Measurement of BeStMan Scalability

Haifeng Pi, Igor Sfiligoi, Frank Wuerthwein, Abhishek Rana
University of California San Diego

Tanya Levshina
Fermi National Accelerator Laboratory

Alexander Sim, Junmin Gu
Lawrence Berkeley National Laboratory

1. Introduction

Berkeley Storage Manager (BeStMan)[1] is a java-based full implementation of Storage Resource Manager (SRM)[2], which is widely used for storage resource management that enables file access and replication in distributed environments. The open specification of SRM has been remarkably successful because it provides a consistent homogeneous interface to the Grid and allows various sites to interoperate despite possible difference in the infrastructure and technologies of the storage resources.

According to the general specification of SRM, BeStMan includes some important features, for example space management, data movement, load balancing etc. These functionalities are useful for Grid users to reserve space, move files and conduct other storage related operations. From the system perspective, BeStMan is able to interact with remote storage on their behalf and significantly improve the interoperability between different Grid sites.

In recent development of scientific computing, some new distributed file systems are integrated into the Grid storage system, for example Hadoop distributed file system (HDFS)[3]. In a typical architecture of the Grid site with BeStMan + HDFS (or other file systems) implementation, the BeStMan server is running under the Gateway mode and equivalent to a load balancing frontend for data transfer. BeStMan can work on top of a number of disk-based POSIX-compliant file systems, such as NFS, GPFS, PVFS2, GFS, Ibrix, HFS+, Hadoop, XrootdFS and Lustre. It also works with any existing file transfer services, such as gridftp, http, https, bbftp and ftp.

In large-scale scientific collaboration, the storage resource of a Grid site is always open to and shared with thousands of users. Storage system have to simultaneously support high throughput data transfer, intensive random local data access for user analysis jobs, low latency real time data analysis from interactive applications. It requires a highly scalable and available frontend system that provides general data access to the users and applications.

Since many of the Grid sites have adopted the scale-out strategy for both computing power and storage capacity, the overall number of CPUs and storage space can increase by a large factor during a few years with almost the same implementation of middleware and system architecture. While most of grid services are still run inside a single server, those network-facing services may

be one of bottleneck for coping with increasing number of clients, either from local or remote sites. Large scale grids, like the Open Science Grid (OSG)[4], LHC-CMS Grid[5], LHC-ATLAS Grid[6] etc., are composed of tens of thousands of CPUs and Petabytes of storage distributed across different countries and involving a level of one hundred Grid sites and thousands of authorized users and groups.

The scheduling of Grid resources is essentially to mediate the competition between users and groups based on available resources and priority set by either the collaboration or each site. The randomness of Grid resource access pattern and lacking of an effective collaboration level scheduling mechanism mean the Grid network-facing services must be scalable and reliable enough to survive the extreme high accessing rate for an extended period of time. Some organized activities, such as Monte Carlo event generation and simulation, may use thousands of CPUs and stage-out the results to one or more storage systems. It has become more and more common a single user can take high percentage of available CPU slots for a relatively short time and remote copy all the output files to one site. These use cases indicate that the storage frontend needs to be capable of handling thousand of clients nowadays to ensure the proper functioning of the site and provide reliable computing source and services to the collaboration.

Since 2009 many efforts have been taken to measure, understand and improve the scalability of BeStMan. As shown in the following figure, in the test of using large number of clients to access BeStMan conducted in 2009 [7], we observed the linear relationship between the effective processing rate of the requests at the server side and number of clients running simultaneously.

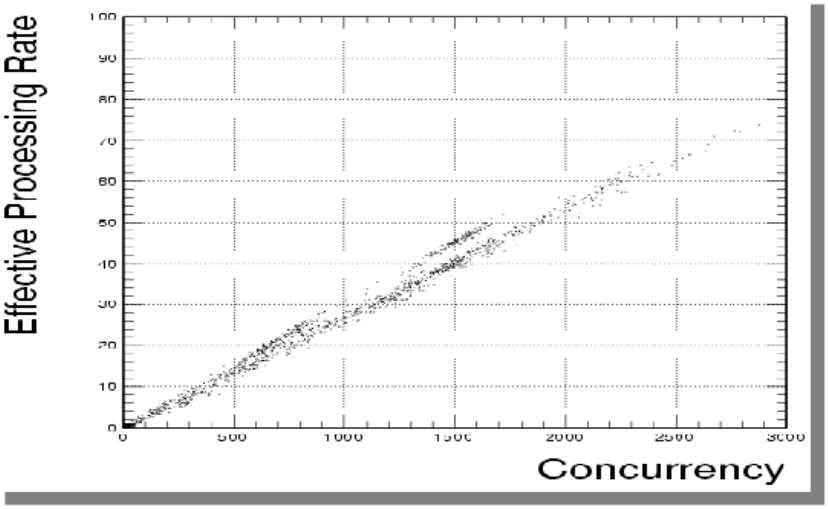


Fig.1 Correlation between Effective Processing Rate and Client Concurrency measured in 2009

This paper summarizes some recent testing activities for

- The BeStMan based on Globus[8] container in the latest OSG release. Currently this flavor of BeStMan is widely deployed for the production at many OSG Grid sites. The adoption of HDFS at Compact Muon Solenoid (CMS) in 2009 facilitated the use of BeStMan as a viable solution for the frontend of the storage system.

- The BeStMan2 based on Java Tomcat in the test release. It is expected to eventually

replace the Globus container because of its lacking of maintenance in recent years and observed limitation in its scalability and reliability.

Most of the functionalities and features of BeStMan and BeStMan2 are expected to be the same except the server configuration. In the rest of the paper, the "BeStMan" is used for either the general technology of this tool, or specifically the Globus-based implementation.

2. Scalability Testing Tools and Method

The measurement of scalability of BeStMan requires running a large number of clients to access the frontend service simultaneously. In order to mimic the real access pattern, those clients are better to run in geographically distributed Grid sites with various latency, bandwidth, processing power in the client host, local I/O capacity etc.

There are a number of tools that can be used to conduct the test of BeStMan scalability and reliability. In this work we mainly choose condor glide-in based tools[9], simply because it is able to build a virtual layer (a compute pool) on top of Grid CPU resources and provides an easy and flexible way to submit and manage testing jobs. Despite difference in the actual job slots at different Grid sites, glide-in is able to setup a collaboration wide or customized environment and match the actual job slots with user job according to specific rule or requirement.

Condor Glide-in is one of the pilot job technologies that runs at the worknode (WN) and is late-bind with the user jobs. Once a matching is made between the glide-in and user job, the job is downloaded and executed via glide-in. This mechanism allows some special monitoring and management tasks to be done without involving local queuing system at the sites, which is a powerful means to control the testing jobs, since in normal Grid job management, once the job is sent to a remote site, the management of the job is largely handled by the remote site queueing system, which puts some limitation on the type of the scalability test to be conducted. The architecture of Condor glide-in is shown in Fig.2.

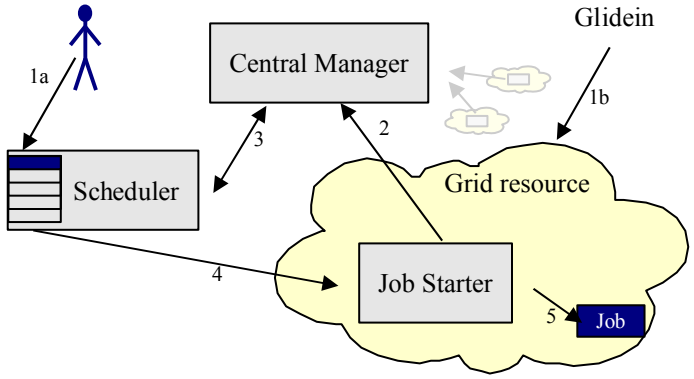


Fig.2 Overview of Condor glide-in Technology

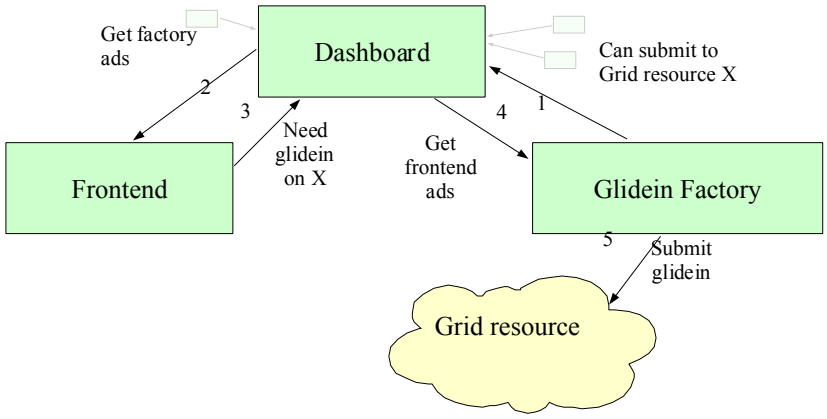
The job control is done through condor starter, which works as a local job manager of a particular slot at the WN. It is able to provide detailed status and information about the WN and job.

The server side glide-in daemon is mainly to bookkeep both the user request and available CPU slots, and behave like a resource broker. Our previous work shows that the scalability of glide-in system is big enough to simultaneously run close to 30,000 jobs across the Grid to utilize almost

all available standard job slots.

2.1 GlideinWMS for BeStMan Scalability Test

GlideinWMS (Glide-in based Workload Management System)[10] is a Grid workload management system to automate condor glide-in configuration, submission and management. GlideinWMS is used widely for several large scientific Grid for handling users jobs for physics analysis and Monte Carlo production.



The glideinWMS is composed of two parts: the glide-in factory that handles submitting glide-in to Grid sources and manages alive glide-ins, and the frontend that specifies which Grid resources are targeted and how many glide-ins are sent. Condor ClassAds are used for communication between the glide-in factory and frontend. The GlideinWMS Architecture is shown in Fig. 3.

In our test, we use GlideinWMS to send jobs to some typical Grid sites and use the normal job slots at the site. The test application will be shipped by glide-in and run at the WN. In this way, the randomness of user job access pattern is repeated in the test environment, since the availability of job slots at the remote sites, type of WN and all the local hardware and WAN networking features are the same as normal Grid jobs.

Fig. 3 Overview of GlideinWMS Architecture

The drawback of this approach is the proceeding of the tests is completely controlled by the Grid instead of the tester:

- The tester has to compete with other Grid users to acquire CPU slots
- The execution of testing job is automated by GlideinWMS on which sites, type of WNs etc.
- The concurrency of clients achieved by the test may vary from time to time. The validity of results depends on whether the network-facing service is “scalable” enough to respond to the real time change in the client access.
- Usually as testing jobs acquire more and more resources, the overall number of clients running across the Grid might cause the server malfunctioning. Some sites

may fail to maintain the connection to the server and lead to other networking problems. In this case, the site administrator has to kill all the testing jobs.

Since the test uses normal job slots, it is difficult to run a large number of clients simultaneously. We implement multi-threading in the test job. Each job can run multiple threads, which is configureable, as if there are multiple jobs running at the same WN. We found this approach is able to accomplish large concurrency of clients with running limited number of jobs. There is chance that multiple jobs are running at the same WN, while each job runs multiple threads, which may cause problem to the WN. We limited the maximal number threads less than 5. No negative impact on the computing system is found during the test.

The CPU resources from CMS Virtual Organization (VO) is targeted for the test. The GlideinWMS frontend is configured to present proper Grid user authentication and authorization during the test. So does the user environment at the WN. Since the test of I/O of BeStMan is a generic operation, no VO-dependent application or environment is actually used for the test.

Each job is configured to finish a predefined amount of I/O operations against the BeStMan server. The information of the operation (starting time and end time of each operation) is recorded in a log file and shipped back after the job finishes. The I/O of jobs themselves and their activities are found negligible on the system load and network throughput.

2.2 GlideinTester for BeStMan Scalability Test

The Glidein Tester[11] is a fully automated framework for scalability and reliability testing of centralized, network-facing services using Condor glide-ins. The main purpose of this testing system is to simplify the testing configuration and operation. For example, to achieve a certain level of concurrency, we have to send a large number of jobs to glideinWMS, how many jobs can run simultaneously is out of our control. A test system can actually actively acquire and monitor the number of free CPU slots in the Condor pool and manage to send testing jobs to those slots when some conditions are met, for example we want to run X jobs simultaneously.

Following is a short description of the new testing system architecture, which is built on GlideinWMS. The GlideinTester architecture is shown in Fig.4.

- It uses a virtual-private Condor glide-in pool managed by glide-in factory. It sends jobs to the Grid in the same way as normal glide-in submission.

- A daemon is running at the frontend to pass the user testing request to the glide-in factory. The user request for CPU resources is based on the concurrency at which the test targets. Once the number of glide-ins that are able to accept user jobs is equal to the testing jobs, those jobs will be submitted.

- In the real implementation, the user can specify multiple concurrencies in the configuration for a chain of tests. The configuration parameters include how long each test takes, how much the concurrency for every test etc.

- The system has a template for user to include executable and add customized log in addition to the standard testing log provided by the system. In every run, each job has its own directory at the submission side to hold the output and standard Condor log files.

- All Condor debugging and monitoring features are inherited from the standard release, thus available for the tests.

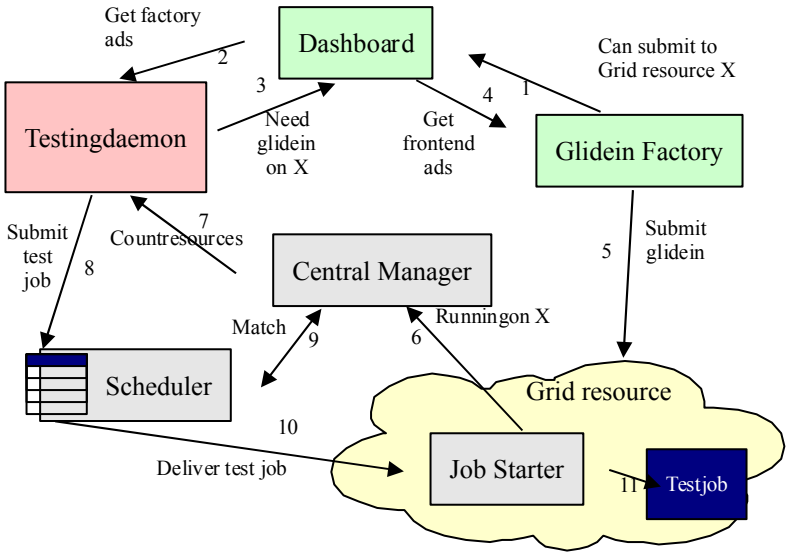


Fig. 4 Overview of GlideinTester Architecture

Since GlideinTester is a dedicated tool for scalability and reliability tests, a high level of concurrency should be achieved. Using the normal job slots available in the Grid will be limited by how many slots the test can acquire from various sites and whether those job slots are available during a particular period of time. In the CMS VO, the total number of slots is O(30k) [12]. Due to the priority setting and policy of each site, using up to 5% of total slots for the test is typically the limit. In order to overcome this difficulty, we use some collaboration sites and Condor sleep slots. Following is an overview of how the dedicated Condor slots are incorporated into the test infrastructure:

- The glide-ins from the GlideinTester to certain sites will specify what Condor slots to use at those sites. Those sites will pass the glide-in to the Condor sleep slots, which usually are not available for normal user jobs. From job itself, there is no difference between normal job slot and sleep slot, in terms of running the applications or user jobs within glide-in.

- In the Condor technology, Condor sleep slots provide extra available slots on each WN. At five CMS sites, the local batch system implements more batch slots per WN than CPU cores. For example at UCSD a hyperthreaded Dual Quad-Core WN have 50 batch slots, while only a 8-16 of them are actually used for running user jobs. For analysis and other related jobs, the system is commonly configured to run one batch slot per core to maximize the usage of resources and provide necessary and sufficient processing power for the applications to be run by the jobs. The rest batch slots on each WN, as “sleep” slots and restricted from normal access, can be used for some work of special purposes, such as the scalability and reliability test.

- The sleeps slots are manged by the same queueing system and same site policy. Technically from user environment point of view, there is no difference between normal batch slots and “sleep” slots, so the whole test infrastructure and technology can be easily integrated with and supported by those Grid sites that are willing to provide resources.

- In general, the tests using “sleep” slots will not cause any trouble to the cluster and other user jobs, because the load of the scalability and reliability test is mainly on the server side. On the client side, the application's waiting time dominates with very small usage of CPU and memory, which allows running much more testing jobs at the WN than normal usage. Normal user jobs might consume up to 100% of CPU and a fairly amount of memory. For example, a typical CMS analysis job will use 1-2 GB of memory, while for the scalability and reliability test, this is not the case.

- With the help of using “sleep” slots, we can potentially run tens of thousands of testing jobs, which is at the similar scale of all available job slots of the CMS VO, without affecting the normal operation of the sites. It is also much easier to acquire those slots since only a few sites are involved and there is no other competition for those resources.

2.3 Testing Methods

Each testing job is configured to run a loop of query on the BeStMan server for listing all the files in a given directory in the storage system, the storage is either a virtual local file system or a real mounted Hadoop distributed file system (HDFS).

Throughout the test, the load and resource consumption (CPU and memory) are primarily on the BeStMan server and its access point to the storage file system (e.g. FUSE mount at the BeStMan server).

3. BeStMan Configuration

3.1 System Implementation

BeStMan is part of the VDT (Virtual Data Toolkit) release. The detailed implementation procedure that is maintained by OSG can be found at. The implementation of BeStMan is usually based on following storage architecture:

- BeStMan provides frontend services of SRM v2.2 interface on a dedicated BeStMan server for a disk-based storage system.

- GridFTP servers conduct the high performance, secure and reliable data transfer. The GridFTP service is deployed over a bunch of servers, which can be either dedicated servers, or WN that processes user jobs, or a combination of WN and storage node which provides storage space to the local SE.

- BeStMan performs load-balancing among those GridFTP servers. The load-balancing is done via real time monitoring. Gratia GridFTP transfer probe has been integrated with BeStMan operation that selects GridFTP of the lowest load in the system (we called load-probe method). A simple GridFTP selector (we called random-selector) that select an alive GridFTP randomly or using Round-Robin algorithm from the list is also developed and available in the release for reliability and simplicity of usage.

  The use of load-probe or random-selector method depends on the implementation of GridFT at the sites. With relatively small number of GridFTP (typically less than 10 servers) equipped with large I/O capacity, the load-probe is needed to finely balance the load across those servers.

  For a site running large number of GridFTP, e.g. running this service on WN at the order of 100s of independent GridFTP services, the load-probe method can be too costly. In the case that there is significant redundancy of total GridFTP services running in the system, the number of available connections provided by GridTP services will be much larger than the minimal "needed" connections to saturate the WAN network bandwidth, the Random-Selector is a preferred choice.

- For a production storage system, the GUMS[13] is normally used for authorization because of flexibility and functionality provided to the site administration. Usually a standalone GUMS server needs to be deployed. For simple testing purpose with limited amount of users and reducing the overhead of accessing GUMS to further speed up BeStMan service, grid-mapfile based solution can be used.

The overview of the typical SE architecture using BeStMan is shown in Fig. 5[14].

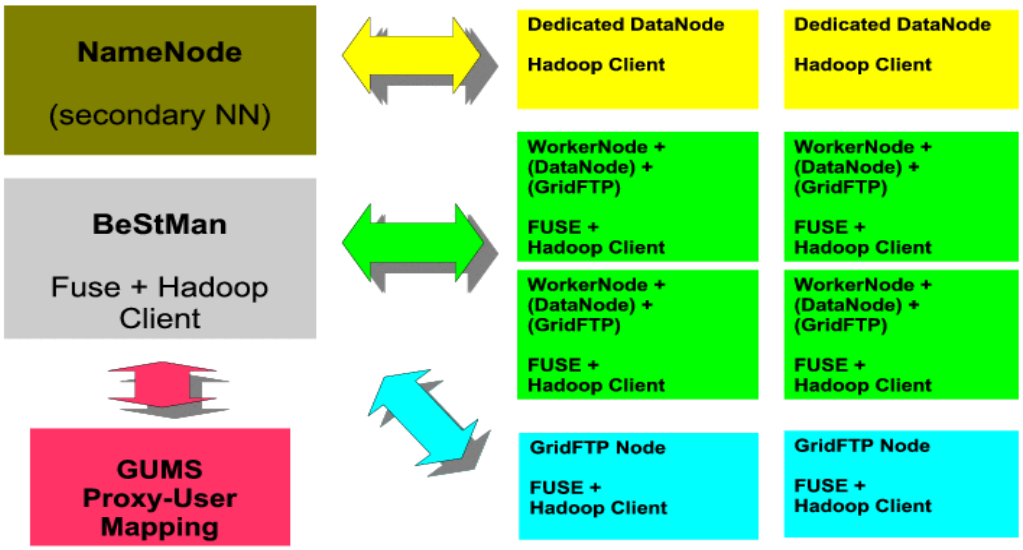


Fig. 5 Overview of BeStMan-SE Architecture

In the implementation of BeStMan, to make it access the storage via a POSIX-compliant

interface, FUSE is usually needed. This can be a separate deployment. Since many Grid site started to use HDFS as the backbone of the SE, an integrated HDFS-based SE implementation is supported and maintained by OSG. Through this approach, the core SE configuration includes following components:

- HDFS namenode (NN) installation. The HDFS NN provides the namespace of the file system of the SE.

- HDFS datanode (DN) installation. The HDFS DN provides the distributed storage space to the SE. In the typical deployment strategy, the Compute Node can also be configured as DN.

- FUSE installation on HDFS and BestMan. The FUSE provides the POSIX-compliant interface of HDFS for local data access of the SE.

- GridFTP installation. It provides grid data transfer from or to the SE. It should be emphasized that the interface between BeStMan and storage, and between BeStMan and GridFTP can be more generic, while the interface between GridFTP and storage will be less likely general, which is complicated by how the SE file system mange the disk space and whether it provides a POSIX-compliant interface for data access.

  The interface between GridFTP and storage system (e.g. a distributed file system) can be a serious bottleneck for high throughput data transfer. For the architecture using a distributed file system across the cluster, the WN I/O will involve internal data replication of the file system, external WAN-read or write of data, and local data access of user jobs. In the integration of HDFS-GridFTP, the GridFTP needs to use native HDFS client to access the file system and buffer the incoming data in memory because the HDFS doesn't support asynchronous writing and for some cases the GridFTP is not able to write data to disk storage fast enough.

  From current system integration and data access strategy, most of system dependent implementation takes place at GridFTP level.

- BeStMan installation. It provides the SRM v2.2 interface of the SE and performs the load-balancing of the GridFTP servers.

3.2 Software Release of HDFS and BeStMan

The VDT packaging and distribution of HDFS and BeStMan is based on YUM[15]. All components are packaged as RPMs. Two YUM repositories are available

- Stable repository for wider deployments and production usage.
- Testing repository for limited deployments and pre-release evaluation.

This distribution is independent of the Pacman packaging of the VDT: it is separately versioned, and separately packaged. It is expected the future releases to eventually be common with the rest of the VDT, as the "rest" of the VDT begins to be packaged as RPMs. For now, the VDT distribution of HDFS and its related BeStMan are distinct from the rest of the VDT.

The stable YUM repository is enabled by default through the osg-hadoop RPM, and contains the "golden release" supported by OSG for LHC operations. The "golden release" is tested and validated by a couple OSG sites.

The version we used for the test is: 2.2.1.3.13 for BeStMan and 2.0.1 for BeStMan2.

3.3 Configuration Parameters

Following is a list of configuration parameters, of which the configuration has significant impact on the performance of BeStMan.

Generic system configuration:

- Max Heap Size of Java application, this parameter specifies total virtual memory can be used BeStMan processes. Assuming each incoming client thread uses 2.5 KB, to process 2000 of clients requests simultaneously, 5GB is needed for server to work properly.

- Limit of max number of opening files (File Descriptor) and processes for a user, since high concurrency at the server requires opening a large number of files to be processed simultaneously.

- System wide File Descriptors (FD) limit, which can be set in the kernel variable.

- TCP/IP tuning, Number of incoming connection backlog, which needs to be set up to 5000 or 10000, or through kernel auto-tuning.

- BeStMan log level. Switch off log level will improve the scalability.

- Use grid-mapfile instead of GUMS to eliminate the possible limit in GUMS scalability when testing high concurrency of BeStMan query service.

For BeStMan based on Globus container:

- Max number of threads in Globus container, which directly determines how many incoming client requests can be processed by the server simultaneously.

For BeStMan based on java tomcat and Jetty container:

- Max number of threads in Jetty container, which is similar to the setting of Globus container.

Our tests showed that without tuning above parameters the maximal concurrency level of BeStMan out of the standard release is near 500-600 for both BeStMan and BeStMan2.

3.4 Test Hardware

Following is the BeStMan server configuration at UCSD:

- Quad dual-core AMD Opteron Processor 275, 2.2 GHz and 1MB cache for each processor
- 8 GB of RAM

- 180GB of local disk space

The test HDFS cluster configuration is:

- 12 datanodes and 1 namenode
- Each node has Quad dual-core AMD Opteron Processor 275, 1.0 GHz and 1MB cache for each processor
- 8 GB of RAM
- 2-3 TB of disk space for the HDFS

The networking features:

- 1 Gb/s Ethernet uplink
- Two 10 Gb/s external link for the test and production cluster, SONIC and ESnet
- At UCSD, the BeStMan uses FUSE mount to access the HDFS storage.

A similar BeStMan sever is deployed at Fermi National Accelerator Laboratory (FNAL). The FNAL BeStMan uses local host disk as the storage file system.

4. Results and Discussion

In the rest of the paper, we differentiate the test results between BeStMan and BeStMan2. The technology specification of BeStMan and BeStMan2 is described in Introduction.

4.1 Test of BeStMan with small directory

We used GlideinWMS to test the performance of BeStMan. Similar results are observed in the test in 2010, which is shown in Fig. 6. We observe a wide spreading in the correlation between the effective processing rate at BeStMan and high concurrency region which shows the scale of the number of clients running simultaneously.

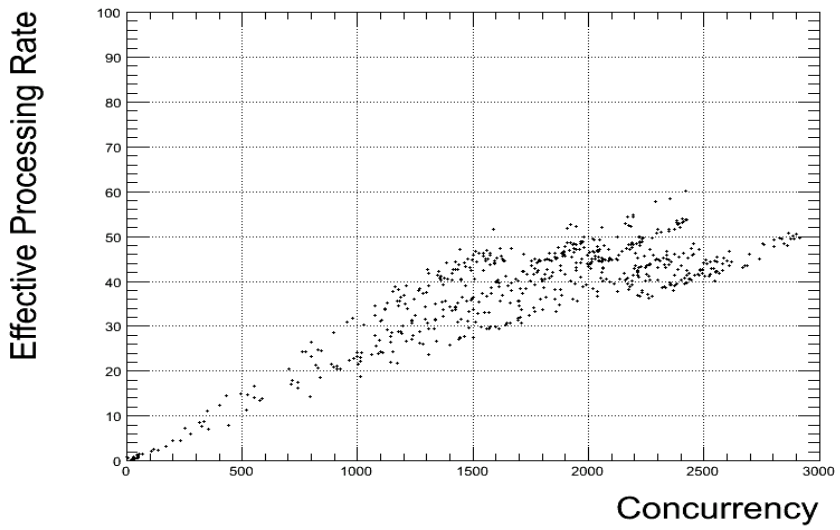


Fig. 6 Correlation between Client Concurrency and Effective Processing Rate for BeStMan with small directories

The test is against a small directory with ~10 files. The listing command takes very short time in the native file system comparing to BeStMan activities to process the request including

authentication, authorization, assignment of threads and remote I/O.
On average etc. 1000 (2000) client concurrency makes ~25 (50) Hz processing rate, which is in the scale of last year's result as shown in Fig.1.

Overall the BeStMan shows following features in its scalability

- There is strong linear correlation between the average effective processing rate and client concurrency. There average ratio is 1 Hz per 40 clients. More clients push the server processing rate higher.

- The above correlation indicates that the BeStMan servers spends fixed amount of time on processing an incoming request which is irrelevant to the number of requests. As number of clients increases, the more requests are processed in parallel.

- If the concurrency goes very high (for example more than 2500 client concurrency), the BeStMan stops functioning and hangs. Even the there is no client activities later, BeStMan is not able to recover, so we have to restart the BeStMan.

4.2 Test of BeStMan2 with small directory

We use glidetester to test BeStMan2 deployed at FNAL and UCSD. The correlation between effective processing rate and client concurrency for BeStMan2 is shown Fig.7.

For BeStMan2, a much different correlation is observed:

- The average effective processing rate has little dependency on the client concurrency.

  For BeStMan2 running at FNAL, the local disk file system is used as target for BeStMan2 storage services. The scalability of FNAL is higher than the HDFS file system used at UCSD. We expect it is at least 30% slower for HDFS via FUSE to run those POSIX commands.

  The difference in BeStMan2 performance may also come from the limitation of total number of opening files and how FUSE efficiently handles a large number of commands simultaneously.

- The above correlation indicates that the BeStMan2 servers spends fixed amount of system resource for processing all incoming requests which is irrelevant to the number of requests. As the number of requests increases, the same resource is shared with more requests, which leaves no increase in the effective processing rate.

- If the concurrency goes very high, the BeStMan2 stops accepting new requests. But it is able to recover later when all the existing requests are processed.

- The overall scalability of BeStMan2 is shown in the high end of effective processing rate vs concurrency distribution for the BeStMan. This shows the BeStMan2 delivers better performance for most of time. It is only when the client concurrency beyond 2500, the BeStMan may be able to provide higher performance than BeStMan2.

  But our tests also show that when the concurrency of more than 2500, the system is overly stressed, and the effective processing rate is not a good indicator of quality of

service any more.

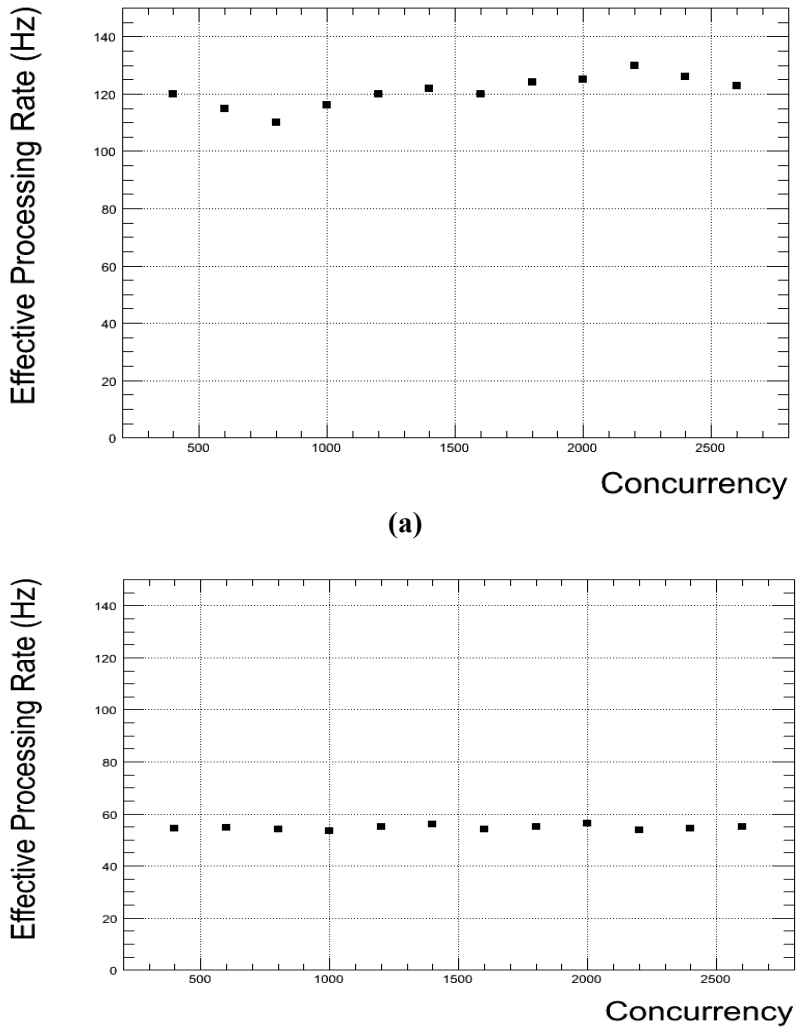


Fig. 7 Correlation between Effective Processing Rate and Client Concurrency for BeStMan2 with small directories at (a) FNAL and (b) UCSD

4.3 Test of BeStMan2 with various size of directories

With big directories, the scalability we measured involves not only the BeStMan2, but also the FUSE and distributed file system, since the processing time for running on the file system and FUSE is not negligible comparing to the processing time of BeStMan itself.

Fig. 8 and 9 shows the Effective Processing Rate and Time vs Client Concurrency for directories of 20, 100, 200 and 1000 files respectively. In the computing system administration and design of computing software at CMS VO, we usually limit the total number of files in a directory below 1000. A directory hold a large number of files may cause various problem for the health of the whole storage system.

The results show:

- The Effective Processing Rate is stable vs client concurrency for a given number of files in the directory.
- The larger the directory, the lower the scalability and the longer the processing time for BeStMan2 to finish the request.
- There is a close to linear relationship between processing time and size of the directory.

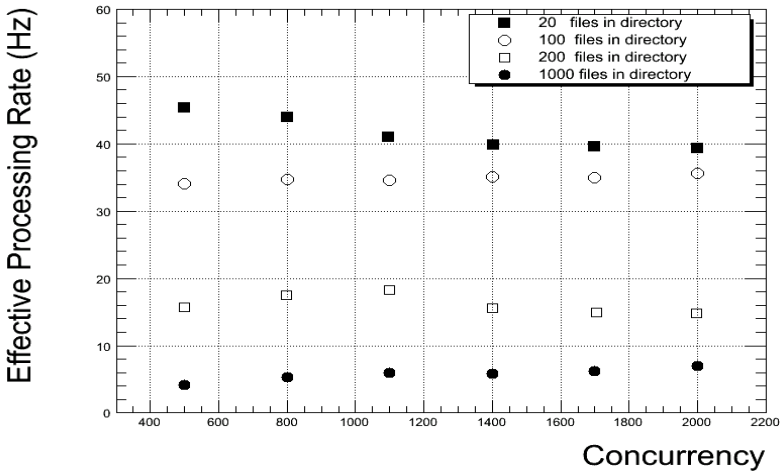


Fig. 8 Correlation between Effective Processing Rate and Client Concurrency for large directories

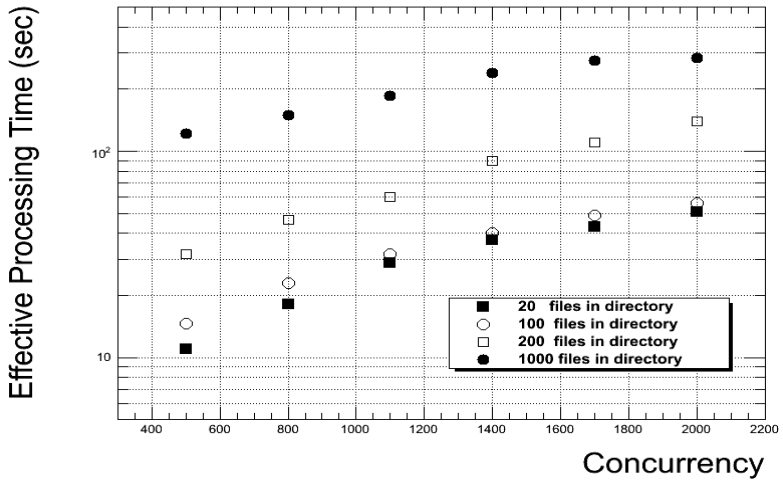


Fig.9 Correlation between Effective Processing Time and Client Concurrency for large directories

5. Conclusion

We measure the scalability of recent OSG released BeStMan and BeStMan2. There is significant difference in the scalability features between two types of BeStMan technology. The BeStMan2 shows better performance and reliability from running the high throughput and high available service point of view.

References

1. Berkeley Storage Manager (BeStMan), http://sdm.lbl.gov/bestman/
2. Storage Resource Manager (SRM), http://sdm.lbl.gov/srm-wg/doc/SRM.v2.2.html
3. Hadoop Distributed File System (HDFS), http://hadoop.apache.org
4. Open Science Grid (OSG), http://www.opensciencegrid.org
5. CMS Virtual Organization, http://lcg.web.cern.ch/lcg/
6. ATLAS Virtual Organization, http://lcg.web.cern.ch/lcg/
7. "Hadoop File System as part of a CMS Storage Element", CMS Document, 2009
8. Globus, http://www.globus.org/
9. Condor glidein, http://www.cs.wisc.edu/condor/
10. glideinWMS, http://www.uscms.org/SoftwareComputing/Grid/WMS/glideinWMS
11. "glideTester at OSG Scalability Area Tools, http://sourceforge.net/projects/osgscal
12. CMS Analysis Operations, CMS Conference Report, 2009-088
13. GUMS, https://www.racf.bnl.gov/Facility/GUMS/1.3/index.html
14. "Hadoop Distributed File System for the Grid", IEEE NSS/MIC, 2009
15. OSG Hadoop YUM release, http://vdt.cs.wisc.edu/components/hadoop.html