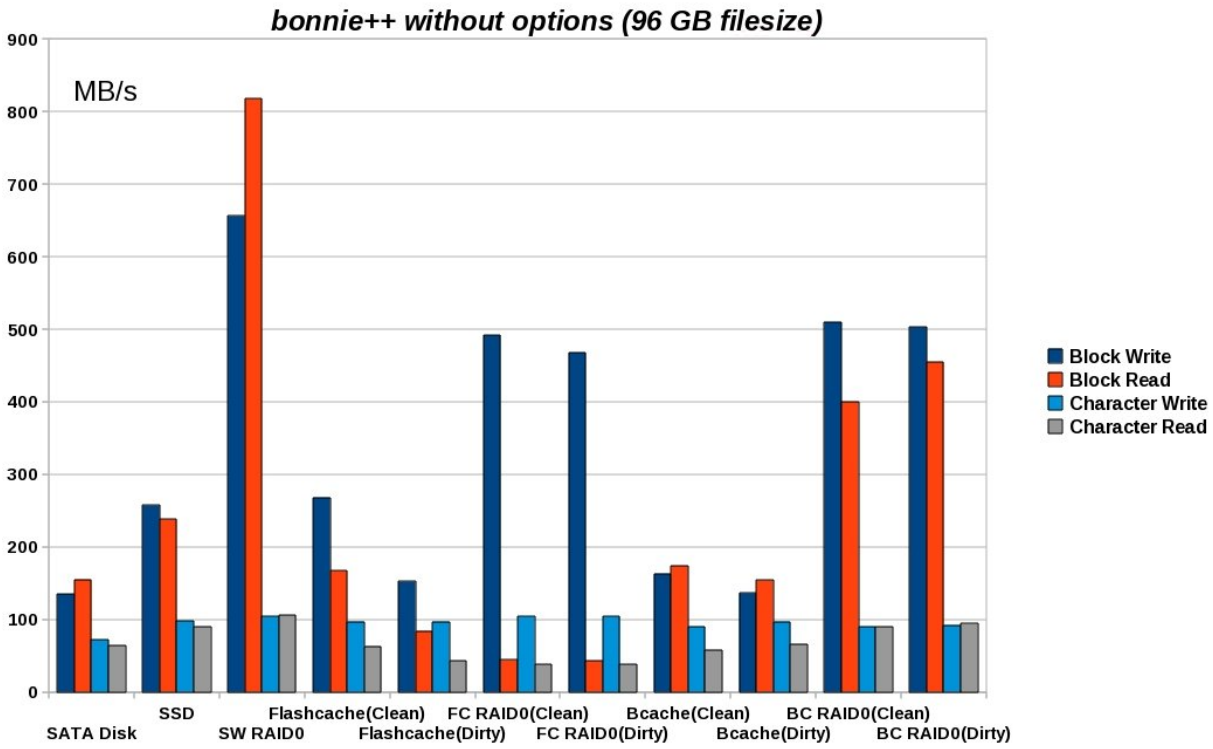


Figure 4. Bonnie++ sequential read/write performance.

References

[1] Haswell Microarchitecture: http://en.wikipedia.org/wiki/Haswell_(microarchitecture)
[2] van Gemmeren P, *et al.* 2012 I/O Strategies for Multicore Processing in ATLAS *2012 J. Phys.: Conf. Ser. 396 022054*
[3] Bcache: http://bcache.evilpiepirate.org/
[4] Flashcache: https://github.com/facebook/flashcache/
[5] Bonnie++: http://www.coker.com.au/bonnie++/
[6] Iozone: http://www.iozone.org/