Using Solid State Disk Array as a Cache for LHC ATLAS Data Analysis

W Yang, A B Hanushevsky, R P Mount, on behalf of the ATLAS Collaboration

SLAC National Accelerator Laboratory
2575 Sand Hill Road, Menlo Park, CA 94024, USA

E-mail: yangw@slac.stanford.edu, abh@slac.stanford.edu, rmount@slac.stanford.edu

Abstract. User data analysis in high energy physics presents a challenge to spinning-disk based storage systems. The analysis is data intense, yet reads are small, sparse and cover a large volume of data files. It is also unpredictable due to users' response to storage performance. We describe here a system with an array of Solid State Disk as a non-conventional, standalone file level cache in front of the spinning disk storage to help improve the performance of LHC ATLAS user analysis at SLAC. The system uses a long period of data access records to make caching decisions. It can also use information from other sources such as a work-flow management system. We evaluate the performance of the system both in terms of caching and its impact on user analysis jobs. The system currently uses Xrootd technology, but the technique can be applied to any storage system.

1. Motivation

User data analysis in the LHC ATLAS Experiment is usually data intensive. Many analysis jobs read large volumes of data. The IOs are small, sparse but intense. At large data centers, most of the storage devices are rotational hard disk drives (HDD). Spinning disks usually have ~5ms of rotational latency [1]. Archiving good performance with spinning disks under this kind of IO pattern requires spreading data evenly across a large number of spinning disks.

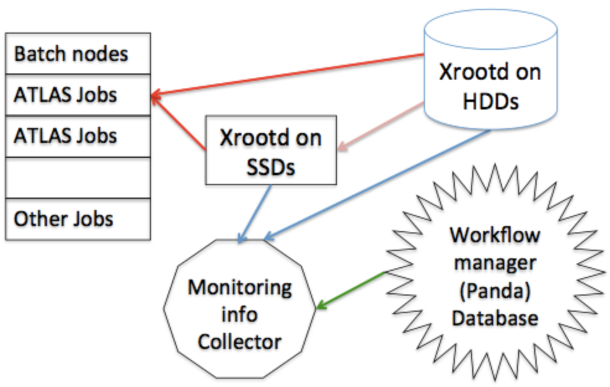
Solid State Disks (SSD) have much shorter access latency, of the order of ~25μs [2]. However, SSDs are expensive, have limited write cycles, and long data erase+write time (~2ms). To achieve reasonable write performance, file systems need to separate erase and write operations by passing the TRIM/UNMAP command to the SSDs [3]. No current RAID implementations allow TRIM to pass through. So in this study the SSDs are used as an unprotected cache.

The data re-access cycles of the ATLAS user analysis activities are usually long, or don't exist. Users also tend to "punish" well performing storage systems simply because they prefer to run more analysis if they can. This presents a challenge to the caching algorithm/mechanism of an SSD cache.

The much shorter access latency also implies that each read takes shorter time. This has a potential to improve the CPU usage efficiency of user analysis.

2. Architecture of the SSD cache

We implemented an SSD cache as shown in Figure 1 at the ATLAS Western Tier 2 Center (hosted by the SLAC National Accelerator Laboratory). We chose a centralized SSD storage cache node because it is relatively easier to setup and manage than distributing the SSDs to HDD storage nodes.



🟥 Jobs input data, Jobs will try SSDs first, then HDDs if needed
🟧 SSD stage-in data from HDD based on caching algorithm
🟦 SSD and HDD boxes send Xrootd monitoring info to collector
🟩 Fetching a list of input data files of the upcoming Panda jobs

Figure 1. The Architecture of the SSD cache, along with the HDD storage.

We used Xrootd [4] software to manage the HDD storage and SSD cache, including space cleaning in the SSD cache. It is also responsible for delivering data to clients. We add the SSD cache in between users and the HDD storage. User jobs read data files from the SSD cache using an Xrootd client such as ROOT [5] or “xrdcp”, the xrootd copy utility. If the SSD cache doesn’t have the files, the Xrootd server on the SSD cache will redirect the client to the HDD storage. Xrootd servers keep track of the data IOs of each file, and send this information to a Monitoring Info Collector, which saves the information in its own storage. The caching algorithms will use this information to predict future IO access patterns. The caching algorithm may also fetch a list of input data files from the PanDA [6] workflow management system. A caching decision is made using the above two sources of information. The SSD cache works at file level. Sub-file level caching is still under development.

The SSD cache node has dual Intel Xeon E5620 2.4Ghz CPUs, 24GB memory, dual 10Gbps NICs, and an LSI SAS 9200-8e HBA 6Gbps (non-RAID) with 12 OCZ Talos 2C series MLC SSDs (total ~11TB). It runs 64bit RedHat Enterprise Linux 6. We use a non-RAID HBA and RHEL6 in order to pass TRIM to the SSDs.

3. Xrootd IO monitoring information

During operation, the Xrootd servers on the SSD and HDD storage can send out, in real time, detailed IO information via UDP packets [4]. On the SSD cache node, we run a Monitoring Info Collector to assemble these UDP packets [5]. So by the time a file access is closed, we have a complete record of file access history. After filtering out records we don’t need, we save them to a ROOT file for caching algorithms to make caching decisions. A typical file access record looks like the following (Figure 2):

```
file_lfn=/atlas/.../NTUP_SUSY.01271227._000009.root.1
file_size=1041320520
start_time=1381343564
read_bytes_at_close=24768903
```

Figure 2. A file access record assembled by the Xrootd Monitoring Info Collector.

4. Caching algorithms and cache performance

Many storage systems do not provide detailed IO information down to the level of individual IOs. We developed three caching algorithms that make caching decisions based of the levels of detailed IO information available to use:

- The frequency a file is accessed.
- Number of bytes read from a file.
- Number of bytes read from a file and a list of input files from the upcoming user analysis jobs

The SSDs we used are Multi-Level Cell (MLC) devices. They have limited rewrite cycles. To prevent quick wear out, the caching algorithms are allowed to stage in new data files up to 2% of the total SSD space per hour. Furthermore, the caching algorithms will not stage in files whose records show obvious sequential reads and/or writes.

4.1. Caching algorithm using file access frequency information

From the Xrootd monitoring data, the algorithm builds a table (Table 1) with 10 periods. Each cell in the table is the number times the file is accessed in that period. A period is 12 hours. Period 1 is the most recent period and we right shift the table every 12 hours.

Table 1. A table listing the file access frequency during the last 10 periods.

	Period 1	Period 2	…	Period 10
File 1	4	2		0
…	1	3		1
File N	2	0		7

4.2. Caching algorithm using number of bytes read information

This algorithm looks at how many bytes have been read from each file. Every hour the algorithm builds a table (Table 2) from the last 5 days' monitoring data, sorted by the right-most column. It filters out files that haven't been read frequently enough (e.g. less than 5 times during the last 5 days), or are already in the SSDs.

Table 2. A table listing both file accessing frequency and the average percentage of bytes being read.

	Number of access in the last 5 days	Average number of bytes read from the file divided by file size
File A	7	0.72
…	17	0.20
File X	5	0.01

4.3. Caching algorithm using number of bytes read and analysis job input data information

The algorithm is similar to the previous one except the caching algorithm also checks against the list of input files of the upcoming user analysis jobs from the PanDA workflow management system: Every hour the algorithm builds a table (Table 3) from the last 5 days' monitoring data, and fills the data in the blue columns. Every 20 minutes the algorithm uses the upcoming PanDA jobs information to predict the future need from jobs. It inserts the two green columns to the table accordingly. It then sorts the right-most column and makes caching decision.

Table 3. A table lists file access frequency, the average percentage of bytes being read, and the predicted reading pattern.

	Number of reads in 5 days	**Average bytes read/file size**	**# of reads by upcoming jobs**	**Total read/file size by upcoming jobs**
File A	7	0.73	0	0
File B	17	0.20	2	0.4

File C	0	0.1 (assume)	5	0.5
…				
File X	5	0.01	3	0.03

4.4. Cache performance

With algorithm 1, we could not consistently use the SSD cache effectively. Most of the time it staged in more data to the SSD cache node than the amount of data the cache delivered.

Figure 4 shows the cache performance of algorithm 2 and algorithm 3. Both caching algorithms use the cache effectively. Algorithm 3 also reduces the amount of data staged into the SSD cache node. This reduces the load to the back end HDD storage, and more importantly, reduces the wear out of the SSDs.

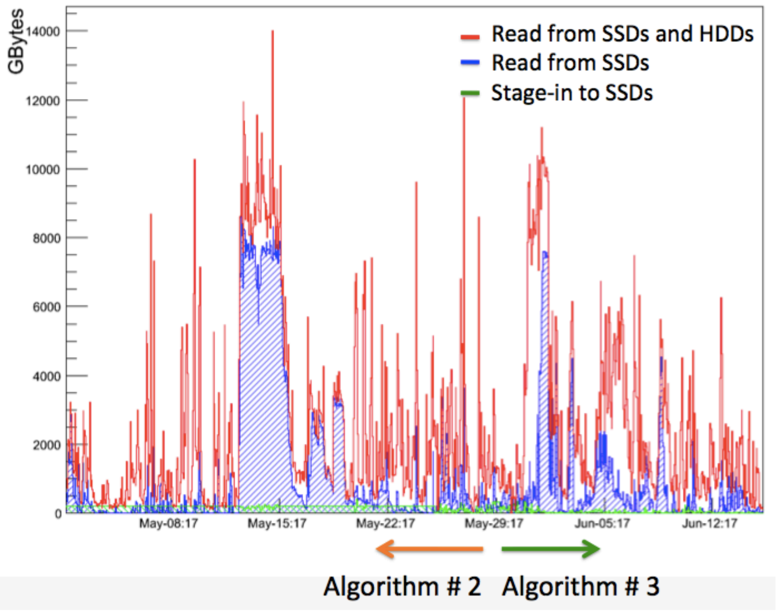


Figure 3. Cache performance using caching algorithm 2 and 3 during a 45 day period. Each bin is one hour. The two 10Gbps NICs on the SSD node deliver up to 2.5GByte/s (9000GByte/hour).

5. The SSD cache impact on user analysis jobs

As mentioned previously, using SSDs can potentially reduce the amount of time user analysis jobs spend waiting for data. To study this effect, we chose two sets of identical hardware groups that ATLAS jobs would run upon. By manipulating the firewall rules on Group A, we let jobs running on Group A skip the SSDs and go directly to HDDs, while jobs on Group B tried the SSD first.

We compared the CPU time and wall clock time information extracted from the batch system. Since one batch job runs multiple user analysis jobs, we cannot compare CPU and wall clock time of an individual user analysis job. But we can compare the total CPU time and wall clock time contribution of these two groups in a given period. Table 4 compares these contributions for a period of 33 days. During this period, jobs running on Group B (which utilized the SSD cache) were able to run more jobs (as indicated by more CPU consumption), and use the CPU more efficiently.

September 13-15 is a period when the SSD cache hit rate is very high, as shown in the Figure 5. During this period, we expect input data for jobs running on Group B will mostly read from SSDs, while jobs running on Group A will exclusively read from HDDs. Table 5 show that during this period, Group B delivered more CPU time with less wall clock time.

Table 4. CPU time and wall clock time contribution from both machine groups during a 33 day period.

2013-09-04 to 10-07	CPU hours	Wall clock hours	CPU/Wall
Group A	25776.9	34143.3	0.75
Group B	28508.6	35258.7	0.81

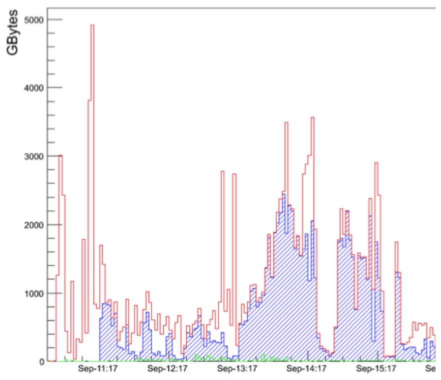


Figure 4 Cache performance

Table 5 CPU and wall clock time contribution from both machine groups during 9-13 to 9-15

09-13 to 09-15	CPU hours	Wall clock hours	CPU/Wall
Group A	49.5	240.2	0.21
Group B	59.8	166.3	0.36

6. Conclusions

The SSD cache at the ATLAS Tier 2 at SLAC demonstrated that it indeed worked as a cache. It can also speed up user analysis jobs thanks to SSD's low access latency and high sustained IO operations per second.

Due to the fluctuating nature of the Grid based user analysis jobs running at the Tier 2, it is difficult to achieve high cache hit rate at all times. We have accumulated more than one year of Xrootd IO monitoring data and we hope this will help us identify access patterns and improve cache hit rates.

The setup we have is at an R&D stage. The operational complexity makes it vulnerable to mistakes. Some of the software is not in production quality and requires constant manual checking. We will make improvements in this area in order to improve the quality of the Xrootd IO monitoring data.

References

[1] "Red Hat Documentation: Hard Drive Performance Characteristics". redhat.com

[2] A L Shimpi "The SSD Anthology: Understanding SSDs and New Drives from OCZ". AnandTech.com. 2009-03-18.

[3] Intel Corporation (2010-09-14). "Intel® High Performance Solid State Drive - Advantages of TRIM". Intel.com.

[4] A Dorigo et al., 2005 Xrootd A highly scalable architecture for data access, *WSEAS Transactions on Computers* (2005)

[5] R Brun and F Rademakers, *Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996*, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86

[6] T Maeno 2008 *J. Phys.: Conf. Ser.* **119** 062036

[7] A Hanushevsky 2013 XRootD System Monitoring Reference, retrieved from http://XRootD.slac.stanford.edu/doc/prod/xrd_monitoring.pdf

[8] M Tadel 2004 Gled - an implementation of a hierarchic server-client model *Applied Parallel and Distributed Computing (Advances in Computation: Theory and Practice* vol 16) ed Pan Y and Yang L T (Nova Science Publishers) ISBN 1-59454-174-4