PROCEEDINGS OF SCIENCE

Performance and Stability of the Chelonia Storage System

Jon Kerr Nilsen
University of Oslo, Dept. of Physics, P. O. Box 1048, Blindern, N-0316 Oslo, Norway
University of Oslo, Center for Information Technology P. O. Box 1059, Blindern, N-0316 Oslo, Norway
E-mail: j.k.nilsen@usit.uio.no

Salman Toor*
Computing Uppsala University, Box 256, SE-751 05 Uppsala, Sweden
E-mail: salman.toor@it.uu.se

Zsombor Nagy
Institute of National Information and Infrastructure Development NIIF/HUNGARNET, Victor Hugo 18-22, H-1132 Budapest, Hungary
E-mail: zsombor@niif.hu

Bjarte Mohn
Dept. of Physics and Astronomy, Div. of Nuclear and Particle Physics, Uppsala University, Box 535, SE-751 21 Uppsala, Sweden
E-mail: bjarte.mohn@gmail.com

Alex Read
University of Oslo, Dept. of Physics, P. O. Box 1048, Blindern, N-0316 Oslo, Norway
E-mail: a.l.read@fys.uio.no

In this paper we present Chelonia, a storage system designed to fill the requirements gap between those of large, sophisticated scientific collaborations which have adopted the grid paradigm for their distributed storage needs, and of corporate business communities which are gravitating towards the cloud paradigm. The similarities to and differences between Chelonia and several well-known grid- and cloud-based storage solutions are commented. The design of Chelonia has been chosen to optimize high reliability and scalability of an integrated system of heterogeneous, geographically dispersed storage sites and the ability to easily expand the system dynamically. The architecture and implementation in term of web-services running inside the Advanced Resource Connector Hosting Environment Dameon (ARC HED) have been described discussed earlier. In this paper we present results of tests in both local-area and wide-area networks that demonstrate the fault-tolerance, stability and scalability of Chelonia.

The International Symposium on Grids and Clouds (ISGC) 2012,
Febraury 26 - March 2, 2012
Academia Sinica, Taipei, Taiwan

*Speaker.

PoS(ISGC 2012)009

1. Introduction

As computationally demanding research areas expand, the need for storage space for results and intermediate data increases. Currently running experiments in fields like high energy physics, atmospheric science and molecular dynamics already generate petabytes of data every year. The increasing number of international and even global scientific collaborations also contributes to the growing need to share (and protect) data efficiently and effortlessly.

Over the years, different concepts have evolved to deal with the challenge of handling increasing volumes of data and the fact that the data tend to be generated and accessed over vast geographic regions. The largest storage solutions nowadays can be roughly divided between the data grids developed and used by scientific community, and cloud storage arising from the needs of the corporate business community. While both concepts have the same goal of distributing large amounts of data across distributed storage facilities, their focuses are slightly different.

While different research groups have different requirements for a storage system, a set of key characteristics can be identified. The storage system needs to be *reliable* to ensure data integrity. It needs to be *scalable* and capable to dynamically *expand* to future needs, and given that many research groups today use computational grids to process their data, it is highly favorable if the storage system is *grid-enabled*.

In this paper we have presented the stability and performance of the Chelonia storage system [4, 16]. With Chelonia we aim at designing a system which fulfills the requirements of global research collaborations, but which also meets the specifications of user-centric interfaces and transparency. With Chelonia it is possible to build from a simple storage systems for sharing holiday pictures to medium-scale storage systems for storing scientific data.

This paper is organized as follows: In Section 2 we give a brief overview of Chelonia storage system. Sections 3 and 4 present the performance and stability of Chelonia, respectively. A comparison with other storage solutions on the market is given in Section 5. Section 6 presents conclusion and ongoing development work.

2. Overview of Chelonia Storage System

Chelonia consists of a set of SOAP-based web services residing within the Hosting Environment Daemon (HED) [11] service container from the ARC grid middleware [1]. The Chelonia communication layer is provided by the next generation Advanced Resource Connector (ARC) Grid middleware, developed by NorduGrid [7] and the EU-supported KnowARC project [5]. Together, the services provide a self-healing, reliable, scalable, resilient and consistent data storage system. The distribution of data is random over the available geographically distributed Shepherd nodes. Data is managed in a hierarchical global namespace with files and subcollections grouped into collections[1]. A dedicated root collection serves as a reference point for accessing the namespace. The hierarchy can then be referenced using Logical Names. The global namespace is accessed in the same manner as in local filesystems. There is no explicit limit imposed on the size of the Chelonia storage but keeping the presented stability and performance, Chelonia can manage around 10-million objects by capitalizing hierarchical namespace in a distributed environment.

[1] A concept very similar to files and directories in most common file systems.

Being based on a service-oriented architecture, the Chelonia storage consists of a set of services. The Bartender provides the high-level interface to the user; the Librarian handles the entire storage namespace, using the A-Hash as a meta-database. The A-Hash can be configured as centralized or replicated using the Oracle Berkeley DB [8] (BDB); A Shepherd is a front-end for a physical storage node. Note that any of the services can be deployed in multiple instances for high availability and load balancing. [16, 14, 15] will provide comprehensive technical and operational details about the system.

As is the case for all openly accessible web services, the security model is of crucial importance for the Chelonia storage. While the security of the communication with and within the storage system is realized through HTTPS with standard X.509 authentication, the authorization related security architecture of the storage can be split into three parts;

- **Inter-service authorization** maintains the integrity of the internal communication between services. To enable trust between the services, they need to know each other's Distinguished Names (DN's). This way a rogue service would need to obtain a certificate with that exact DN from some trusted Certificate Authority (CA).
- **Transfer-level authorization** handles authorization for uploading in and downloading files. When a transfer is requested, the Shepherd will provide a one-time Transfer URL (TURL) to which the client can connect.
- **High-level authorization** considers the access policies for the files and collections in the system. The ARC middleware services, has its own security framework and policy language for describing access policies and requests.These policies are stored in the A-Hash, in the metadata of the corresponding file or collection, providing a fine-grained security in the system.

The only part a user will (and should) see from a storage system, the client tools are an important part of the Chelonia storage system. In addition to a command-line interface, two ways of accessing Chelonia are supported.

- **FUSE Module** provides a high-level access to the storage system. Filesystem in Userspace (FUSE) [6] allows users to mount the Chelonia namespace into the local namespace, enabling e.g. the use of graphical file browsers.
- **Grid Job** can access data through the ARC middleware client tools, one needs to go through Data Manager Components (DMC's). The ARC DMC allows grid jobs to access Chelonia directly. As long as A-REX, the job execution service of ARC, and ARC DMC are installed on a site, files can be downloaded and uploaded by specifying the corresponding URL's in the job description.

3. System Performance

Here we report measurements of the performance of the most frequently used actions on files and collections of files in both LAN and WAN-based deployments of the Chelonia services as well as for the basic centralized and replicated configurations of the services.

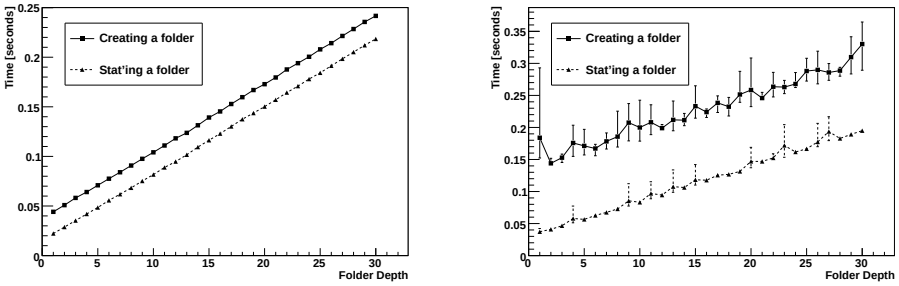
3.1 Adding and Querying the Status of Files

In a hierarchical file system files are stored in levels of collections and sub-collections. The time to add or get a file depends mainly on two factors; the number of entries in the collection where the file is inserted (the width of the collection), and the number of parent collections to the collection where the file is inserted (the depth of the collection). Based on these two factors we have run two different tests:

- **Depth test** tests the performance when creating many levels of sub-collections. The test adds a number of sub-collections to a collection, measures the time to add and stat the sub-collections, then adds a number of sub-sub-collections to one of the sub-collections and so forth. To query a collection at a given level means that all the collections at the higher levels need to be queried first. In Chelonia, each query causes a message to be sent through TCP. Hence, it is expected that response-time will increase linearly as the depth in the collection hierarchy increase. As every message is of equal size, this test ideally depends only on network latency.

- **Width test** tests the performance when adding many entries (collections) to one collection. The test is carried out by adding a given number of entries to a collection and measuring the time to add each entry and the time to stat the created entry. When adding an entry to a collection, the system needs to check first if the entry exists. In Chelonia, this means that the list of entries in the collection needs to be transferred through TCP. It is therefore expected that the time to add an entry will increase linearly as this list increases and ideally the time will depend only on the bandwidth of the network.

Both tests were run in two types of environments: In the Local Area Network (LAN) setup four computers were connected to the same switch. A centralized A-Hash service, a Librarian, a Bartender, and the client were each run on a separate computer. In the Wide Area Network (WAN) setup, the client and Bartender were located in Uppsala, Sweden, while the Librarian and a replicated A-Hash consisting of three replicas were located in Oslo, Norway.

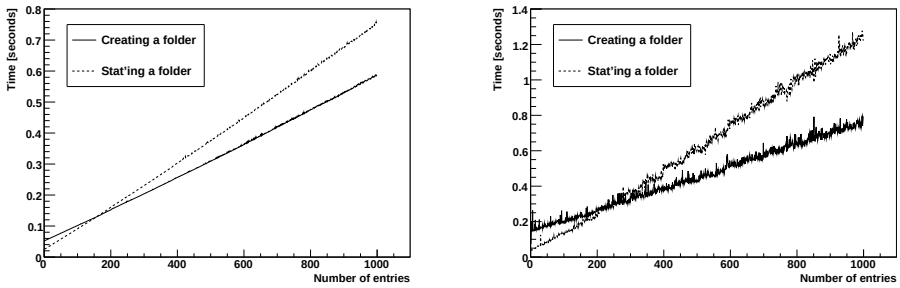


(a) Services running on LAN (b) Services running on WAN

Figure 1: Time to add an entry to a collection (continuous line) and time to get status of a collection (dashed line) given the hierarchical depth of the collection.

The test result for the depth test is shown in Figures 1(a) (LAN) and 1(b) (WAN). The continuous lines show the time to create an entry, while the dashed lines show the time to get the status of

the collection. All the plots are averages of five samples, with the error bars representing the minimum and maximum values. The LAN test shows a near-perfect linear behavior, with the error bars too small to be seen. As mentioned earlier, since the packet size for each message is constant in this test, the main bottleneck (apart from Chelonia itself) is the network latency. Since all computers in the LAN test are connected to the same switch we can assume that the latency is near constant. Hence, the LAN test shows that in a very simple network scenario Chelonia works as expected, with the network being the major bottleneck. Notice also that creating an entry consistently takes 0.021 s longer than getting the status of the collection, again corresponding to the extra message needed to create an entry.



(a) Services running on LAN (b) Services running on WAN

Figure 2: Time to add an entry to a collection (continuous line) and time to get status of a collection (dashed line) given the number of entries already in the collection.

In the WAN test, the complexity is increased, as in addition to sending messages over WAN, the A-Hash is now replicated. The time still increases linearly, albeit with more fluctuation due to the WAN environment. Creating an entry at the first level in the hierarchy now takes 0.11 s longer than getting the status of the collection, corresponding to the fact that the entry needs to be replicated three times. However, at higher levels, getting the status over WAN is actually faster than getting the status over LAN. The effect of the replicated A-Hash will be discussed in more detail in Section 3.4.

The test results for the width test are shown in Figures 2(a) (LAN) and 2(b) (WAN). The continous lines show the time to add entries to a collection and the dotted lines show the time to get the status of a collection containing the given number of entries. The operations were repeated five times, with the plots showing the averages of these five samples. As expected, the time increases linearly with an increasing number of entries. The results of the WAN test fluctuate more than those of the LAN test, which is expected; in the LAN test all services are on computers connected to the same switch, while in the WAN test, the services are distributed between two different countries. However, the WAN test shows similar linearity, albeit with a slightly higher response time. It is worth noticing that for the width test, in contrast to the depth test, we see no benefit of using a replicated A-Hash. This is due to the fact that the bandwidth is the limiting factor in this test.

An interesting feature of both tests is that while creating an entry takes more time when there are only few entries in the collection, creating new entries is actually faster than stating the collection when the collection has many entries. This is due to the fact that getting the status of a collection requires the metadata of the collection (and hence the list of entries in the collection)

to be transferred first from the A-Hash to the Librarian, and second from the Librarian to the Bartender and last from the Bartender to the client. When creating an entry, neither the Bartender nor the client needs this list of entries, so that less data is transferred between the services. However, for fewer entries, creating new entries is relatively expensive since the Bartender needs to query the Librarian twice, first to check if it is allowed to add the entry and second to actually add it. In Chelonia there is no internal limit on the collection size but it has been observed that after 5000 objects in a single collection the response time started to increase. Thus for acceptable system response, it is preferable to limit the maximum size of a single collection to less than 5000 objects.

3.2 File Replication

In this section we will demonstrate how Chelonia works with different file states to ensure that a file always has the requested number of valid (ok) replicas and how Chelonia distributes the replicas in the system in order to ensure maximum fault tolerance of the system.

The concept of automatic file replication in Chelonia works mainly with the coordination of the Librarian and the Shepherd services. The Shepherds periodically check with the Librarians to see if there are sufficient replicas of the files in Chelonia and if the checksums of the replicas are correct. If a file is found to have too few replicas, the Shepherd informs a Bartender about this situation and together they ensure that a new replica is created at a different storage node. A file having too many replicas will also be automatically corrected by Chelonia - the first Shepherd to notice this will mark its replica(s) as unneeded and later delete it (them). Replicas with invalid checksums are marked as invalid, and as soon as possible replaced with a valid replica.

The test system consists of one Bartender, one Librarian, two A-Hashes (one client and one master) and five Shepherds. All services are deployed within the same LAN. As a a starting point 10 files of 114 MB are uploaded to the system and for each file 4 replicas are requested. Thus the initial setup of the test system contains 40 replicas with an initial distribution of file replicas as shown in the table attached to the Figure 3.

Shepherd	Initial	Final
S1	9	9
S2	8	7
S3	8	8
S4	7	9
S5	8	7
Total	40	40

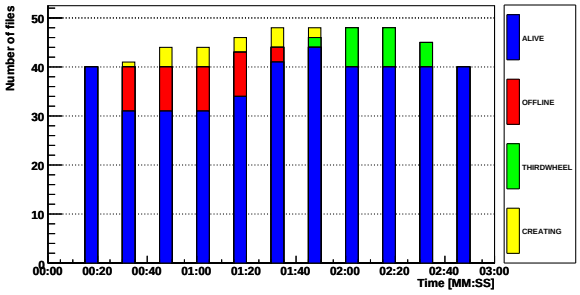


Figure 3: Table shows the initial and final load distribution of 40 files on 5 Shepherds. Figure (on the left) shows the number of replicas and their corresponding states when querying the test system every 15 seconds. Shepherd S3 is turned offline between 00:15 and 00:30 and back online between 01:15 and 01:30.

The first phase of the file replication test was to kill one of the Shepherd services, S3, of the test system. Chelonia soon recognized the loss of this service (no heartbeat received within one cycle) and started compensating for the lost replicas. File replicas in Chelonia have states ALIVE, OFFLINE, THIRDWHEEL or CREATING which are recorded in the A-Hash. Initially the test

system had 40 ALIVE replicas (10 files with 4 replicas each), but when the Librarian did not get the S3 heartbeat it changed the state of the 8 replicas stored in S3 to OFFLINE.

At this point a number of files stored in our Chelonia setup had too few ALIVE replicas. The Shepherds check periodically that files with replicas stored on its storage element have the correct number of ALIVE replicas. Thus, in the next cycle the S1, S2, S4 and S5 Shepherds started to create new file replicas which initially appeared in the system with the state CREATING.

Figure 3 gives a graphical overview of the number of replicas and the replica states in the test system every 15 seconds. At 15 s (00:15) all 40 file replicas were ALIVE as explained above, but soon thereafter S3 was turned off and the other Shepherds started creating new replicas. Queried at 30 s (00:30) the system contained 32 ALIVE, 8 OFFLINE and 1 CREATING file replica.

While the system worked on compensating for the loss of S3, the second phase of the replication test was initiated by turning the S3 shepherd online again. The reappearence of the S3 Shepherd can be seen in Figure 3 at 90s (01:30) as a significant increase in the number of ALIVE replicas. In fact there were now too many ALIVE replicas in the system and at 105 s (01:45) 2 replicas were marked as THIRDWHEEL (the Chelonia state for redundant replicas). THIRDWHEEL replicas are removed from Chelonia as soon as possible, and during the next 45 s the system removed all such replicas. A query of the system at 165 s (02:45) shows that the system once again contained 40 ALIVE replicas. The final distribution of replicas between Shepherds is given in table shown in Figure 3.

3.3 Multi-User Performance

While any distributed storage solution must be robust in terms of multi-user performance, it is particularly important for a grid-enabled storage system like Chelonia where hundreds of grid jobs and users are likely to interact with the storage system in parallel. To analyze the performance of Chelonia in such environments we have studied the response time of the system while increasing the number of simultaneous users (multi-client).

To analyze the performance of Chelonia, we have studied the response time of the system while increasing the number of simultaneous users (multi-client). Due to limited hardware resources, multiple clients for the tests were simulated by running multiple threads from three different computers. Each client thread creates 50 collections sequentially and tests were done for an increasing number of simultaneous clients. For each test the minimum, average and maximum time used by the client was recorded. Figure 4 shows the system response times for up to 100 simultaneous clients using the above-mentioned testbed deployment. The shown test was run in a LAN environment with one centralized A-Hash, one Librarian and one Bartender.

The test results show that the response time of the system increases linearly with an increasing number of simultaneous clients. From 40 clients and onwards the difference between the fastest client and slowest client starts to become sizable. When running 50 clients or more in parallel, it was occasionally observed that a client's request failed due to the heavy load of the system. When this happened, the request was retried until successfully completed, as shown by the slightly fluctuating slope of the mean curve. The same linearity was seen in the corresponding WAN test (not shown), albeit with a factor two higher average time, consistent with the results observed in the depth and width tests in Section 3.1.

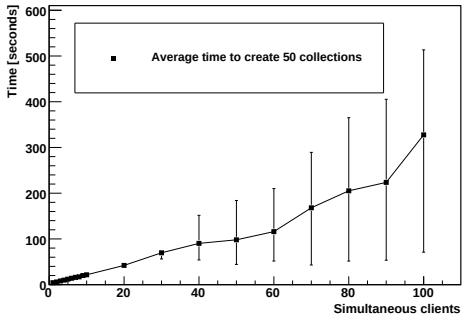


Figure 4: Average (square), minimum (lower end of bar) and maximum (upper end of bar) system response time as a function of the number of simultaneous clients of the system. Each client creates 50 collections sequentially.

As can be noted in Figure 4, for more than 30 clients, the maximum times increase approximately linearly while the minimum times are close to constant. The reason for this is a limitation on the number of concurrent threads in the Hosting Environment Daemon (HED). If the number of concurrent requests to HED reaches the threshold limit, the requests are queued so that only a given number of requests are processed at the same time. In the test each client used only one connection for creating all 50 collections. Hence, the fastest request was one that had not been queued so that when the number of requests was above the threshold, the minimum timing did not depend on the total number of clients. On the other hand, the slowest request was queued behind several other requests so that the maximum time increased with the number of simultaneous clients.

3.4 Centralized and Replicated A-Hash

The A-Hash service is a database that stores objects which contain key-value pairs. In Chelonia, it is used to store the global namespace, all file metadata and information about itself and storage elements. Being such a central part of the storage system, the A-Hash needs to be consistent and fault-tolerant. The A-Hash is replicated using the Oracle Berkeley DB [8] (BDB), an open source database library with a replication API. The replication is based on a single master, multiple clients framework where all clients can read from the database and only the master can write to the database. While a single master ensures that the database is consistent at all times, it raises the problem of having the master as a single point of failure. If the master is unavailable, the database cannot be updated, files and entries cannot be added to Chelonia and file replication will stop working. The possibility of the master failing cannot be completely avoided, so to ensure high availability means must be taken to find a new master if the first master becomes unavailable. BDB uses a variant of the Paxos algorithm [13] to elect a master amongst peer clients: Every database update is assigned an increasing number. In the event of a master going offline, the A-Hash clients sends a request for election, and a new master is elected amongst the clients with the highest numbered database update.

To test the fault tolerance and performance overhead of the replicated A-Hash in a controlled environment, four tests have been set up with services and a client on different computers in the same LAN:

1. **Centralized**: One client contacting a centralized A-Hash was set up as a benchmark, as this is the simplest possible scenario.

2. **Replicated, stable**: One client contacting three A-Hash instances (one A-Hash master, two A-Hash clients) randomly. All the A-Hashes were running during the entire test.

3. **Replicated, unstable clients**: Same setup as in point 2, but with a random A-Hash client restarted every 60 seconds.

4. **Replicated, unstable master**: Same setup as in point 2, but with the master A-Hash restarted every 60 seconds.

While Setups 1 and 2 test the differences in having a centralized A-Hash and a replicated A-Hash, Setups 3 and 4 test how the system responds to an unstable environment. In all four Setups the system has services available for reading at all times. However, in Setup 3, one may need to reestablish connection with an A-Hash client and in Setup 4 the system is not available for writing during the election of a new master. During the test the client computer constantly and repeatedly contacted the A-Hash for either writing or reading for 10 minutes. During write tests the client computer reads the newly written entry to ensure it is correctly written.

	Reading			**Writing**		
	Min (s)	*Avg (s)*	*Max (s)*	*Min (s)*	*Avg (s)*	*Max (s)*
Centralized	0.0033	0.0037	0.0134	0.0038	0.0042	0.0144
Replicated, stable	0.0034	0.0037	0.0132	0.0168	0.0339	1.0576
Replicated, unstable clients	0.0034	0.0037	0.2895	0.0164	0.0342	1.1311
Replicated, unstable master	0.0034	0.0037	1.9711	0.0162	0.0448	60.9028

Table 1: Timings for reading from and writing to a centralized A-Hash compared with a stable replicated A-Hash, a replicated A-Hash where clients are restarted and a replicated A-Hash where the master is re-elected.

Table 1 shows timings of reading from and writing to the A-Hash for the four setups. As can be seen, for reading, all four setups have approximately the same performance. Somewhat surprisingly, the replicated setups actually perform better than the centralized setup, even though only one client computer was used for reading. While reading from more A-Hash instances is an advantage for load balancing in a multi-client scenario, one client computer can only read from one A-Hash instance at a time. Thus, the only difference between the centralized and replicated setups in terms of reading is how entries are looked up internally in each A-Hash instance, where the centralized A-Hash uses a simple Python dictionary, while the replicated A-Hash uses the Berkeley DB which has more advanced handling of cache and memory.

Looking at the write performance in Table 1, there is a more notable difference between the centralized and the replicated setups. On average, writing to the replicated A-Hash takes almost 10 times as long as writing to the centralized A-Hash. The reason for this is that the master A-Hash will not acknowledge that the entry is written until all the A-Hash instances have confirmed that they have written the entry. While this is rather time-consuming, it is more important that the A-Hash is consistent than fast. It is however worth noticing when implementing services using the A-Hash, that reading from a replicated A-Hash is much faster than writing to it.

4. System Stability

While optimal system performance may be good for the day-to-day user experience, the long-term stability of the storage system is an absolute requirement.

To test the system stability, a Chelonia deployment was run continuously over a week's time. During the test a client regularly interacted with the system, uploading and deleting files and listing collections. The deployment consisted of one Bartender, one Librarian, three A-Hashes and two storage nodes, each consisting of a Shepherd and a Hopi service (One for the ARC native HTTP(S) server). All the services and the client ran on separate servers in a LAN environment.

Figure 5 shows the overall memory utilization for seven of the services for the entire run time (top), the A-Hashes and the Librarian in the first 24 hours (bottom left) and the Librarian, the Bartender and one of the Shepherds in the last 37 hours (bottom right). The memory usage was measured by reading the memory usage of each service process in 5 seconds (singular) intervals using the Linux `ps` command.

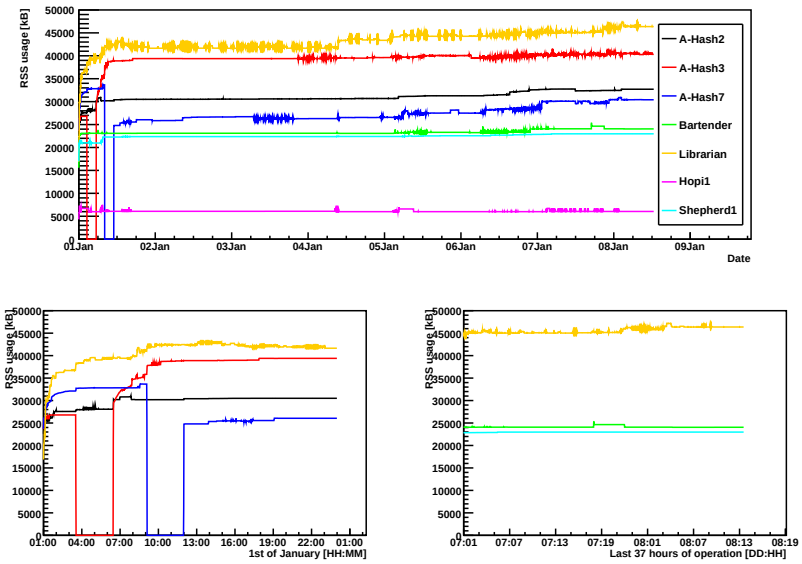


Figure 5: Resident memory utilization of the Chelonia services during an 8 day run.

The most crucial part of Chelonia, when it comes to handling server failures, is the replicated A-Hash. If the A-Hash becomes unavailable, the entire system is unavailable. If a client A-Hash goes down, the Librarians may need to find a new client. If the master A-Hash goes down, the entire system will be unavailable until a new master is elected. The bottom left of Figure 5 shows the memory consumption of the three A-Hashes, A-Hash2, A-Hash3 and A-Hash7, and the Librarian during the first 24 hours of the stability test. When the test started A-Hash3 (red line) was elected as A-Hash master, and the Librarian started using A-Hash7 (blue line) for read operations. After 2.5 hours, A-Hash3 was stopped (seen by the sudden drop of the red line), thus forcing the two remaining A-Hashes to elect a new master between them. While not visible on the figure, the A-Hash, and hence Chelonia, was unavailable for a 10 seconds period during the election, which

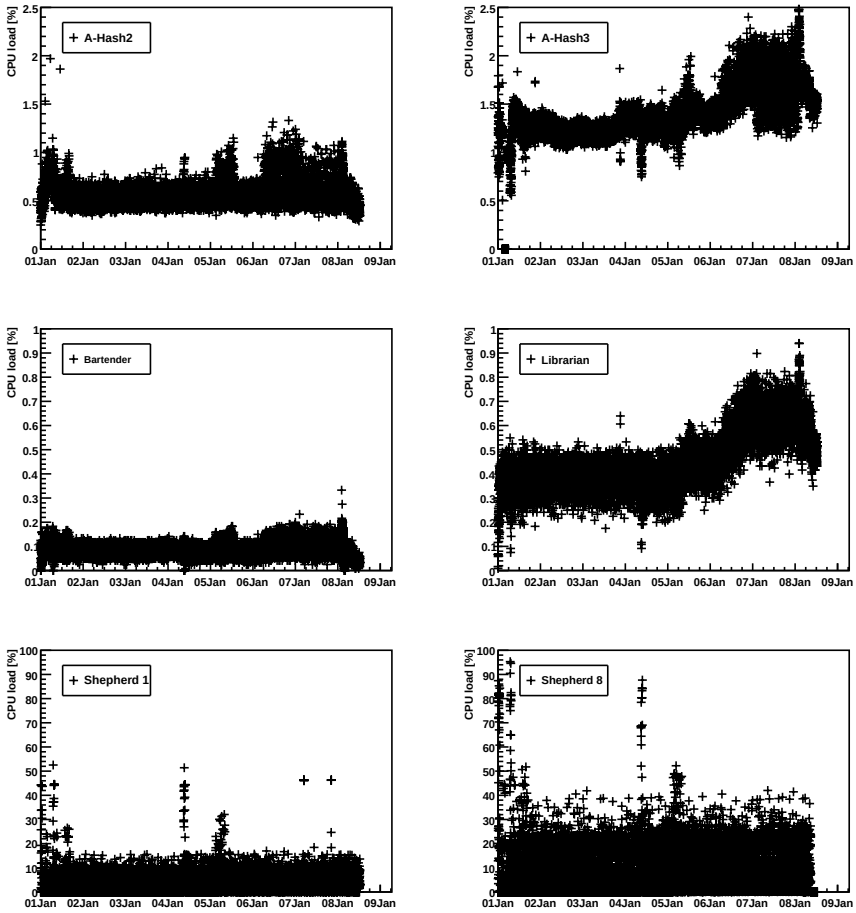


Figure 6: CPU load of the Chelonia services during an 8 day run. Each point is an average of the CPU usage of the previous 60 seconds.

incidentally was won by A-Hash2 (black line). After three additional hours, A-Hash3 was restarted, thus causing an increase of memory usage for the master A-Hash as it needed to update A-Hash3 with the latest changes in the database. Eight hours into the test, the same restart procedure was carried out on A-Hash7 which was connected to the Librarian. This time there was no noticeable change in performance. However A-Hash3 increased memory usage when the Librarian connected to it.

The bottom right of Figure 5 shows the memory usage of the Bartender, the Librarian and one of the shepherds in the last 37 hours of the test. Perhaps most noteworthy is that the memory usage is very stable. The main reason for this is due to the way Python, the programming language of Chelonia, allocates memory. As memory allocation is an expensive procedure, Python tends to allocate slightly more memory than needed and avoids releasing the already acquired memory. As a result, the memory utilization gets evened out after a period of time even though the usage of the system varies. During the run-time of the test, files of different sizes were periodically uploaded to and deleted from the system. The slight jump in memory usage for the Bartender (green line) was

during an extraordinary upload of a set of large files (1GB). This jump was followed by an increase in memory for the Librarian when the files were starting to be replicated between the Shepherds, thus causing extra requests to the Librarian.

Figure 6 shows the CPU load for six of the services. While the load on the A-Hashes, the Bartender and the Librarian are all below 2.5% of the CPU, Shepherd-1 (bottom left) and Shepherd-8 (bottom right) use around 10% and 20%, respectively. While the difference between the Shepherds is due to the fact that Shepherd-1 was run on a server with twice the number of CPU's as the server of Shepherd-8, the difference between the Shepherds and the other services is due to the usage pattern during the test. To confirm that the files stored on the storage node are healthy, the Shepherd calculates a checksum for each file, first when the file is received and later periodically. In periods where no new files are uploaded, the Shepherds use almost no CPU as already stored files don't need frequent checksum calculations. However, when files are frequently uploaded, deleted and re-uploaded, as was the case during the test, the number of checksum calculations, and hence the CPU load, increases significantly. This can particularly be seen on January 1 and 4 when a set of extra large files were uploaded, causing spikes in the CPU load of the two Shepherds. Note also that the spikes occurs at the same time for both Shepherds, as should be expected in a load-balanced system.

5. Related Work

There are a number of grid and cloud storage solutions in the market, focused on different storage needs. While direct performance comparison with Chelonia is beyond the scope of this paper, some similarities and differences between Chelonia and related storage solutions are worth mentioning.

While Chelonia is designed for geographically distributed users and data storage, Hadoop [3] with its file system HDFS is directed towards physically closely-grouped clusters. HDFS builds on the master-slave architecture where a single NameNode works as a master and is responsible for the metadata whereas DataNodes are used to store the actual data. Though similar to Chelonia's metadata service, the NameNode cannot be replicated and may become a bottleneck in the system.

When compared to typical grid distributed data management solutions, the closest resemblance with Chelonia is the combination of the storage element Disk Pool Manager (DPM) and the file catalog LCG [2] File Catalog (LFC) [12]. By registering all files uploaded to different DPM's in LFC one can achieve a single uniform namespace similar to the namespace of Chelonia. However, where Chelonia has a strong coupling between the Bartenders, Librarians and Shepherds to maintain a consistent namespace, DPM and LFC have no coupling as the registration and replication of files is handled on the client side. If a file is removed or altered in DPM, this may not be reflected in LFC. In Chelonia, a change of a file has to be registered through the Bartender and propagated to the Librarian before it is uploaded to the Shepherd.

Scalla [9] differs from Chelonia as it is designed for use on centralized clusters, while Chelonia is designed for a distributed environment. Scalla is a widely used software suite consisting of an xrootd server for data access and an olbd server for building scalable xrootd clusters. Originally developed for use with the physics analysis tool ROOT [10] , xrootd offers data access both

[2]LHC (Large Hadron Collider) Computing Grid

through the specialized xroot protocol and through other third-party protocols. The combination of the xrootd and olbd components offers a cluster storage system designed for low latency, high bandwidth environments. In contrast, Chelonia is optimized for reliability, consistency and scalability at some cost of latency and is more suitable for the grid environment where wide area network latency can be expected to be high.

In the cloud storage family, Amazon Simple Storage Service (S3) [2] promises unlimited storage and high availability. Amazon uses a two-level namespace as opposed to the hierarchical namespace of Chelonia. In the security model of S3, users have to implicitly trust S3 entirely, whereas in Chelonia users and services need to trust a common independent third party Certificate Authority.

6. Conclusion and Future Work

The Chelonia storage system is a simple-to-use storage solution with grid capabilities. The presented tests are designed to give an understanding of how Chelonia behaves in a real-life environment while at the same time controlling the environment enough to get interpretable results. The tests have shown that Chelonia can handle both deep and wide hierarchies as expected. The system has shown self-healing capabilities, both in terms of service and of file availability. Multiple clients have accessed the system simultaneously with reasonable performance. The features and the capabilities empower Chelonia to be comparable with other storage solutions.

In addition to the continuous process of improvements and code-hardening (based on user feedback) there are plans to add some new features. The security of the current one-time URL based file transfers could be improved by adding to the URL a signed hash of the IP and the DN of the user. In this way the file transfer service could do additional authorization, allowing the file transfer only for the same user with the same IP.

Because of the highly modular architecture of both Chelonia and the ARC HED hosting environment, the means of communication between the services could be changed with a small effort. This would enable less secure but more efficient protocols to replace HTTPS/SOAP when Chelonia is deployed inside a firewall. This modularity also allows additional interfaces to Chelonia to be implemented easily. For example, an implementation of the WebDAV protocol would make the system accessible to standard clients built into the mainstream operating systems.

Another possible direction for enhancing the functionality of Chelonia is to add handling of databases. In addition to files, the system could store database objects and use databases as storage nodes to store them. Relational databases allow running extensive queries to get the desired information. In a distributed environment, high availability and consistency is often ensured by the replication of data. Access to multiple copies of the data in the system also allows queries to be run in parallel.

7. Acknowledgements

We wish to thank Mattias Ellert for vital comments and proof-reading. Additionally, we would like to thank UPPMAX, NIIF and USIT for providing resources for running the storage tests. The

work has been supported by the European Commission through the KnowARC project (contract nr. 032691) and by the Nordunet3 programme through the NGIn project.

References

[1] Advanced Resources Connector. `http://www.nordugrid.org/arc/`.

[2] Amazon Simple Storage Service. `http://s3.amazonaws.com`.

[3] Apache Hadoop. `http://hadoop.apache.org/`.

[4] Chelonia Web page. `http://www.nordugrid.org/chelonia/`.

[5] EU KnowARC project. `http://www.knowarc.eu/`.

[6] Filesystem in Userspace. `http://fuse.sourceforge.net/`.

[7] NorduGrid Collaboration. `http://www.nordugrid.org/`.

[8] Oracle Berkeley DB. `http://www.oracle.com/technology/products/berkeley-db/index.html`.

[9] Chuck Boeheim, Andy Hanushevsky, David Leith, Randy Melen, Richard Mount, Teela Pulliam, and Bill Weeks. Scalla: Scalable Cluster Architecture for Low Latency Access Using xrootd and olbd Servers. Technical report, Stanford Linear Accelerator Center, 2006.

[10] Rene Brun and Fons Rademakers. ROOT – An object oriented data analysis framework. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 389(1-2):81–86, April 1997.

[11] D. Cameron, M. Ellert, J. Jönemo, A. Konstantinov, I. Marton, B. Mohn, J. K. Nilsen, M. Nordén, W. Qiang, G. Rőczei, F. Szalai, and A. Wäänänen. *The Hosting Environment of the Advanced Resource Connector middleware*. NorduGrid. NORDUGRID-TECH-19.

[12] Akosh Frohner. Official Documentation for LFC and DPM. `https://twiki.cern.ch/twiki/bin/view/LCG/DataManagementDocumentation`.

[13] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[14] Zs. Nagy, J. K. Nilsen, and S. Toor. *Chelonia - Self-healing distributed storage system*. NorduGrid. NORDUGRID-TECH-17.

[15] Zs. Nagy, J. K. Nilsen, and S. Toor. *Chelonia Administrator's Manual*. NorduGrid. NORDUGRID-MANUAL-10.

[16] J. K. Nilsen, S. Toor, Zs. Nagy, and B. Mohn. Chelonia – A Self-healing Storage Cloud. In M. Bubak, M. Turala, and K. Wiatr, editors, *CGW'09 Proceedings*, Krakow, 2 2010. ACC CYFRONET AGH. ISBN 978-83-61433-01-9.