LEAF: A data cache and access system across remote sites

To cite this article: Y D Cheng et al 2018 J. Phys.: Conf. Ser. 1085 032008

View the article online for updates and enhancements.
LEAF: A data cache and access system across remote sites

Y D Cheng, Q Xu, C Wang
Institute of High Energy Physics, CAS, Beijing, China
E-mail: chyd@ihep.ac.cn

Abstract. Nowadays, distributing computing technologies has been widely used in high energy physics field. Computing job is usually scheduled to the sites where the input data was pre-staged in. This model will lead to some problems including low CPU utilization, inflexibility, and difficulty in highly dynamic cloud environment. The paper proposed a data cache and access system, which presents one same file system view at local and the remote sites, supporting directly data access on demand. Then the computing job can run everywhere no need to know where data is located. The system was implemented based on streaming and caching mechanism. The test results showed the performance was better than traditional file system on high-latency WAN.

1. Introduction
Nowadays, distributing computing technologies has been widely used in high energy physics field. The Worldwide LHC Computing Grid (WLCG) [1] is one of the largest distributed computing system in the world. More than 500 thousand CPU cores and 980 PB storage from 167 sites are involved in WLCG collaboration until April 2017. To process the BES-III experiment's data the distributed computing infrastructure [2] based on DIRAC middleware [3] has been setup and became operational since 2013. Current BES-III distributed infrastructure includes 18 resource centers from 7 countries, providing access to about 3000 CPU cores and 0.5 PB of disk space. But most of sites are lack of dedicated administrator. Furthermore, public clouds such as Amazon AWS, Microsoft Azure, and Alibaba Aliyun are gradually popular today in High Energy Physics (HEP) area.

Computing in high energy physics is typical data-intensive application, and it needs a large amount of input data or output data in one task. And these data is usually organized in file system. Each file contains thousands of event data. Currently local computing model is typically cluster composed of storage system and computing nodes. The job runs on the computing nodes, fetching input data from storage system or putting output data into storage system. Grid is one kind of distributed computing system, which can be considered as global cluster of local cluster. Cloud in HEP is something like cluster or grid on virtual machines. The typical architecture of HEP computing system is depicted in figure 1.

When a job is submitted to a distributed computing system such as grid or cloud, it will be finally scheduled to one site containing a local cluster. Then the input data should be moved into the same cluster too. For example, WLCG deploys File Transfer Service (FTS) [3] which is responsible for distributing the majority of LHC data across the WLCG infrastructure. The data is pre-staged into one site, then the job can be scheduled to the site where the input data is. In such a distributed computing model, there are some problems as follows:

1) If one site does not have enough storage space, it cannot keep enough input data, and then there is no enough jobs running on the site. So it will potentially lead to low CPU utilization.
2) It is usually up to site administrators to decide which data should be transferred. If the data distribution is imbalance, it is difficult to schedule jobs fairly across different sites.

3) Virtual computing resources can be created in public cloud on demand, but it is not possible to transfer a large amount of data to remote cloud immediately, and some user’s jobs cannot run without input data.

4) The whole file is transferred to remote site, but user’s job only is usually interested in a few of events in the file, maybe only 1/1000, event more less [4]. So too much data is transferred based on the granularity of file.

The feasible solution to solve the above problems is to mount local file system directly on remote sites, which presents the same file system view at local and remote sites, then applications can run everywhere. However, traditional file systems such as Lustre, gluster, EOS cannot work well on high latency wide area network. We designed and implemented a data cache and access system – LEAF based on streaming and caching mechanism. This paper will describe the architecture, implementation and test results of LEAF.

2. The design and implementation

2.1. Design conception

LEAF aims at providing a same file system view at local and the remote sites, and good access speed over WAN as well. Client requests can be served as soon as one small fraction of file is available before the whole file is fully downloaded. It also should be portable, compatible, scalable, secure and reliable.

LEAF has two kinds of instances including one main site where all data is stored and serval remote sites where user can access data from main site on demand. To implement the system, we adopted the following design conceptions.

1) File system view consistency between main site and remote site. It is achieved by periodical metadata synchronization from main site and real-time update during file access.

2) File system syntax conversion between POSIX and file transfer. Data is transferred using multi-stream, chunk, non-block and other technologies to improve the performance across WAN.

3) Easy to deploy in internal network. Some small sites or computing nodes on public cloud do not have public IP, or network sockets are restricted by firewall. However, it is usually allowed to use HTTP protocol to access external resources. LEAF services running on remote sites can be easily deployed in internal network by exploiting HTTP protocol.

4) Easy to be compatible with existing HEP software. The existing HEP applications such as BES Offline Software System (BOSS) [5] should use the system to access remote data without modification. LEAF is implemented as an Xrootd [6] client pluglin, and can be transparently called by ROOT framework, or use XrootdFS (a Posix Fileysitem for XrootD) to mount as a local file system.

2.2. Architecture
The architecture of LEAF is depicted in Figure 2. There are three components including TransferD, CacheD and Xrootd Client Plugin. TransferD is deployed in main site. CacheD is deployed in remote sites. Xrootd Client Plugin is a client module.

![Diagram of LEAF architecture]

Figure 2. The architecture of LEAF composed of three components, TransferD, CacheD and Xrootd client plugin

2.3. Implementation

2.3.1. Data transfer service
Data transfer service is used to move data between main site and remote site, which has two components including TransferD and client library. TransferD is a daemon running at main site, which is configured with public IP address. Client library is deployed at remote sites, which is called by CacheD. Client does not need public IP. TransferD supports different data sources including local file system, storage system like Hadoop, and database such as Hbase.

Data transfer service is developed based on Tornado web framework [7]. Tornado is a Python web framework and asynchronous networking library. By using non-blocking network I/O, Tornado can scale to tens of thousands of open connections, making it ideal for long polling, WebSockets, and other applications that require a long-lived connection to each user.

Data transfer service can download data from main site or upload data into main site. If data transfer service receives a request, it will download or upload data using multi-streams in parallel to improve the performance. Data transfer service client routines are similar to file system POSIX calls including stat, readdir, read, write, link and so on. The parameters of these routines are also like POSIX, such as file path, mode, offset, buffer, etc.

2.3.2. Disk cache service
Disk cache service is used to cache data in remote site, which has three components including CacheD, DB and Client tool. CacheD is a daemon running in remote site, which receives requests from client, and then checks the information in the DB and decides to send data from cache or get data from remote site. DB stores file metadata of CacheD and bitmap which indicates which blocks of one file has been cached. Currently DB supports MySQL and ramcloud system [8].

If a file is opened for read, CacheD firstly checks if it exists in the cache. If it exists, CacheD will open the cached file and return fd. If it does not exist in the cache, CacheD will create an empty file with the same size as the original file in main site. If one application read a file, CacheD will check if the requested block exists in the cache. If the block is in the cache, CacheD will return block data to the application. If the request block does not exist in the cache, CacheD will get fix-sized block from
main site and then modify bitmap related to the file metadata, then return the block data. The workflow of LEAF read operation is depicted in figure 3.

![Workflow of LEAF read operations](image)

**Figure 3.** The workflow of LEAF read operations

If a file is opened for write or create, CacheD firstly checks if it exists in the cache. If it exists, CacheD will return an error because it is allowed to modify existing file in remote site. If it does not exist in the cache, CacheD will create an entry in main site, then create an empty file in remote site. CacheD then opens the empty file and return fd. CacheD then continue to receive data from application and write data into the file. After the write operation finishes, CacheD uploads the file to the main site, and update file metadata in main site as well. The workflow of LEAF write operation depicted in figure 4.

![Workflow of LEAF write operations](image)

**Figure 4.** The workflow of LEAF write operations

2.3.3. Xrootd Client Plugin
LEAF client is implemented as an Xrootd Client Plugin to ensure HEP software access data transparently. Usually HEP software support ROOT framework [9] which can access file by using xrootd API, for example XrdPosix. LEAF can be accessed by most HEP software. If some software do not support ROOT framework, LEAF can be mounted as local file system through XrootdFS. XrootdFS is a Posix filesystem for an Xrootd storage cluster. It is based on FUSE (Filesystem in Userspace) and runs in user space. LEAF presents a local file system, and any software can access it. The implementation of Xrootd Client is depicted in figure 5.
Figure 5. The implantation of Xrootd Client Plugin in LEAF

Xrootd Client Plugin can get data either from XrdPosix or XrootdFS, then it firstly connect CacheD to check if the block is in cache. If yes, CacheD will tell the physical path of the file. If not, it will connect CacheD to get the block from main site, then CacheD returns the physical path of the file. Finally, Xrootd Client Plugin gets real data from disk using xrootd API.

3. Test bed and results
We have setup a test bed including two sites between Beijing and Chengdu to evaluate the performance of the system. The distance of the two sites is about 2000 KM. The network latency is about 35 ms, bandwidth is 1 Gpbs, and iperf benchmark tool reported about 80 MB/s. The test results showed LEAF data transfer performance can achieve nearly 80 MB/s, and the performance is getting better with the increasing of stream number as depicted in figure 6.

Figure 6. The performance of LEAF data transfer system between Beijing and Chengdu with 1Gpbs network link

As we know, TCP throughput is not only decided by bandwidth, and it is directly impacted by latency and packet loss. So traditional network file system such as NFS, EOS and Luster cannot work well in wide area network. The paper [10] showed NFS performance has degraded drastically while network latency was increased from 10 ms to 90 ms. We used tc tool to simulate network latency and found that the performance of EOS and lustre was decreased by 97.8% and 99.6% in the case of 100 ms latency. Then we tested the performance of LEAF with 1Gbps link and 100 ms latency. The test results showed the performance was decreased by 31%, much better than EOS and Lustre. The results was given in figure 7.
4. Summary
Distributed computing such as grid and cloud is widely used in high energy physics field. It is more flexible to access data directly from remote sites on demand. But traditional file systems won’t work well over high-latency network. LEAF is designed as an extension of main storage system, which aims at improving data access performance over WAN. The test results showed LEAF was not impacted by network latency greatly and it had good performance across remote sites.

5. Acknowledgements
This presentation is supported by National Natural Science Foundation of China (No. 11575223) and National key Research Program of China “Scientific Big Data Management System” (No.2016YFB1000600).

References
[1] Shiers J. The worldwide LHC computing grid (worldwide LCG)[J]. Computer physics communications, 2007, 177(1): 219-223.
[2] Belov S, Suo B, Deng Z Y, et al. Design and operation of the BES-III distributed computing system[J]. Procedia Computer Science, 2015, 66: 619-624.
[3] Ayllon A A, Salichos M, Simon M K, et al. FTS3: New Data Movement Service For WLCG[J]. J.phys. Conf.ser, 2014, 513(3):032081.
[4] Liu B. High performance computing activities in hadron spectroscopy at BESIII[J]/ J.phys. Conf.ser, 2014, 523(1):012008.
[5] Wei-Dong Li, Ya-Jun Mao, Yi-Fang Wang. Chapter 2 The BES-III Detector and Offline Software[J]. International Journal of Modern Physics A, 2009, 24(supp01):9-21.
[6] Dorigo A, Elmer P, Furano F, et al. XROOTD - A highly scalable architecture for data access[J]. Wseas Transactions on Computers, 2005, 4(4):348-353.
[7] Dory M, Parrish A, Berg B. Introduction to Tornado: Modern Web Applications with Python[M]. " O'Reilly Media, Inc.", 2012.
[8] Ousterhout J, Gopalan A, Gupta A, et al. The RAMCloud storage system[J]. ACM Transactions on Computer Systems (TOCS), 2015, 33(3): 7.
[9] Antcheva I, Ballintijn M, Bellenot B, et al. ROOT — A C++ framework for petabyte data storage, statistical analysis and visualization[J]. Computer Physics Communications, 2009, 182(6):1384-1385.
[10] Radkov P, Yin L, Goyal P, et al. A Performance Comparison of NFS and iSCSI for IP- Networked Storage[C]// Usenix Conference on File & Storage Technologies Usenix Association. 2004:101--114.