Improving Storage Performance with bcache in a Virtualization Scenario

Konrad Meier[1], Martin Ullrich [2], Dirk von Suchodoletz[3]

Abstract: The development of hard disk technology has not kept up with the general speed-up of computers and modern day technological developments like virtualization and cloud deployment. Flash storage, on the contrary, improves access speed and bandwidth significantly but at comparably higher costs. As in most scenarios only a fraction of the actual data stored on disk is ever used in arbitrary operation, a combination of both storage technologies can offer a good balance of speed and capacity. This resulted in tiered storage solutions offered by a range of commercial system providers. If the procurement of a new storage system is out of question, a Linux based flash cache approach can significantly improve performance of existing systems with moderate efforts in terms of setup and configuration.
This paper describes how an overloaded storage system primarily used as NFS based storage system for virtualization hosts could be brought back to operation by a multilayer data caching strategy to overcome high latencies and a low throughput. The resulting system design combines the benefits of high capacity slow spinning disk storage with flash only storage to a hybrid system. The presented approach was tested and evaluated in a real world productive environment. Various performance measurements and data are presented.

1 Introduction

Spinning magnetic disks were the backbone of permanent storage in computer technology for more than forty years. While the capacity was increased from some few Megabytes in the beginning to a several TeraBytes in modern day disks, this unparalleled development was not matched in terms of increased bandwidth and reduced access latency. Quite some effort was put into increasing the bandwidth and access times[4] of single disks, but the current technology has hit certain physical barriers. In particular, the access times have not changed significantly in the development of recent years [Wo09]. Storage manufacturers undertook quite some research to overcome those limits by combining multiple disks in complex configurations. But the exponential increase in storage demand and new applications like virtualization, cloud technology or big data easily outrun these developments. These applications usually generate random IO patterns, which are not handled well above a certain threshold by most traditional storage systems and appliances. This issue is fairly independent of whether directly attached disks, network attached storage (NAS) or storage area networks (SAN) are deployed. Especially, the additional layer of a network filesystem

[1] Computer Center, University of Freiburg, konrad.meier@rz.uni-freiburg.de
[2] Computer Center, University of Freiburg, martin.ullrich@rz.uni-freiburg.de
[3] Computer Center, University of Freiburg, dirk.von.suchodoletz@rz.uni-freiburg.de
[4] For a standard 15k RPM disk (e.g. Seagate Savvio 15K.3 300GB) the average latency is 2.0 ms and the disk seek-time for read operations is 2,6 ms. On everage this gives about 217 IOPS; The sequential read bandwidth depends on the position on the disk (outer edge or inner edge) and varies between 202 and 151 MByte/s.

as used with NAS can add to the problem. Nevertheless, a significant advantage of the network based solutions is a fast and easy live migration of virtual machines and a centralized data management.

In virtualization environments where a large number of virtual machines access files on the same NAS or SAN systems, traditional spinning disks are heavily challenged by the generated IO patterns. The reason is that each VM reads and writes into a different (large) file of the same storage. From the view-point of the storage system, the observed IO pattern is highly unpredictable.
Flash storage (in the form of solid state drives (SSD)) looks like the natural solution to this challenge as a single disk can easily handle multiple times of the bandwidth of spinning disks and especially since it is much more potent regarding IOPS[5] - 217 IOPS of an enterprise harddisk[6] compared to 28k (write) and 97k (read) IOPS of a SATA attached flash disk[7] (even more for special purpose devices like PCIe attached flash storage). Nevertheless, even with the evident decrease in price per Gigabyte, flash still cannot challenge magnetic hard-drives in terms of capacity.

Looking at the amount of accessed blocks on a storage system within a certain time frame reveals that most of the data is cold, meaning seldomly or never read. Only a small percentage is often accessed. Thus, transferring all data to flash is not required at all, but a proper strategy is needed to send hot data blocks to flash and keep cold blocks on magnetic mass storage. The state of the art solution to this problem is so-called tiered storage. It uses SSDs for caching to buffer data before being sent to the spinning drives [By12] [Ch15]. Actively used data is cached in flash, and cold-data is stored on spinning hard disks.

As many traditional and often expensive storage systems are still in use and cannot easily be replaced by betteperforming systems before their planned end of life, a cost-efficient but effective approach to bridge this time is required by many computer centers.

We evaluated a comparably novel multilayer caching strategy, which includes the use of memory-caching and SSD-caching. For the SSD-caching we used the block layer cache bcache[8] for which not much long term experience exists. The proposed solution is based on free open source software and utilizes commodity hardware. It is therefore usable in a wide range of applications. For example for applications like parallel storage in HPC[9], where large amounts of data, in some research workflows easily hundreds of TeraBytes, need to be stored and accessed for high performance processing. Back-end storage for cloud scenarios can be expected to profit as well.

[5] Input/Output Operations Per Second, an often used performance metric for storage devices.
[6] Seagate Savvio 15K.3 300GB; For calculation see footnote 4
[7] Samsung Enterprise SSD SM863 SATA 960GB
[8] bcache: Linux kernel block layer cache; [bc]
[9] HPC: High-Performance Computing

2 Initial Situation

The initial system without caching is given in Figure 1. As storage back end a system equipped with 56 x 3TB HDD and 4 x SSDs is used.[10] The SSDs are used as a read cache. The storage provided by the system is internally configured as a Dynamic Disk Pool [De15] and then exported as a LUN[11] over Fibre Channel. Internally every LUN is mapped to one of the two RAID controllers. The NFS server connects to the Storage System by two redundant 8 GBit/s wide Fibre Channel links. The system is using Fibre Channel multipathing provided by the Linux Device-Mapper. The imported block storage is mounted by the NFS server as an *ext4* filesystem to provide various NFS exports for the ESX Cluster over 10 GBit/s Ethernet. On the VMware ESX-Cluster[12] we have 150+ running VMs on five servers. Each host connects to the NFS server over NFSv3 and uses multiple 10 GBit Ethernet connections for redundancy[13].
The chosen setup was easy to use since new VMs could effortlessly be provided and automatically placed on the NFS datastore. The general challenge of this configuration lies in

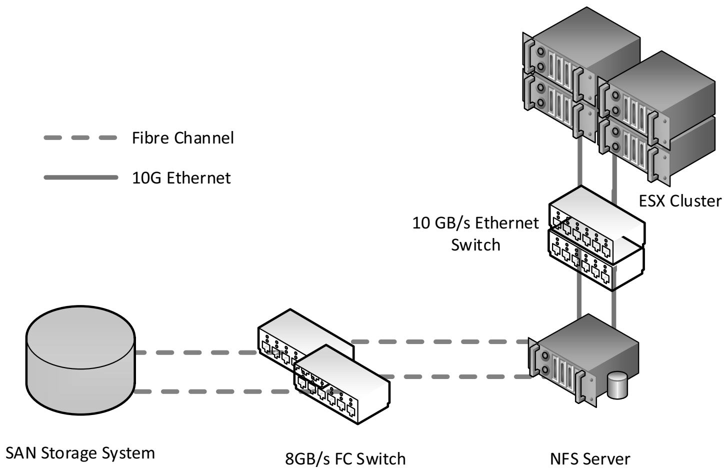

Abb. 1: Structure of the Virtualization Environment with shared NFS-based NAS

the complexity of the layered filesystem and block device setup. Within the VMs the OS is writing to a file which translates to a block IO like it would do in a physical machine. Because of virtualized hardware this block IO is translated to an IO on an open file on the storage handled by the virtualization hypervisor. This finally triggers a block IO on the network attached storage system.

[10] The system used is a DELL PowerVault-MD3660f dual-controller (active/active). The system was developed by LSI-Engenio (today NetApp) and is based on SANtricity. A widely available block storage product of similar features is offered by a number of manufacturers.

[11] LUN: A logical unit number identifies physical or virtual storage devices

[12] In this setup we used VMware ESX 5.5

[13] LACP configuration on the server and switches

A slow or delayed IO on the storage system looks like a failing block device to the operating system running inside a VM. At a certain threshold the OS is remounting the filesystem read-only (for data integrity reasons). To mitigate that scenario VMware tools (or open-vm-tools) usually increase SCSI timeouts[14] within the VM to prevent that a filesystem is remounted read-only too early. To save disk space, the container files are created using thin provisioning. Those files are constantly growing if a VM writes data to its disk. Since multiple containers are increasing in size parallelly, the container files are fragmented over time. This means that even sequential IO patterns from inside the VM do not necessarily write sequentially to the underlying storage system. For a storage system with spinning magnetic disks, fragmentation is therefore a problem since a lot of seeking operations are required to access data.

Challenges also arise from the NFS server side. In favor of performance the NFS client is often configured in *async* mode. This means that writing operations are acknowledged but not necessarily written back to storage system. Instead they are queued in RAM on the server. The opposite operation mode *sync*, ensures that every block is written, but significantly adds to latency as it has to wait for the underlying storage layer to complete the operation.
If the storage server is becoming congested, a long queue of blocks to be written builds up in the NFS server. This results in a large wait IO load that can stall the system entirely. It even might result in lost outstanding IO operations and the container files become corrupted. This corruption is much more dangerous than single files not properly written, as the files on the storage system are block devices to the virtual machines. Lost block writes may destroy filesystems and may render VMs unbootable.

The read and write pattern of the Virtualization environment is highly random. Every single VM generates only a small number of IO operations but in sum over all VMs the number of operations fluctuates between 1000 and 3000 IOPS.[15] Different problems were identified. Some VMs were configured with insufficient amount of RAM and started swapping to the storage system. Increasing the amount of RAM and removing the swap partition improved the situation to a certain degree. Moreover, the log file verbosity of some VMs generated a significant amount of (unnecessary) IO operations. In most cases it was no problem to set a less verbose logging behavior of the services.
The lack of a centralized backup system for the virtualization environment made it desirable that every VM itself writes a backup to an external backup system. This again generates a substantial amount of IOPS, especially at night during the backup runs.

The high number of IO operations leads to high IO latencies for some IO operations and could be measured within the virtual machines. An example output from the tool *fio*[16] looks like this:

[14] scsi device timeout: e.g. */sys/block/sda/device/timeout*; also refer to vmware knowledge base: http://kb.vmware.com/kb/1009465

[15] These values were readout directly from the storage system with the management tools provided by the manufacturer.

[16] Fio: Flexible I/O Benchmark; fio configuration: randread, size=8192MB; http://git.kernel.dk/?p=fio.git;a=summary

```
    lat (usec) : 250=26.96%, 500=72.49%, 750=0.29%, 1000=0.07%
    lat (msec) : 2=0.09%, 4=0.08%, 10=0.01%, 20=0.01%, 50=0.01%
    lat (msec) : 100=0.01%, 250=0.01%, 500=0.01%, 750=0.01%, 2000=0.01%
    lat (msec) : >=2000=0.01%
```

Most of the operations (>99%) are handled within one millisecond or less. But some operations require significantly more time. In some cases an operation even takes several seconds to be processed. In VMs this repeatedly led to long wait IO phases in which the VM had to wait for requested blocks. The reason for this is that for some IO patterns the spinning hard disks were not able to reposition and respond in time, causing high latencies for some IO requests in the storage system. With tools like *iotop*, *htop* and *vmstat* it was possible to analyze the situation on the NFS server. For example, a high amount of wait IO load is visible in *htop* and *vmstat*. The load was generated by the NFS kernel server waiting for block requests from the storage system.

3 Available Caching Solutions

For a SSD caching setup based on Linux, different approaches have been implemented over the last years:

- Flashcache [Fa]: Initially developed by Facebook and released as open source in 2011. It is implemented on top of the Linux kernel Device-Mapper of the Linux kernel.
- EnhanceIO [ST]: A fork based on Flashcache. In development since 2013. It implements a new write-back engine and is not based on the device mapper.
- dm-cache [dm]: Also based on the Linux kernel Device-Mapper framework. dm-cache was merged into the linux mainline kernel with version 3.9 (April 2013). It requires three devices to operate, a storage device, a caching device and a metadata device.
- bcache [bc]: Based on Linux kernel block layer. Bcache was merged into Linux mainline kernel with version 3.10 (June 2013). It is easy to use and configurable during operation through the proc filesystem. For example it is configurable if sequential IO is cached or directly written to the storage system.

Several authors have compared the different implementation approaches. The effect of Flashcache and bcache on I/O performance is evaluated in the work of Chritopher Hollowell et al. [Ho14] for a High-Energy Physics computing setup. In a white paper from Dell [AR13] the authors compare bcache and dm-cache as two caching solutions available in the Linux mainline kernel.

The possibility to cache write operations (write-back) is important in our case since random write operations were congesting our storage system. This is supported by all of the listed software solutions. To avoid a conflicting configuration, we decided not to use dm-cache because we already used the Device-Mapper for our Fibre Channel multipath configuration. Instead we decided to use bcache because it is already in the kernel and is based on the Linux block layer.

4 Caching Setup

To overcome the poor IO performance within the virtualization environment a new NFS server with a multi-layer caching strategy was deployed. Two objectives were pursued. First, reducing the read operations in the back end storage system through a bigger read cache in the NFS server. Second, aggregating the random write operations to a more linear write pattern, by using a write cache in the NFS server.
For the first objective the amount of RAM in the storage node was increased significantly by the factor of 8 to 256 GByte. The RAM is only used as a read-cache because of its nonpermanent nature. Otherwise, in the case of power loss, data stored in RAM and not yet written would be lost.
For the second objective, Linux *bcache* is used as a read and write cache. Bcache is a kernel block layer cache, that allows to use SSDs as cache for slower hard drives. Bcache can be used as read and writeback cache at the same time.

For the new NFS server with bcache a complete new machine with the following specification was deployed: 2 x 8 Core Intel E5-2640v3 2.6 GHz; 256 GByte DDR4 RAM; Ethernet: 2 x 10 GBit/s; Fibre Channel: 2 x 8 GBit/s Fibre Channel Host Adapter; bcache Storage: 4 x 960 GByte Samsung SM863 Enterprise SATA SSD;

The additional CPU cores in the new server are used to start a high number of NFS server threads. In comparison to the old system we increased the number of server threads from 8 to 256. This allows to handle more NFS requests parallelly.
The two Ethernet interfaces are configured as LACP[17] network device. A RAID-Controller is used to configure the four SSD Drives as RAID 10[18]. RAID 10 was used because it is the best compromise between storage capacity, redundancy and performance. Choosing suitable SSD drives is a critical task. We expected a write stream of about 100 MB/s. This results in ˜8,4 TByte data written to the cache every day. A normal consumer SSD is specified for about 75-150 TBW[19]. We selected a middle class enterprise SSD Samsung SM863[20] with a specification of 6160 TBW. This gives an expected life-time of about 3 years. The connection to the storage subsystem remained unchanged (2 x 8 GBit/s Fibre Channel). The interaction of the components in the new NFS server is shown in Figure 2.

As a fast first level cache the RAM of the NFS server is used as a Linux kernel page cache. Unallocated RAM is used to cache data from the storage devices. About 236 GByte is used as file cache. Half of the file cache is marked as inactive. This means that this cache data

[17] LACP: Link Aggregation Control Protocol; IEEE specification for bonding of multiple network links.
[18] RAID 10: striped set from a series of mirrored drives
[19] TBW: TeraByte Written. Specifies the total amount of data that can be written to the disk before flash cells starts to die. For example the popular Sasmung 840 EVO 250 GB is specified for 75 TBW.
[20] Alternate SSDs considered were: Intel SSD DC S3610 800 GB with 5300 TBW; Sandisk CloudSpeed Ascend 960 GB with 1752 TBW; Toshiba HK3R2 960 GB with 1752 TBW

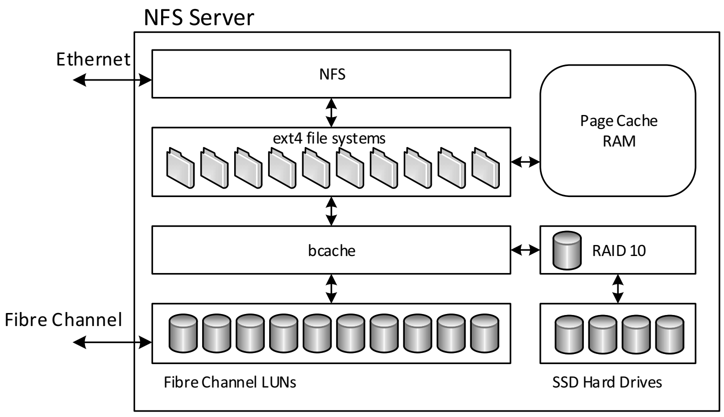

Abb. 2: NFS Server RAM and SSD caching structure

has not been accessed for a longer time period. The values are only changing slightly over time.

For bcache to work, a minimum of two block devices are necessary. A Cache-Device to store the cached data and a Backing-Device that should be cached. In our setup the RAID 10 device provided by the RAID controller is used as bcache Cache-Device. The Backing-Devices are the LUNs provided over Fibre Channel. To confine problems caused by corrupt filesystems, we decided to provide 10 x 2 TByte LUNs instead of a single 20 TByte LUN. Every LUN was then configured as Backing-Device and the resulting block device was formated with an ext4 filesystem. The resulting 10 filesystems are exported over NFS and used in the ESX cluster as NFS datastores. As shown in 2 a single Cache-Device is used to cache the 10 LUNs provided.

The default configuration of bcache can be modified by the proc filesystem. For our setup we activated the *write back* option and disabled *sequential cutoff*. With enabled *write back*, write operations are stored on the cache device (SSD) and later copied to the backing device. This allows bcache to collect write operations and write them to the backing device in a more sequential order. The *sequential cutoff* detects sequential write operations and redirects the operations directly to the Backing-Device. In our setup we disabled *sequential cutoff* to ensure that each write operation is made to the Cache-Device first. This way the number of write operations to our storage system is minimized.

5 Results

The new caching setup (RAM + bcache) reduces the IO latencies within the virtual machines significantly. The same fio test as stated before shows that the overall IO latency was reduced:

```
    lat (usec) : 250=99.89%, 500=0.08%, 750=0.01%, 1000=0.01%
    lat (msec) : 2=0.01%, 4=0.01%, 10=0.01%, 20=0.01%, 50=0.01%
    lat (msec) : 100=0.01%
```

The amount of dirty data in the bcache fluctuates over time and is given in figure 3 for each Backing-Device. Some caching devices like bcache0 and bcache3 have to handle virtual machines with a more write intensive workload. The amount of dirty data for each caching device is limited to approximately 19 GByte[21]. It is possible to use more space as write cache, but first results show that the default configuration of bcache is sufficient. The figure shows the first eight days after start of the new caching setup. The migration of the VM Container files to the new cached datastores can be clearly seen as a spike in the figure. After this initial copy procedure, the VMs were powered back on and resumed normal operation. The read operations cache hit ratio for every cached device is given in

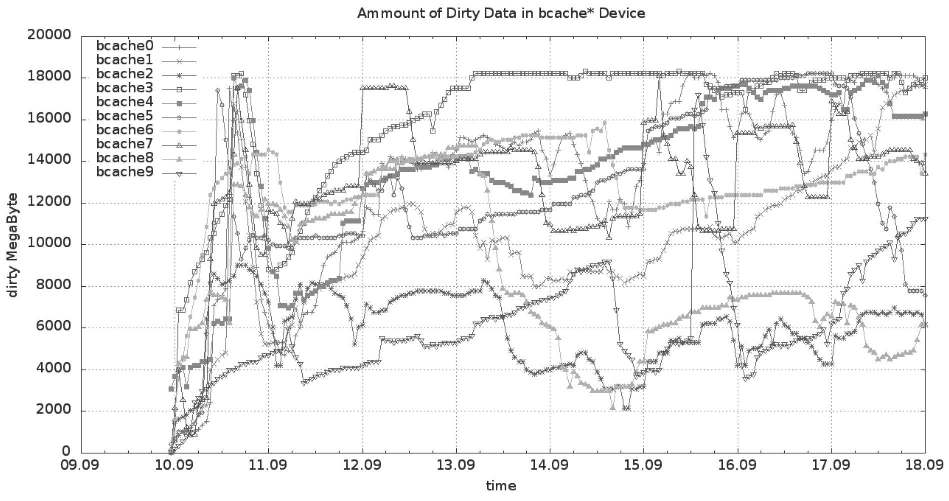

Abb. 3: Amount of dirty data in bcache devices over a period of eight days

figure 4(a-j). The ratio fluctuates heavily over time for some bcache devices. For bcache0 and bcache3 the average cache hit ratio is low, but the amount of dirty data in the cache is high. This suggests that VMs with a more write intensive workload are running on that data store.

Before the system was taken into production, several failure scenarios where tested to ensure that data written to the write back cache is not corrupted in case of an outage. For this a VM was writing a known data to a datastore and the bcache NFS server was rebooted, reset and disconnected from the power. In none of the tested scenarios data was lost or corrupted. After the server rebooted successfully the NFS client connection was restored and all write operations continued.

[21] Default bcache config: 10% of the caching device is used as write cache. This space is divided by the number of caching devices. In this setup: 1919, GByte * 0,1 / 10 = 19,193 GByte

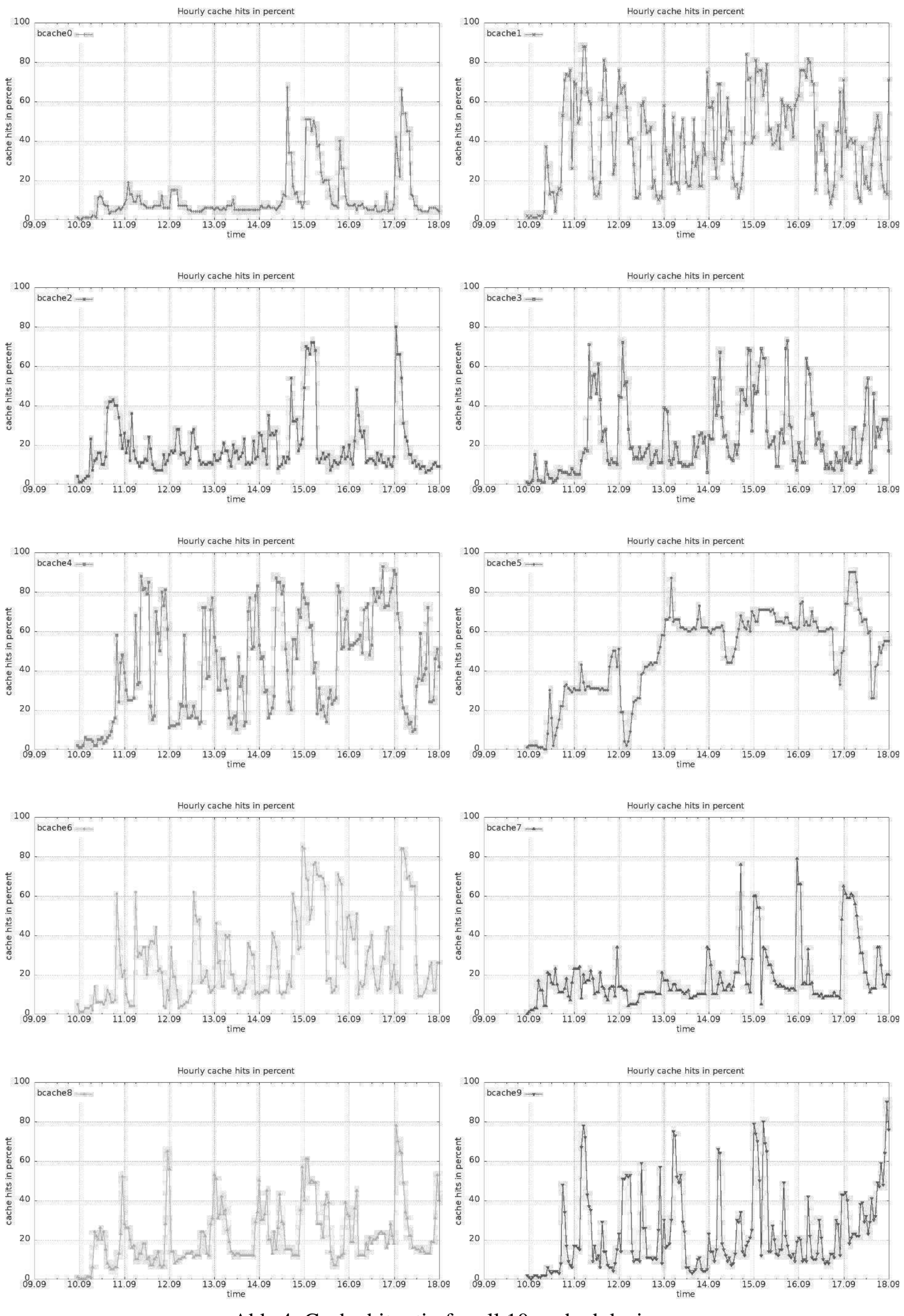

Abb. 4: Cache hit ratio for all 10 cached devices

6 Conclusion

The overall setup took less than two days to complete and migrate all data. The setup was designed in a way to allow a single SSD to fail and to get replaced without any downtime. In total we invested an amount of 10.000 Euro to bridge the gap until the new storage system is in place. The relatively moderate investment significantly improved the performance and even allowed to add new virtual machines (a process which was stopped due to the issues).
The setup has now been running for more than 4 month without any failure or data loss. The SSDs are permanently monitored to check for the amount of data written (TBW limit) and to look for possible failures. A total amount of 97.4 TB has been written to the bcache (SSD disks). This is significantly less than predicted (cf. section 4). The amount of data written by bcache corresponds to the values reported by the SMART from the disks. Due to the RAID 10 configuration every disk reports about 50 TB written.

Initial results show that the cache significantly reduces the load on the spinning disks as well as the overall I/O latency. Hot data is cached on the SSD drives and can be accessed much faster than the data on the spinning disks.

References

[AR13] Ali, Ahmad; Rose, Charles: , bcache and dm-cache - Linux Block Caching Choices in Stable Upstream Kernel. http://en.community.dell.com/cfs-file/__key/telligent-evolution-components-attachments/13-4491-00-00-20-43-81-99/LinuxBlockCaching.pdf, 2013. Accessed: 2016-1-10.

[bc] bcache. https://www.kernel.org/doc/Documentation/bcache.txt. Accessed: 2016-1-10.

[By12] Byan, S.; Lentini, J.; Madan, A.; Pabon, L.; Condict, M.; Kimmel, J.; Kleiman, S.; Small, C.; Storer, M.: Mercury: Host-side flash caching for the data center. In: Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on. pp. 1–12, April 2012.

[Ch15] Chen, X.; Chen, W.; Lu, Z.; Long, P.; Yang, S.; Wang, Z.: A Duplication-Aware SSD-Based Cache Architecture for Primary Storage in Virtualization Environment. Systems Journal, IEEE, PP(99):1–12, 2015.

[De15] Dell Storage Engineering: , Dynamic Disk Pools Technical Report. http://i.dell.com/sites/doccontent/shared-content/data-sheets/en/Documents/Dynamic_Disk_Pooling_Technical_Report.pdf, 2015. Accessed: 2016-1-10.

[dm] dm-cache. https://www.kernel.org/doc/Documentation/device-mapper/cache.txt. Accessed: 2016-1-10.

[Fa] Facebook Flashcache. https://github.com/facebook/flashcache/. Accessed: 2016-1-10.

[Ho14] Hollowell, Christopher; Hogue, Richard; Smith, Jason; Strecker-Kellogg, William; Wong, Antonio; Zaytsev, Alexandr: The Effect of Flashcache and Bcache on I/O Performance. Journal of Physics: Conference Series, 513(6):062023, 2014.

[ST] STEC EnhanceIO SSD Caching Software. https://github.com/stec-inc/EnhanceIO. Accessed: 2016-1-10.

[Wo09] Wood, Roger: Future hard disk drive systems. Journal of Magnetism and Magnetic Materials, 321(6):555 – 561, 2009. Current Perspectives: Perpendicular Recording.