Chelonia: A self-healing, replicated storage system

2011 J. Phys.: Conf. Ser. 331 062019

(http://iopscience.iop.org/1742-6596/331/6/062019)

Journal of Physics: Conference Series **331** (2011) 062019 doi:10.1088/1742-6596/331/6/062019

Chelonia: A self-healing, replicated storage system

Jon Kerr Nilsen[1], Salman Toor[2], Zsombor Nagy[3] and Alex Read[1]

[1] Department of Physics, University of Oslo
[2] Department of Information Technology, Uppsala University
[3] National Information and Infrastructure Development Institute, NIIF/HUNGARNET

E-mail: j.k.nilsen@fys.uio.no, zsombor@niif.hu, salman.toor@it.uu.se, a.l.read@fys.uio.no

Abstract. Chelonia is a novel grid 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 gravitating towards the cloud paradigm. Chelonia is an integrated system of heterogeneous, geographically dispersed storage sites which is easily and dynamically expandable and optimized for high availability and scalability. The architecture and implementation in term of web-services running inside the Advanced Resource Connector Hosting Environment Dameon (ARC HED) are described and results of tests in both local -area and wide-area networks that demonstrate the fault tolerance, stability and scalability of Chelonia will be presented. In addition, example setups for production deployments for small and medium-sized VO's are described.

1. Introduction

During recent years a number of storage projects have addressed the demands of managing extremely large data volumes (see, e.g., [1, 2, 3, 4]). While these projects provide systems with many sophisticated features, they tend to require a fair amount of resources in terms of software, hardware and human expertise. Even though the resource requirements can often be unfeasible for smaller user groups, the user requirements related to storage systems are the same. The system should be user-friendly and highly-available. It must be consistent, secure, and reliable. Thus, there is a need for a storage solution which provides the basic features of huge-scale storage systems while reducing the requirements in terms of management and resources. The Chelonia storage system [5, 6] is a community effort from NorduGrid, the team behind the Advanced Resource Connector [7] (ARC). Chelonia is designed to provide a self-healing, secure and reliable system which aggregates geographically distributed storage nodes with minimal efforts in deployment and management.

The article is organized as follows. Section 2 gives the features provided by Chelonia while Section 3 describes the system components and functionality. A first deployment of Chelonia and some identified issues will be presented in Section 4 before concluding with a discussion of the future direction of Chelonia in Section 5.

2. Why Chelonia: The Benefits

One of the main benefits of Chelonia is the easy deployment. Installation can be done from binary packages which exist for a wide range of operating systems (several Linux flavors, Mac and Windows too). The configuration can be done with easy to understand INI-style configuration files. For most cases the initial setup consists of choosing a node type (metadata node or storage node), choosing a directory on the local filesystem where Chelonia can store its data, specifying the URLs of the metadata nodes, and acquiring X.509 certificates. In special cases a very fine-grained configuration is also possible with a detailed XML file. While Chelonia works as an interplay of multiple services, they can be grouped as metadata node services and storage node services, which makes it easier to understand and handle.

Another benefit of Chelonia is that it is easily extendable. Adding a new storage node does not require configuration at the metadata nodes or the other storage nodes. This makes it easy to just install a new node, set the URLs of the metadata nodes in the configuration file of this new node, and start it. The new node will register itself, and it will be considered next time a new file is uploaded or an existing file needs more replicas. This process to extend the number of storage nodes is simple.

In terms of maintenance, Chelonia is self-healing both in terms of the metadata and the data. The metadata nodes provide a replicated and consistent way to store the metadata. The metadata about file replicas is kept synchronized with the state of the actual replicas on the storage nodes, which means that if the state of the replica changes on the storage node, this information will eventually be changed in the metadata automatically. All files in the system have the number of required replicas specified, and Chelonia automatically maintains the number of replicas by creating new ones and deleting unneeded ones. Thus the system requires very little administration effort to keep it running.

A low-maintenance storage system needs to recover easily from failures. In case of the failure of a metadata node, the system keeps running with the remaining nodes (if it has at least three metadata nodes), and simply restarting the failed node will make it rejoin the network, catch up with the changes, and start to function again automatically. The failure of a storage node will make the replicas of that node unaccessible, but because the files have more replicas on other storage nodes, they will be still available to the users. By restarting the failed node, it can rejoin the other storage nodes, even if its files are lost. New replicas will be created in order to maintain the specified number.

3. How Chelonia Works

Chelonia has a web-service-based architecture built on four services, all hosted by the ARC Hosting Environment Daemon [8] (HED) and each responsible for a dedicated set of operations. Together, they store files in a global, hierarchical namespace. Standard X509 certificates are used both for client-service authentication and for inter-service authentication. The communication, between the services and between the clients and the system, is based on SOAP over HTTPS. The data transfer can be done over, e.g., HTTPS. The access to Chelonia is provided with three different clients. First, a light-weight command-line interface provides full interaction with Chelonia. Second, a FUSE module provides a graphical user interface, albeit with a limited set of features with respect to access modification. Third, an ARC Data Management Component provides access for grid jobs through the ARC protocol.

Chelonia consists of four services, as shown in Figure 1. Here, the Bartender (cup), the Librarian (book) and the A-Hash (space-ship) comprise the metadata services in the top three circles, while the Shepherds (staffs) manage the storage nodes in the bottom three services.

Each instance of the Shepherd service manages a particular storage node and provides a uniform interface for storing and accessing file replicas. On a storage node there must be at least one independent storage element service (with an interface such as HTTP(S)) which performs

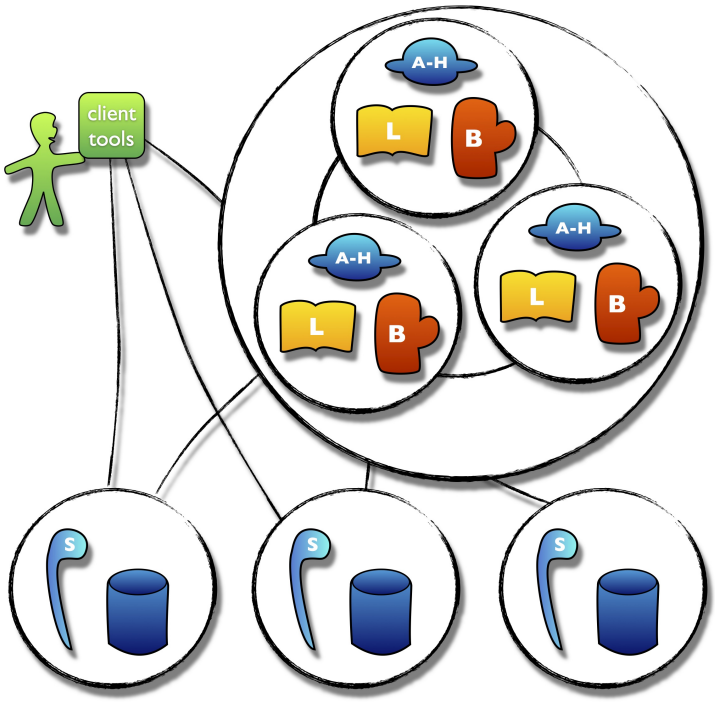

Figure 1. Schematic of Chelonia's architecture. The figure shows the main services of Chelonia; The Bartender (cup), the Librarian (book), the A-Hash (space-ship) and the Shepherd (staff). The communication channels are depicted by black lines and the circles separate the metadata services (top three circles) and the storage services (bottom three circles).

the actual file transfer. A storage node then consists of a Shepherd service and a storage element service connected together. Storage element services can either be provided by ARC or by third-party services. For each kind of storage element service, a Shepherd backend module is needed to enable the Shepherd service to communicate with the storage element service, e.g., to initiate file uploads, downloads and removal, and to detect whether a file transfer was successful or not. Currently there are three Shepherd backends: One for the ARC native HTTP(S) server called Hopi; one for the Apache web server; and one for a service which implements a subset of the ByteIO interface.

The Librarian service manages the hierarchy and metadata of files and collections, handles the Logical Names and monitors the Shepherd services. The Librarian service is stateless, instead it stores all the persistent information in the A-Hash, a replicated, yet simple, key-value database which only supports setting, getting and deleting entries. This makes it possible to deploy any number of independent Librarian services to provide high-availability and load-balancing. In this case all the Librarians should communicate with the same set of A-Hashes in order to use the same database of metadata. As only the master A-Hash can be written to and the Librarian cannot know a priori which A-Hash replica is the master, the Librarian needs to get this information from one of the A-Hashes. For this reason, the master A-Hash stores the list of all available A-Hashes, so that the information is replicated to all A-Hash replicas. As all A-Hash replicas are readable, the Librarian only needs to know about a single, functioning A-Hash at start-up to be able to get this list. During run-time the Librarian holds a local copy of the A-Hash list and refreshes it both regularly and in the case of a failing connection.

The Bartender service provides a high-level interface of the storage system for the clients

(other services or users). The clients can create and remove collections (directories), create, get and remove files, and move files and collections within the namespace using Logical Names. Access policies associated with files and collections are evaluated by the Bartender every time a user wants to access them. The Bartender communicates with the Librarian and Shepherd services to execute the clients' requests. The file content itself does not go through the Bartender; file transfers are directly performed between the storage nodes and the clients. The Bartender also supports so-called gateway modules which make it possible to communicate with third-party storage solutions, thus enabling the user to access multiple storage systems through a single Bartender client. These modules are protocol-oriented in the sense that external storage managers which support a certain protocol will be handled using the gateway module based on that protocol. While excluding some of the features provided by accessing storage managers directly, this approach reduces the number of gateway modules required for different storage managers. The currently available gateway module is based on the GridFTP protocol [9].

The process of replication is essential to ensure reliability and high availability. While designing the architecture of Chelonia, three areas of replication were identified to be required to provide reliable services. First, the replication of data is needed to ensure that the data is available in case of storage node failures. Second, the replication of metadata is needed to ensure that the stored data is always accessible. Third, the replication of the system components is needed to avoid single-points-of-failures in the system. For the replication of data, the Shepherds manage file replicas, periodically checking the health of the files and whether there are enough replicas. If not, the Shepherds automatically initiate replication of the files in question. The metadata is replicated by the A-Hash using the Berkeley DB [10] High-Availability mode. This mode uses a master-slave model, where only the master is available for write-request while the slaves are available for read-requests, and a consensus approach, where a write-request only returns when a majority of the slaves have acknowledged the database-update. The ability to have multiple instances of all the services in the system makes the architecture less prone to hardware failures. For example, if a Bartender goes offline, the clients can direct the request to any other Bartender in the system. Similarly, in case of one Librarian being unavailable, the Bartenders can contact any of the other Librarians in the system without affecting the user experience.

4. Deployment and Issues

Chelonia is still in a relatively fresh state and needs real-life testing and hardening before being considered as a stable, production-ready storage system. Systematic tests concerning stability and performance have been undertaken, both on local and wide-area networks and have shown promising results (for further details, see [6]). Systematic tests can, however, never replace a real deployment. The remainder of this section describes a test-deployment of one realistic use-case for a storage system directed towards smaller user communities, and the issues found during the deployment.

The use-case considers a group of users with a number of desktops and a need to share the spare storage space of these desktops in one common name-space. The spare storage space has no scheduled back-up and individual hard-drives may be erased at any time due to circumstances not directly controlled by the users, e.g., due to re-installation of the operating system. As the individual resources are evidently unstable, replication of all files and services in the system is mandatory. For this purpose, Chelonia was deployed on 13 desktops at the group for Experimental Particle-Physics at the University of Oslo. Each of the 13 desktops had an extra 1 TB hard-drive and all the desktops were in daily use. The deployment was set up with three metadata servers, each hosting one Bartender, one Librarian and one replicated A-Hash. The remaining desktops were used for physical data storage with one Shepherd instance per desktop so that the deployment had 10 TB storage space in total.

After an initial period of optimization and fine-tuning the deployment performed reasonably well under stable conditions, with a response-time for file-upload in the order of $0.1 - 0.6$ s per file when uploading thousands of files. However, when introducing instabilities in the system a number of issues appeared, most of which are traceable to two major issues:

- **The interaction between the Librarians and the Shepherds** A major issue seen during deployment appeared when restarting a Shepherd which stored a few hundred files. This caused one of the Librarians and the master A-Hash to be heavily loaded to the point of having to stop the entire system. The reason for this behavior is the following. When the Librarian notices that a Shepherd is down, it will mark all the corresponding file replicas as offline. When the Shepherd comes online again, the Librarian asks the Shepherd to send all information about its file replicas so that the Librarian can mark the file replicas that survived the offline period as online. This checking is done in a separate thread in the Librarian, and the issue occurs when the time to process the information from the Shepherd is too long. The separate thread will then repeat the request to the Shepherd, which again sends all information, causing the Librarian to work even harder.
- **The Replicated A-Hash** The replicated A-Hash is built around the replication features of Berkeley DB. During deployment, two major issues related to the way Berkeley DB is currently used in Chelonia where noted: First, one of the reasons for using a replicated metadata-base in Chelonia is the possibility to distribute the work-load between the replica-servers by allowing the Librarians to read from different servers. However, with the Berkeley DB, consistency can only be guaranteed when reading from the master replica. Second, Chelonia is designed to have an arbitrary number of A-Hash replicas. However, to avoid so-called split-brain problems, i.e., situations where the replicas are separated to accidentally form separate replica sets, Berkeley DB uses a variant of the Paxos algorithm [11] to ensure that only a replica-set with the majority of replicas is allowed to be updated and accessed. To know the size of the majority, the exact number of metadata replicas must be known at system start-up time.

5. Conclusion and Future Directions

This paper has outlined the features and the architecture of the Chelonia grid storage system. A real-life deployment has been described and the issues it uncovered have been discussed.

Learning from the above-mentioned test deployment, there is clearly work needed to be done to decrease the amount of communication between the Librarian and the Shepherd. In a setup where the goal is to collect storage space from many desktops, it is unfortunate that large chunks of information are pushed from the Shepherds to the Librarian, as this can soon overload the Librarians and the master A-Hash. Another lesson learned is that the replicated A-Hash as currently used in Chelonia may lead to inconsistencies, and requires either changes in the configuration of the underlying Berkeley DB or finding an alternative to Berkeley DB.

In addition to the continuous process of improvements and code-hardening based on user feedback from test deployments such as the one described above, some new features are considered. First, The security of the current one-time URL based file transfers may 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 can do additional authorization, allowing the file transfer only for the same user with the same IP. Second, because of the highly modular architecture of Chelonia, the means of communication between the services may be changed with a small effort. This can 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 can make the system accessible to standard clients built into the mainstream operating systems.

Regardless of the problems seen while setting up the described deployment of Chelonia, and the time it will take to solve the problems, the grid storage market needs an easily deployable, low-maintenance, distributed storage system. Seeing that this niche remains to be filled in the grid storage market, the development of Chelonia will continue.

References

[1] G Behrmann, P Fuhrmann, M Gronager, and J Kleist. A distributed storage system with dcache. *Journal of Physics: Conference Series*, 119(6):062014 (10pp), 2008. doi: 10.1088/1742-6596/119/6/062014.

[2] 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. `http://xrootd.slac.stanford.edu/papers/Scalla-Intro.htm`.

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

[4] M Branco, D Cameron, B Gaidioz, V Garonne, B Koblitz, M Lassnig, R Rocha, P Salgado, and T Wenaus. Managing ATLAS data on a petabyte-scale with DQ2. *Journal of Physics: Conference Series*, 119(6):062017 (9pp), 2008. doi: 10.1088/1742-6596/119/6/062017.

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

[6] Zs. Nagy, J. K. Nilsen, S. Toor, 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.

[7] M. Ellert et al. Advanced Resource Connector middleware for lightweight computational Grids. *Future Gener. Comput. Syst.*, 23(1):219–240, 2007. doi: 10.1016/j.cam.2006.05.008.

[8] 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 `http://www.nordugrid.org/documents/ARCHED_article.pdf`.

[9] Bill Allcock, Joe Bester, John Bresnahan, Ann L. Chervenak, Ian Foster, Carl Kesselman, Sam Meder, Veronika Nefedova, Dary Quesnel, and Steven Tuecke. Data management and transfer in high-performance computational grid environments. *Parallel Computing Journal*, 28:749–771, 2001. doi: 10.1.1.16.4686.

[10] `http://www.oracle.com/technetwork/database/berkeleydb/`. Oracle Berkeley DB.

[11] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998. doi: 10.1145/279227.279229.