Title	Impact of single parameter changes on Ceph cloud storage performance
Author(s)	Meyer, Stefan; Morrison, John P.
Publication date	2016
Original citation	Meyer, S. and Morrison, J. P. (2016) 'Impact of single parameter changes on Ceph cloud storage performance', Scalable Computing, 17(4), pp. 285-298. doi:10.12694/scpe.v17i4.1201
Type of publication	Article (peer-reviewed)
Link to publisher's version	http://dx.doi.org/10.12694/scpe.v17i4.1201 Access to the full text of the published version may require a subscription.
Rights	**© 2016 SCPE.**
Item downloaded from	http://hdl.handle.net/10468/3365

DOI 10.12694/scpe.v17i4.1201
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 4, pp. 285–298. http://www.scpe.org © 2016 SCPE

IMPACT OF SINGLE PARAMETER CHANGES ON CEPH CLOUD STORAGE PERFORMANCE

STEFAN MEYER AND JOHN P. MORRISON *

Abstract. In a general purpose cloud system efficiencies are yet to be had from supporting diverse applications and their requirements within a storage system used for a private cloud. Supporting such diverse requirements poses a significant challenge in a storage system that supports fine grained configuration on a variety of parameters.

This paper uses the Ceph distributed file system, and in particular its global parameters, to show how a single changed parameter can effect the performance for a range of access patterns when tested with an OpenStack cloud system.

Key words: Ceph, Cloud Storage, File systems.

AMS subject classifications. 68M14, 68P20

1. Introduction.

Cloud systems are becoming more and more complex and can be adopted to suit various types of use cases and users make use of cloud resources in very distinct ways. Their requirements change depending on the workload, which include services such as web-, database-, email- or directory-servers, but they can also be used for other purposes, such as virtual desktops, rendering or genome sequencing. The requirements of these services vary not only in the component of the system that is used the most, such as the CPU, memory system or networking, but also the characteristics of it.

When looking at storage systems the requirements of services vary greatly. Access patterns consist of a mixture of reads and/or writes, in sequential and/or random fashion, with varying block sizes. Depending on the application the metric with the highest importance might be the throughput, the average or maximum latency or a mixture of both. The prime motivation for the technical solution here is an attempt to answer the question of how a cloud operator can change the storage configuration to better support these specific access patterns.

The most popular open source cloud stack is the very actively developed OpenStack. In the development of the Kilo release there were 148 contributing organizations plus many independent developers [11] making over 19.000 commits [7]. It is highly customizable and supports many implementations as a back-end for the different storage services. Due to the popularity and interest in the system many companies offer solutions or interfaces that hooks into their own systems, which can be seen in the supported storage or networking back-ends. The storage back-end that will be looked at in this paper for improving performance under specific loads is the open source distributed file system Ceph[1], which was acquired by RedHat in 2014.

The options this combination offers will be subject of this paper. In Section 2 the general idea of offering distinct storage solutions will be explained. In Section 3 the OpenStack storage components will be introduced, followed by an introduction to Ceph in Section 4 and the integration of both and the offered options for performance optimizations. Section 5 is showing the results when the cluster is using different configurations. The conclusion of the experiments is presented in Section 6.

2. Motivation.

OpenStack offers the option to set quotas for the I/O system of the VMs. Limits can be set for the amount of read/write/total bytes per second (B/s) or the same categories for the operations per second (IOPS). This allows fine grained limits that can limit the throughput effectively for small file (via IOPS) and large file (via B/s) access patterns at the same time.

With these quotas it is not possible to create flavours that fit a specific workload, as some important characteristics, such as latency, cannot be captured by those limiting parameters. It is therefore necessary to be able to offer storage options that can be more targeted to the use case. This ranges from using different technologies, such as conventional slow and fast spinning hard drives to solid state drives. These drives in return can be used in many different configurations that influence the respective performance. The configuration options include, among others, the storage I/O scheduler and the file system. It becomes even more challenging

*University College Cork, Ireland ({s.meyer, j.morrison}@cs.ucc.ie).

[1] http://ceph.com

when used in combination with a distributed file system, which seems to be the most common storage back-end used for OpenStack according to [10].

In our previous paper [14] we looked at the option of offering distinct storage pools using different file systems. In this paper we extend this work and look at the impact of changing a single parameter in the Ceph configuration on the performance when tested through an OpenStack cloud deployment using block storage devices. The OpenStack storage components will be described in Section 3 followed by an overview of the distributed file system Ceph in 4.

3. OpenStack Storage Components. OpenStack has three different storage services: Glance, Cinder and Swift. They cover three very different areas. The OpenStack image service Glance is an essential component in OpenStack as it serves and manages the virtual machine images that are central to Infrastructure-as-a-Service (IaaS). It offers an RESTful API that can be used by end users and OpenStack internal components to request virtual machine images or metadata associated with them, such as the image owner, creation date, public visibility or image tags.

Using the tags of the images should allow an automatic selection of the best storage back-end for an individual VM. When an image is tagged with an database tag the storage scheduler should be able to automatically select the appropriate pool to host the VM, like it does for other components, like the amount of CPU core or the memory capacity. In case the tag is absent the image will be hosted on the fall-back back-end, which uses a standard or non-targeted configuration that is used for general purpose scenarios. When users use the correct tag they will benefit from it and the operator will potentially be able to increase the number of users that he can host without risking overall storage performance degradation and different billing rates, as they are optimal solutions for specific workloads (value added features).

The OpenStack object storage service Swift offers access to binary objects through an RESTful API. It is therefore very similar to the Amazon S3 object storage. Swift is a scalable storage system that has been in use in production for many years at Rackspace[2] and has become a part of OpenStack. It is highly scalable and capable to manage petabytes of storage. Swift comes with its own replication and distribution scheme and does not rely on special RAID controllers to achieve fault tolerance and resilient storage. It can also be used to host the cloud images for the image service Glance.

The third storage system of OpenStack is the block storage service Cinder. Cinder is used to either create volumes that are attached to virtual machines for extra capacity that show up as a separate drive within the VM, but it can also be used directly as the boot device. In that case the image from Glance will be converted/copied into a Cinder volume. After it has been flagged as bootable it can be used as the root disk of the VM. The back-ends for Cinder cover a great variety of systems [4], including proprietary storage systems such as Dell EqualLogic or EMC VNX Direct, distributed file systems, such as Ceph or GlusterFS, and network shares using Server Message Block (SMB) or NFS.

Normally it would be necessary to have at least two dedicated storage systems available when all three storage services are desired. This does not allow the flexibility to deal with extra capacity demands on individual services as the hardware has to be partitioned when the system is rolled out. Currently there are two storage back-ends available that can be used for all three services within one system with the support for file-level storage, which is required for supporting live-migration between compute hosts. These are Ceph (see Section 4) and GlusterFS [3]. By using one of these storage back-ends it is possible to consolidate the three services on a single storage cluster and keeping them separated through logical pools. According to the OpenStack user survey in 2014 [10] Ceph has currently the highest popularity as a storage back-end, which will be looked at in more detail in the following section.

4. Introduction to Ceph. Ceph [16] has been designed under the assumption that a large peta-scale storage system is an incremental growing system, where the failure of components are not the exception but rather the norm, and where workloads are constantly changing. At the same time the storage system has to be able to handle thousands of user requests and deliver high throughput [17]. The system replaces the traditional interface to disks or RAIDs with object storage devices that integrate intelligence to handle specific operations

[2] www.rackspace.com

[3] www.gluster.org

themselves. Clients interact with the metadata server to perform operations, such as open and rename, while communicating directly with the OSDs for I/O operations. The algorithm that is used to spread the data over the available OSDs is called CRUSH [18]. From a high level, Ceph clients and metadata servers view the object storage cluster, that consists of possibly tens or hundreds of thousands of OSDs, as a single logical object store and namespace. Ceph's Reliable Autonomic Distributed Object Store (RADOS) [19] achieves linear scaling in both capacity and aggregate performance by delegating management of object replication, cluster expansion, failure detection and recovery to OSDs in a distributed fashion. The following list shows again the individual components and their role and function:

OSDs: A Ceph OSD Daemon (OSD) stores data, handles data replication, recovery, backfilling, rebalancing, and provides some monitoring information to Ceph Monitors by checking other Ceph OSD Daemons for a heartbeat. A Ceph Storage Cluster requires at least two Ceph OSD Daemons to achieve an `active + clean` state when the cluster makes two copies of your data (Ceph makes 2 copies by default, but it is adjustable).

Monitors: A Ceph Monitor maintains maps of the cluster state, including the monitor map, the OSD map, the Placement Group (PG) map, and the CRUSH map. Ceph maintains a history (called an "epoch") of each state change in the Ceph Monitors, Ceph OSD Daemons, and PGs.

MDSs: A Ceph Metadata Server (MDS) stores metadata on behalf of the Ceph Filesystem (i.e., Ceph Block Devices and Ceph Object Storage do not use MDS). Ceph Metadata Servers make it feasible for POSIX file system users to execute basic commands like `ls`, `find`, etc. without placing an enormous burden on the Ceph Storage Cluster.

Ceph is also highly customizable and offers many ways to change the configuration of the cluster. This is not just limited to the number of placement groups per pool (pgp), but spans more than 600 parameters. The value of these parameters will result in many cases from the use case, such as the number of replicas. Furthermore the storage system relies on components that are not part of Ceph directly, like the Kernel I/O scheduler and the corresponding queue length, that have an impact on the performance. How these can be used to support discrete storage pools will be discussed in Section 4.2 to 4.4.

4.1. OpenStack and Ceph. Creating pools within the cluster is necessary to allow access to the storage. These pools can in return be used for the OpenStack services Glance and Cinder. Running Swift with Ceph is also possible through the Rados gateway, which is part of Ceph, that offers a RESTful API to the objects in the cluster. As Glance is only hosting the images for the VMs it does not have any real performance requirements, especially when Cinder is used for actually running the VMs by choosing the `copy to volume` option when creating VMs. Furthermore, even though it is possible to just use Glance without Cinder and use it for booting the images, it does not offer the option to use multiple back-ends at the same time. It is a requested feature[4], but currently it is not available.

Cinder supports multiple back-ends that are attached at the same time. This offers the possibility to create different Cinder Tiers that are connected to different back-end systems with varying capabilities or features, such as having one proprietary storage system and a network share set up as the back-ends or to use different pools from Ceph or completely different Ceph clusters. In the first case the Ceph configuration file is identical and only the Ceph pool are different. In the second case it is necessary to provide multiple Ceph configurations that point to the different clusters, as the Ceph monitors will be running on separate hosts. This might be used to offer low specification versions that offer no resilience to failures for low cost solutions or high cost solid state drive solutions.

4.2. Optimization on a Cluster Level. Tuning the distributed file system on a cluster level gives access to whole range of parameters [3] of Ceph and the underlying components, such as the caching size, recovery settings, journal settings or logging. This allows for perfect adoption to a specific workload, but in contrast does not allow for differentiation and therefore multiple workloads in one cluster. When a decision is made it affects the overall system and can only be slightly tweaked by the parameters that are accessible through the pools (see 4.3).

[4] https://blueprints.launchpad.net/glance/+spec/multi-store

To offer distinct storage variants, the whole storage cluster has to be partitioned and multiple clusters have to be created. It is possible to run multiple clusters on individual hosts, but it has to be guaranteed that the services have unique IP and port combinations. Copying data from one cluster to the other cannot be achieved very easily on the Ceph level, but within OpenStack it is supported with specific limitations. It is not possible to migrate volumes that have snapshots.

4.3. Optimization using Pools and Tiering. When optimizing on a Ceph pool level without changing the Crushmap, the parameters that can be changed is limited in comparison to the options that are available when optimizing on a cluster level. In total there are 19 parameters (full list available at [13]) that change the characteristics of the pool.

Some of these parameters have a direct effect on the performance and behaviour of a storage pool, as they directly influence the pool characteristics. Parameters such as `size`, `min_size`, `pgp_num`, `crush_ruleset`, `hashpspool` and `crash_replay_interval` have a significant impact on how the pool distributes the data across the OSDs and how many copies are stored. They also directly influence reliability, stability and recovery.

Ceph offers the option to add a cache Tier to a pool. Adding fast expensive drives for highly frequented objects with slower cheaper disks that are acting as cold or slower storage. Such tiered systems are very common in enterprise storage systems, *e.g.* Dell Compellent [6], and is now available in software solutions, such as Ceph, as well. Changing the pool parameters effects the movement of the objects between the cache and normal Tier and the used caching algorithm.

The using a cache Tier it is necessary to select one of two operation modes. When using the writeback mode the client will write the data directly to the fast cache tier and will receive an acknowledgement when the request has been finished. Over time the data will be send to the storage tier and potentially flushed out of the cache tier. When a client requests data that does not reside within the cache, the data is transferred first to the cache tier and then served to the client. This mode is best used for data that is changeable, such as video, photo and transactional data.

When using read-only mode the cache will only be used for read access, where it will copy the data from the storage tier to server the read requests. Write access will be sent directly to the storage tier. This operation mode is best used for immutable, such as images and videos for webservices, DNA sequences or radiology images, as reading data from a cache pool that might contain out-of-date data provides weak consistency. For that reason it should not be used for mutable data.

Using pools for differentiation still relies on the underlying configurations, such as disk scheduler or OSD file system settings. All pools will share the same general settings with specific configuration changes for replication count or distribution enhancements. Therefore a pure pool based approach does not offer the best way to make significantly different storage configurations accessible. On the other hand, pools are a good way to partition the storage and to expose it to different users/services through the access control mechanisms of Ceph.

4.4. Heterogeneous Pools. The term heterogeneous pools in this case is used for pools that have different underlying components, such as the I/O scheduler or the file system. The I/O scheduler is set in the operating system for a specific block device, such as `/dev/sdb` and cannot be set for a specific partition. The scheduler comes along with it own settings, such as the scheduler type and the queue depth. Changing these can have a substantial influence on the performance of the file system under specific workloads [15].

The other component that has an impact on performance is the file system. Besides the different functionalities that they offer, they handle some patterns much better than others due to their internal design. Both of these aspect of a heterogeneous cluster configuration have been shown in our previous paper [14].

5. Experimental results. The experiments are focused on changing the whole storage cluster configuration. The storage pools used by OpenStack were deployed on the same hardware and software configuration. By changing the storage cluster configuration for each individual benchmark run shows the impact of each one of those individual changes in comparison to the default configuration.

5.1. Testbed. The configuration used for the practical evaluation consists of three Dell R610 servers connected to a directly attached storage expansion tray through an LSI SAS3444E SAS controller using SAS multi-lane cables, each. Each server is equipped with an Intel Xeon E5603 quad core processor with 1.6 GHz clock speed, 16 GB DDR3 memory and 8 Gigabit Ethernet ports. Each storage tray consists of twelve 1 TB harddrives (Ultrastar A7K1000 [8], Barracuda ES.2 [2], Western Digital RE4 WD1003FBYX [12]), of which only four of each tray will be used for the experiment, but all disks are part of the cluster. The Ceph cluster network uses NIC bonding (IEEE 802.3ad [9]) with four network links for increased bandwidth. The Ceph public network uses three links in the same way. The operating system used for the testbed is Ubuntu 14.04 LTS with the 14.04.2 hardware enablement stack (Ubuntu Utopic) which uses Linux kernel version 3.16, which includes many patches for BTRFS and XFS. The mounting options used for both file systems were the default settings of Ceph. The mounting settings can have an influence on the performance of a file system, but they are not in the focus for this paper.

To run the benchmarks against the Ceph cluster an OpenStack cloud (version Kilo) is used. The OpenStack cloud consists of a dedicated cloud controller and of three Dell R710 servers with two Intel Xeon E5645 hexa cores with 2.4 GHz clock speed and Intel Hyper Threading enabled acting as compute nodes. Each server is connected to the Ceph public network with three bonded Gigabit Ethernet links. The used switch is a Dell PowerConnect 6248 [5]. Jumbo frames were enabled on the switch and all Ethernet links.

The network bandwidth between the nodes is measured using `iperf`. The results are shown in Table 5.1 with the Storage network representing the Ceph cluster network and the Management network being the Ceph public network. The 2 bonded port configuration is used on the controller to access the Ceph cluster for handling the Cinder and Glance volumes and images whereas the 3 bonded ports are used on the compute nodes to run the virtual machines directly off the Cinder volumes.

5.2. Cluster configuration. The Ceph cluster is configured to host multiple pools, pinned to different drives. The pool used for the benchmarks is isolated on the 12 Western Digital RE4 drives. To reduce the impact of the cluster network bandwidth limitation of about 3 Gb/s shown in Table 5.1 the replication count is set to 2. This ensures that each block is transferred only once through the cluster network for replication. With a replication count of three the file would be written to two other hosts which would double the network traffic and therefore reduce the write performance. In a bigger cluster the load from replication is spread and therefore the network bandwidth dependency will be less crucial overall, but in a small cluster it is a limiting factor.

The Ceph configuration for these experiments is left to the default settings and the parameters that are set can be seen in the following configuration snippet. It has the debugging and reporting function on the OSD and Monitors disabled and uses CephX for authentication.

```
[global]
osd_pool_default_pgp_num = 1024
osd_pool_default_pg_num = 1024
osd_pool_default_size = 2
osd_pool_default_min_size = 2

[client]
rbd_cache = false

[osd]
debug ms = 0
debug osd = 0
debug filestore = 0
debug journal = 0
debug monc = 0

[mon]
debug ms = 0
debug mon = 0
debug paxos = 0
debug auth = 0
```

The specific cluster wide parameters tested differ from the default configurations in exactly one parameter. This allows to detect parameters being harmful or beneficial for specific workloads. The tested parameters are listed in Table 5.2.

The selected parameters are all for settings related to the OSDs and the filestore. Using Ceph in combination

Table 5.1
Measured (`iperf`) bandwidth on the different networks.

Network	Storage	Management	
Bonded Ports	4	2	3
Bandwidth	3.08 Gb/s	1.96 Gb/s	2.50 Gb/s

Table 5.2
Tested parameter values and their default configuration.

Parameter	Default	Tested
osd_op_threads	2	1 (B), 4 (C), 8 (D)
osd_disk_threads	1	2 (E), 4 (F), 8 (G)
filestore_op_threads	2	1 (H), 4 (I), 8 (J)
filestore_wbthrottle_xfs_bytes_ start_flusher	41943040	4194304 (K), 419430400 (L)
filestore_wbthrottle_xfs_ios_start_ flusher	500	5000 (M), 50 (N)
filestore_wbthrottle_xfs_inodes_ start_flusher	500	5000 (O), 50 (P)
filestore_queue_max_bytes	104857600	1048576000 (Q), 10485760 (R)
filestore_queue_committing_max_ bytes	104857600	1048576000 (S), 10485760 (T)
objecter_inflight_op_bytes	104857600	1048576000 (U), 10485760 (V)
objecter_inflight_ops	1024	8192 (W), 128 (X)

with OpenStack Cinder and Glance does not require using components such as the RADOS Gateway, which would be necessary when using OpenStack Swift, or the metadata server (MDS).

The `osd_op_threads` parameter sets number of threads to service Ceph OSD Daemon operations. When set to 0 it will disable multi-threading. Increasing the number may increase the request processing rate. Ceph uses a 30 seconds timeout for the requests. So it depends on the used hardware if altering the parameter has a positive or a negative effect.

`osd_disk_threads` sets the number of disk threads, which are used to perform background disk intensive OSD operations. These include scrubbing, which is analogous to `fsck` on the object storage layer, and snapshot handling. This can effect the performance when the scrubbing process happens while there is concurrent data access. Otherwise only one operation can be worked on at one time.

`filestore_op_threads` sets the number of file system operation threads that execute in parallel.

`filestore_wbthrottle_xfs_bytes_start_flusher`, `filestore_wbthrottle_xfs_ios_start_flusher` and `filestore_wbthrottle_xfs_inodes_start_flusher` configure the filestore flusher that forces data from large writes to be written out before the sync in order to (hopefully) reduce the cost of the eventual sync. Previously, the filestore had a problem when handling large numbers of small ios. Ceph throttles dirty data implicitely via the journal, but a large number of inodes can be dirtied without filling the journal resulting in a very long sync time when the sync finally does happen. The flusher was not an adequate solution to this problem since it forced writeback of small writes too eagerly killing performance.

`filestore_queue_max_bytes` and `filestore_queue_committing_max_bytes` set the size of the filestore queue and the amount of data that can be commited in one operation.

`objecter_inflight_op_bytes` and `objecter_inflight_ops` modify the Ceph objecter, which handles where to place the objects within the cluster.

5.3. Performance evaluation. To evaluate the impact of the different configurations the Ceph cluster is exercised using the OpenStack cloud system described in Section 5.1. As the benchmark the tool `fio` [1] is used. The benchmark settings are set to 300 seconds runtime and a 10GB testsize. The IO engine is set to sync, which uses fseek to position the I/O location. Access is set to direct and buffering is disabled. For each run there is a start and a ramp delay of 15 seconds. Random and sequential access patterns are tested for both reads and writes. Block sizes to test are 4k, 128k, 1MB and 32MB. A total of 9 runs for each benchmark configuration is executed to achieve a representative average over multiple runs.

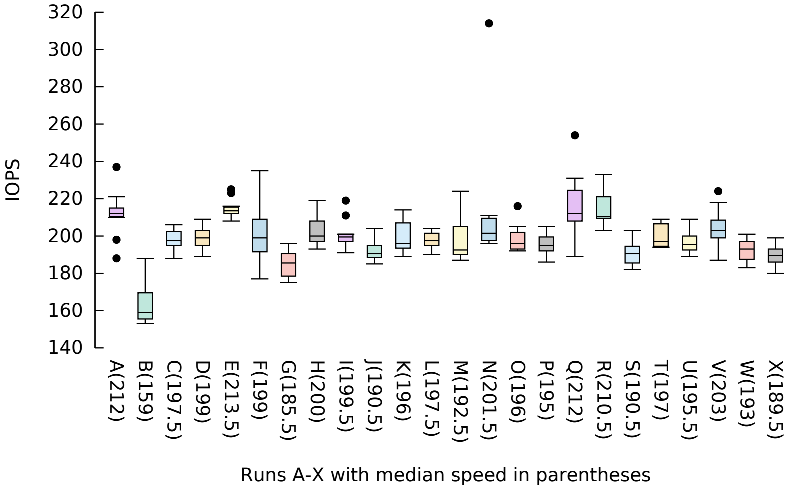

Fig. 5.1. *FIO random read 4k.*

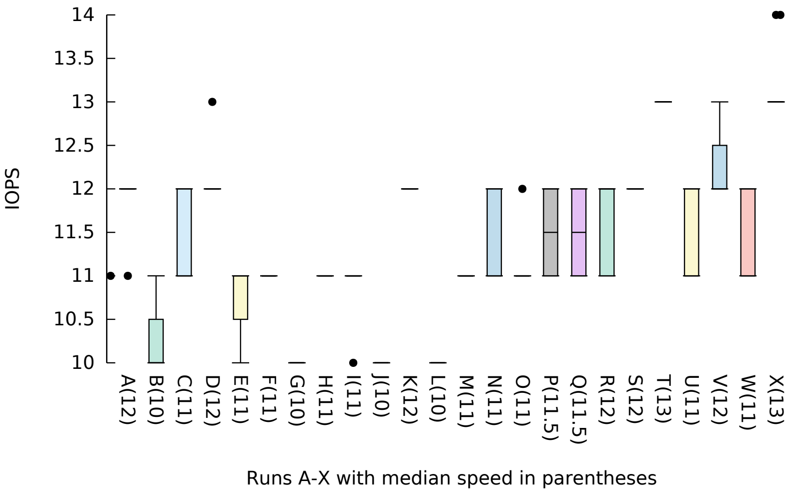

Fig. 5.2. *FIO random write 4k.*

A total of 12 virtual machines, equally distributed across the three compute hosts, is used to stress the system. Each VM is set to use 4 cores and 4GB of memory. The virtual disk is set to use a 100GB Cinder volume. Rados Block Device (RBD) caching is disabled on the Ceph storage nodes and on the compute hosts in the QEMU/KVM hypervisor settings. The diagrams show the mean value across all 12 VMs and their 9 runs.

5.3.1. 4k. The smallest file block size tested is 4k. The performance of mechanical hard drives suffers quite a lot under such loads, whereas SSDs perform much better well under such loads. The unit used to in these graph is IOPS (**I**nput/**O**utput Operations **P**er **S**econd).

Figure 5.1 and 5.2 show the performance under random access workloads for reading and writing. Under random read workloads the most disruptive configuration is with `osd_op_threads=1` (B) with 159 IOPS. In comparison to the other configurations that achieve between 189 (X) and 213.5 (E) IOPS this configuration performs 15% lower than the second lowest and 25% less than the default configuration. In comparison to the default configuration the performance can only be matched, but not be surpassed. For random writes the performance is more even with a difference of 3 IOPS between the best and the worst performing configuration.

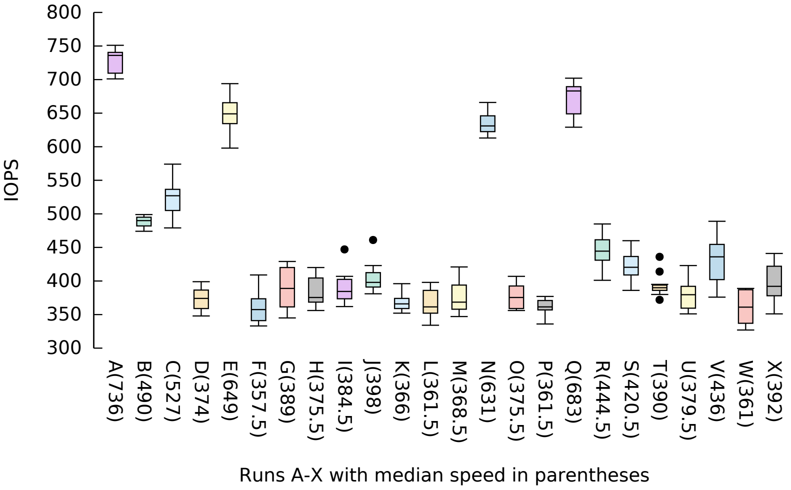

Fig. 5.3. *FIO sequential read 4k.*

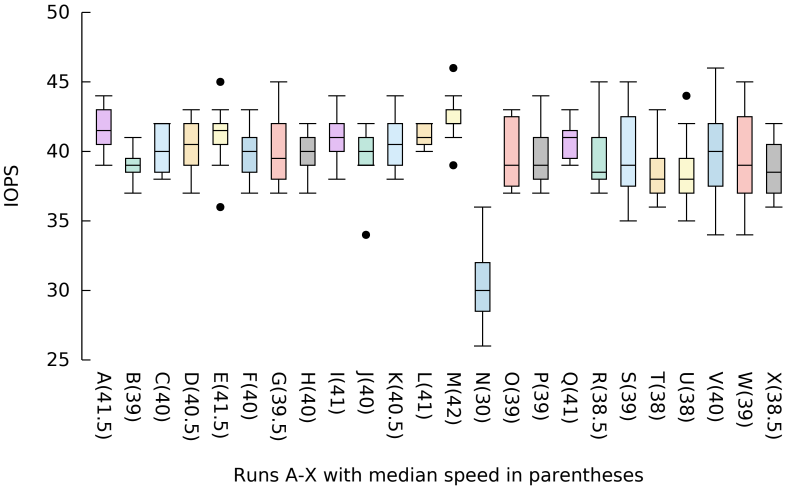

Fig. 5.4. *FIO sequential write 4k.*

Due to the low number of IOPS there can't be any real conclusions about the impact of the parameters.

When the storage system is tested against 4k sequential read access (see Figure 5.3), the difference between the lowest performing configuration (`osd_disk_thread=4` (F)) and the highest (default configuration (A)) is over 105% or 378 IOPS. Well performing configurations, but worse than the default, are `filestore_queue_max_bytes` =1048576000 (Q), `osd_disk_threads=2` (E) and `filestore_wbthrottle_xfs_ios_start_flusher=50` (N). For writing (see Figure 5.4) the results are very even again, except for configuration N, that scores well for read accesses. When this configuration is used the performance drops by 25% in comparison to the others. This means when writing small blocks the small flusher threshold is harmful, whereas it is beneficial when used for reading.

5.3.2. 128k. When using 128k block sizes for random read accesses (see Figures 5.5) all but two configuration (B and N) achieve gains over the default configuration. A maximum gain of 8% is observed for configuration K. When writing random blocks with the same block size the difference is much more significant with K being 14% faster than the default. The performance difference between `filestore_wbthrottle_xfs_bytes_start_`

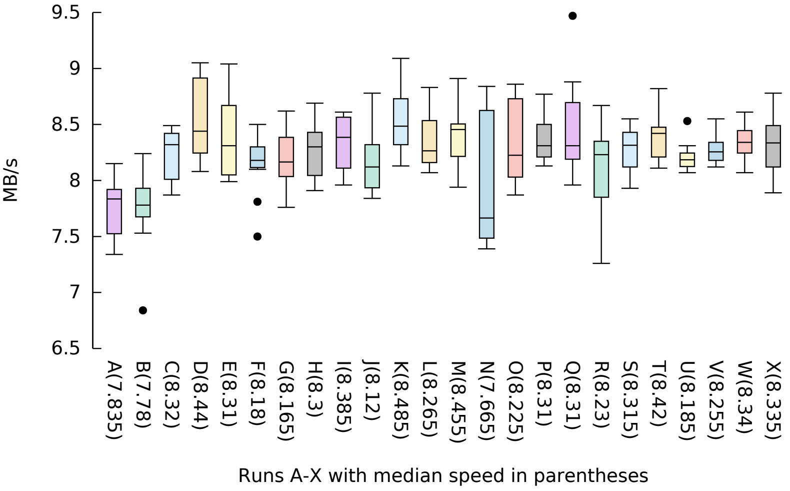

Fig. 5.5. *FIO random read 128k.*

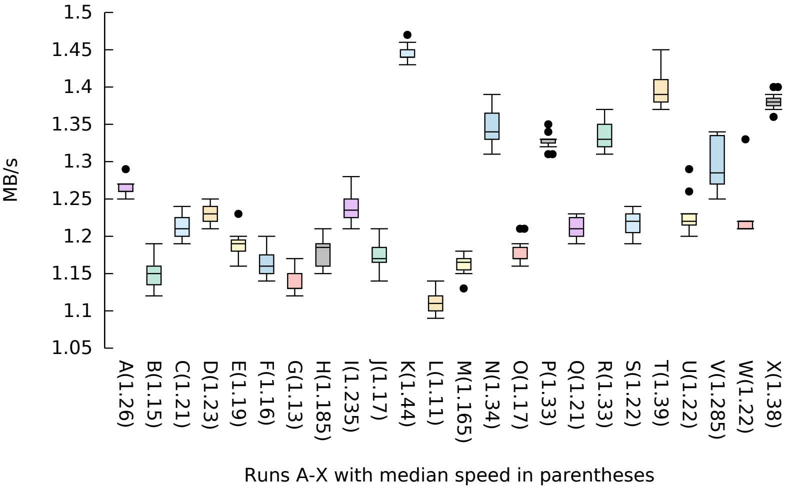

Fig. 5.6. *FIO random write 128k.*

`flusher` being 4194304 (K) and 419430400 (L) is almost 30%. The interesting part is that it is the same parameter with different values that changes the behaviour. A similar behaviour can be observed for the pairs M to X where the smaller value performs better than the larger. The default configurations performance for these runs is always in between the lower and higher configurations.

For sequential read access (Figure 5.7) the performance is very even with a difference of 9% between configuration O (lowest) and Q (highest). The default configuration is surpassed by only four configurations. For sequential write access (Figure 5.8) configurations N and K are the most disruptive, with K performing 16.5% lower than the default configuration. Gains are not observed under this workload.

5.3.3. 1MB. For random 1MB block access (Figure 5.9) `disk_threads=4` (F), `filestore_queue_max_bytes` equals 10485760 (R) and 1048576000 (Q) perform better than the rest. Configuration F is able to improve performance by 10% over the default. The most disruptive configuration is `filestore_op_threads=8` (J) that reduces performance by 13%. For random write access (Figure 5.10) configuration `filestore_wbthrottle_xfs_bytes_start_flusher=4194304` (K) improves the performance by 44.5% over the default configuration. Only configuration Q shows reduced performance. All other configurations improve performance.

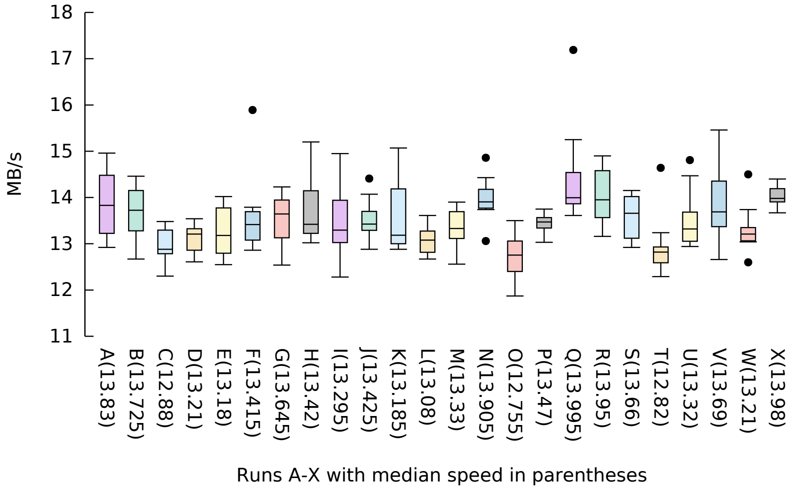

Fig. 5.7. *FIO sequential read 128k.*

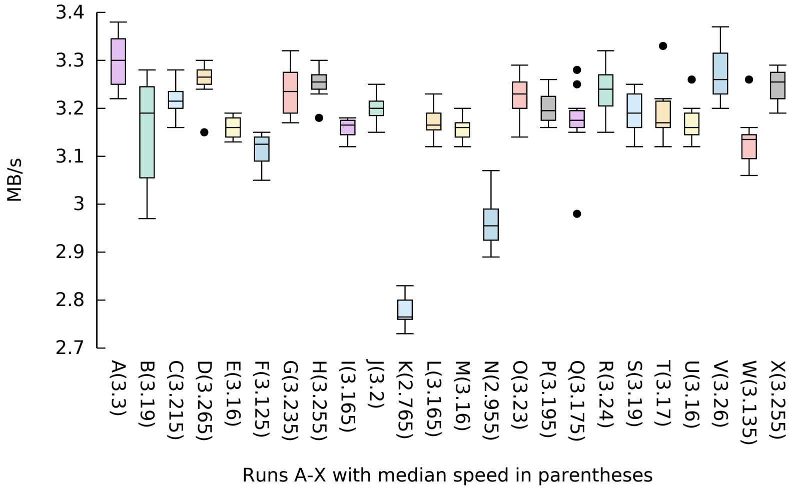

Fig. 5.8. *FIO sequential write 128k.*

Under sequential 1MB read workloads (Figure 5.11) there is no configuration that improves performance. The maximum regression observed is 8% (W). For writing (Figure 5.12) configuration K show the highest gains of 28%.

5.3.4. 32MB. In random 32MB read and write access loads, see Figures 5.13 and 5.14, configuration `filestore_queue_max_bytes=1048576000` (Q) are able to improve performance over the default configuration. For random writes the same parameter with a hundred times smaller queue size (R) is able to surpass it, but not for random reads. Configuration `filestore_wbthrottle_xfs_inodes_start_flusher=5000` (O) is the most disruptive for random reads, reducing performance by more than 2 MB/s in comparison to the default configuration. For random writes multiple configurations (E, H, O) have a strong negative impact. In comparison to the default configuration changing the parameters for 32MB random access workloads is more harmful than useful.

For sequential 32MB read access (see Figure 5.15) all configurations that deviate from the default reduce the performance by up to 14% (C) or 2.2 MB/s. Configurations B and Q reduce performance the least. With

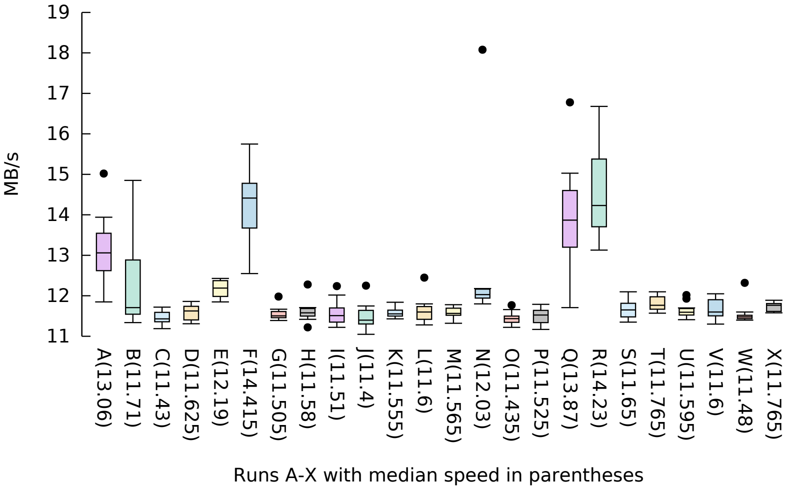

Fig. 5.9. *FIO random read 1MB.*

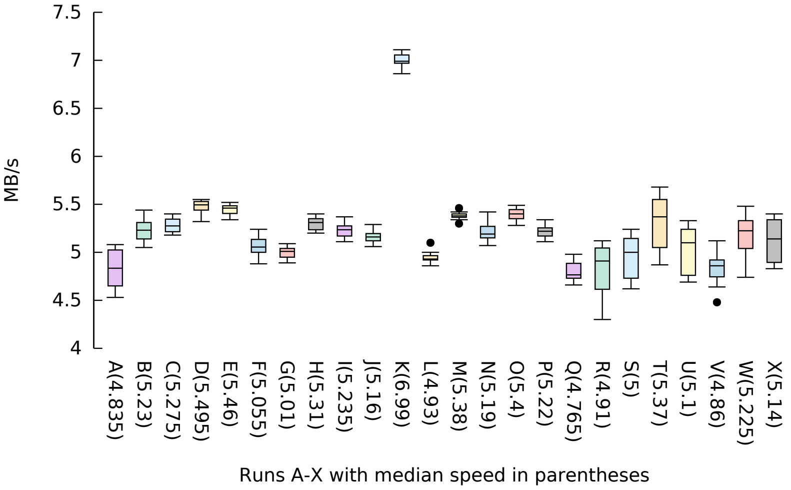

Fig. 5.10. *FIO random write 1MB.*

sequential writes (see 5.16) the results look a bit different. Configuration R can improve performance by 6%, whereas the other configurations reduce performance by up to 14.5% (L).

6. Conclusion. The experiments show that changing a single parameter of a Ceph cluster configuration can have a significant impact on performance when used as a storage back-end for virtual machines. Under specific workloads an improvement of up to 44.5% was observed. In other access patterns the improvement is not that significant, but still an improvement over the default configuration.

Configurations that showed performance degradations of up to 50% under certain workloads have also been observed. The severity of losses in comparison to the default configuration are heavily depending on the workload.

Overall the performance of the default configuration is well balanced. The other configurations were not always able to achieve better performing. Only with some access patterns the majority of the tested configurations was able to improve performance. This shows that with the used hardware configuration the default settings are working well, but leave room for improvement and are not the best possible.

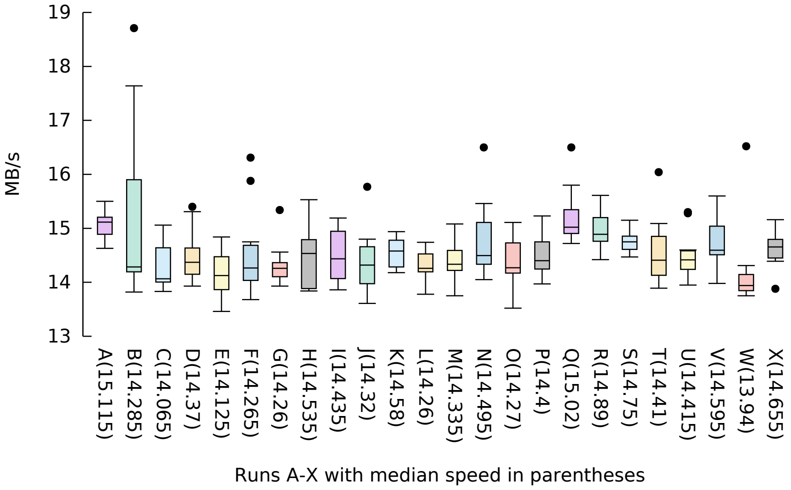

Fig. 5.11. *FIO sequential read 1MB.*

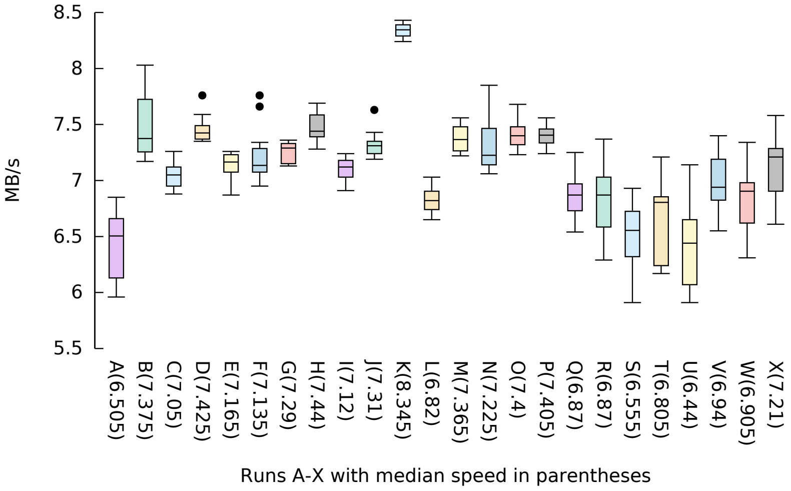

Fig. 5.12. *FIO sequential write 1MB.*

7. Acknowledgement. The research of the first author is supported by the enterprise partnership grant EPSPG/2012/480 from IRCSET and Intel Ireland Ltd.

The research of the second author is supported by the CloudLightning project, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 643946.

REFERENCES

[1] *axboe/fio GitHub*, `https://github.com/axboe/fio` (accessed 2015-12-14).

[2] *Barracuda ES.2 serial ATA*, `http://www.seagate.com/staticfiles/support/disc/manuals/NL35%20Series%20&%20BC%20ES%20Series/Barracuda%20ES.2%20Series/100468393f.pdf`(accessed 2015-02-11).

[3] *ceph/config_opts.h*, `https://github.com/ceph/ceph` (accessed 2015-01-09).

[4] *CinderSupportMatrix OpenStack*, `https://wiki.openstack.org/wiki/CinderSupportMatrix` (accessed 2014-02-12).

[5] *Dell PowerConnect 6200 series switches*, `http://www.dell.com/downloads/emea/products/pwcn/PowerConnect_6200_spec_sheet_new.pdf` (accessed 2015-03-18).

[6] *Dell storage SC series*, `http://www.dell.com/us/business/p/dell-compellent` (accessed 2015-05-25).

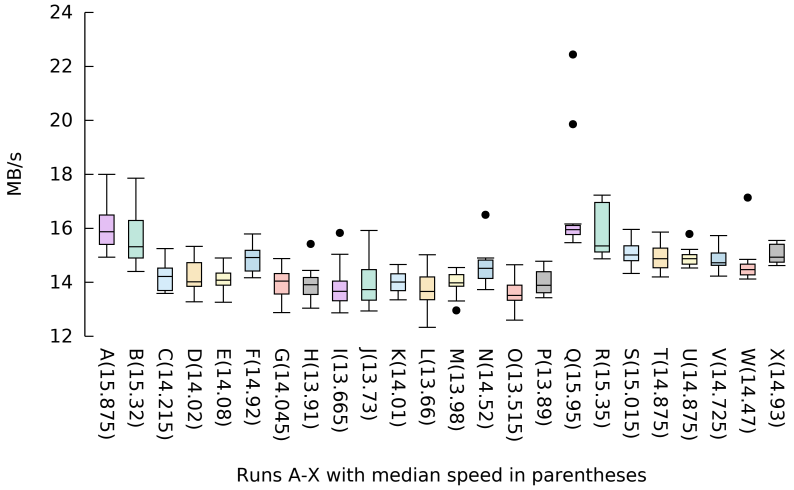

Fig. 5.13. *FIO random read 32MB.*

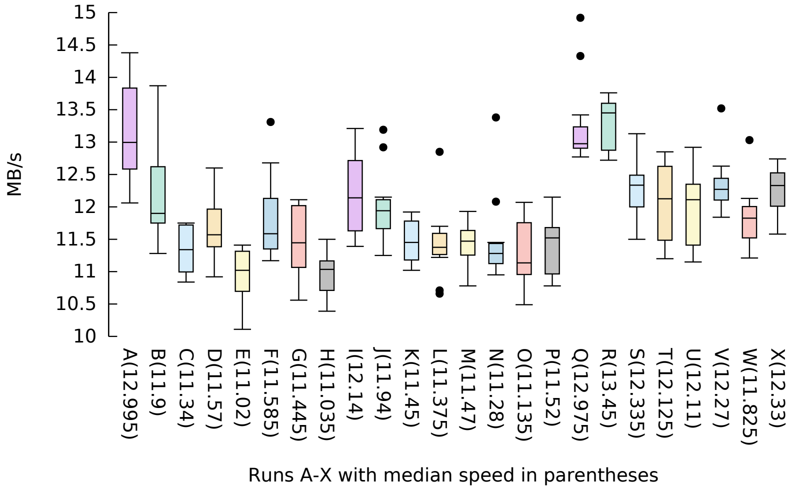

Fig. 5.14. *FIO random write 32MB.*

[7] *A guide to the OpenStack kilo release*, `https://developer.ibm.com/opentech/2015/04/30/a-guide-to-the-openstack-kilo-release/` (accessed 2015-07-01).

[8] *HitachiUltrastar a7k1000*, `http://www.hgst.com/tech/techlib.nsf/techdocs/DF2EF568E18716F5862572C20067A757/\$file/Ultrastar_A7K1000_final_DS.pdf` (accessed 2015-02-11).

[9] *IEEE standard for local and metropolitan area networks–link aggregation*, pp. 1–163, `http://dx.doi.org/10.1109/IEEESTD.2008.4668665` doi:10.1109/IEEESTD.2008.4668665.

[10] *OpenStack user survey insights: November 2014*, `http://superuser.openstack.org/articles/openstack-user-survey-insights-november-2014` (accessed 2015-03-03).

[11] *Stackalytics | OpenStack community contribution in kilo release*, `http://stackalytics.com/?release=kilo` (accessed 2015-07-01).

[12] *WD RE4 series disti spec sheet - 2879-701338.pdf*, `http://www.wdc.com/wdproducts/library/SpecSheet/ENG/2879-701338.pdf` (accessed 2015-12-17).

[13] I. Inktank Storage and contributors, *Pools ceph documentation*, `http://docs.ceph.com/docs/master/rados/operations/pools/` (accessed 2015-07-07).

[14] S. Meyer and J. P. Morrison, *Supporting heterogeneous pools in a single ceph storage cluster*, in 2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), pp. 352–359, `http://dx.doi.org/10.1109/SYNASC.2015.61` doi:10.1109/SYNASC.2015.61.

[15] S. Pratt and D. A. Heger, *Workload dependent performance evaluation of the linux 2.6 i/o schedulers*, in 2004 Linux

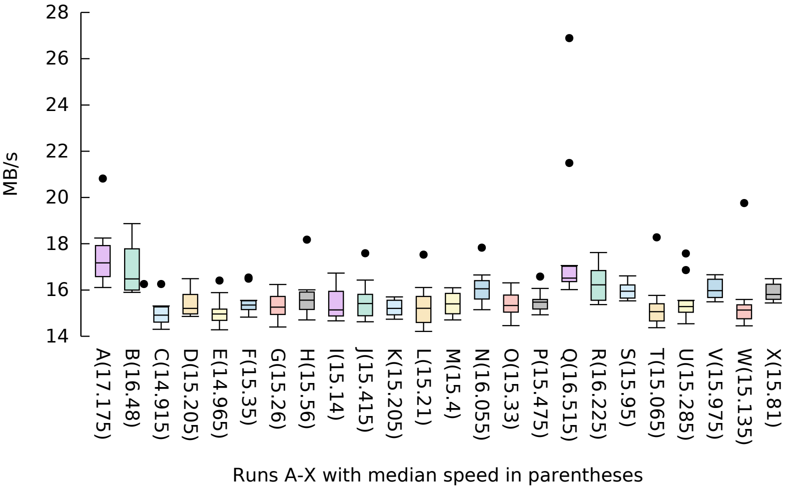

Fig. 5.15. *FIO sequential read 32MB.*

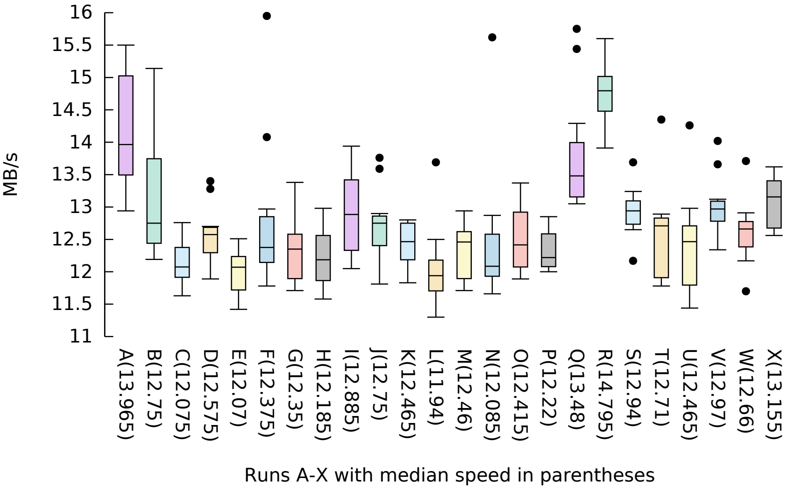

Fig. 5.16. *FIO sequential write 32MB.*

Symposium, `http://landley.net/kdocs/mirror/ols2004v2.pdf#page=139` (accessed 2015-06-30).

[16] S. A. Weil, *Ceph: Reliable, scalable, and high-performance distributed storage*, 2007, `http://pdf-release.net/external/4047524/pdf-release-dot-net-weil-thesis.pdf` (accessed 2014-03-26).

[17] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, *Ceph: A scalable, high-performance distributed file system*, in Proceedings of the 7th symposium on Operating systems design and implementation, USENIX Association, 2006, pp. 307–320, `http://dl.acm.org/citation.cfm?id=1298485` (accessed 2014-03-26).

[18] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, *CRUSH: controlled, scalable, decentralized placement of replicated data*, in Proceedings of the 2006 ACM/IEEE conference on Supercomputing, ACM, 2006, p. 122, `http://dl.acm.org/citation.cfm?id=1188582` (accessed 2014-03-26).

[19] S. A. Weil, A. W. Leung, S. A. Brandt, and C. Maltzahn, *Rados: a scalable, reliable storage service for petabyte-scale storage clusters*, in Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07, ACM, 2007, pp. 35–44, `http://dl.acm.org/citation.cfm?id=1374606` (accessed 2014-03-26).

Edited by: Dana Petcu
Received: May 11, 2016
Accepted: July 17, 2016