Journal of Physics: Conference Series **396** (2012) 032027 doi:10.1088/1742-6596/396/3/032027

A Scalable proxy cache for Grid Data Access

Traian Cristian CIRSTEA[1], Jan Just KEIJSER, Oscar Arthur KOEROO, Ronald STARINK, Jeffrey Alan TEMPLON

Nikhef, Science Park 105, 1098 XG Amsterdam, The Netherlands
[1] Stan Ackermans Institute, Technische Universiteit Eindhoven, Eindhoven, NL

E-mail: templon@nikhef.nl

Abstract. We describe a prototype grid proxy cache system developed at Nikhef, motivated by a desire to construct the first building block of a future https-based Content Delivery Network for grid infrastructures. Two goals drove the project: firstly to provide a "native view" of the grid for desktop-type users, and secondly to improve performance for physics-analysis type use cases, where multiple passes are made over the same set of data (residing on the grid). We further constrained the design by requiring that the system should be made of standard components wherever possible. The prototype that emerged from this exercise is a horizontally-scalable, cooperating system of web server / cache nodes, fronted by a customized webDAV server. The webDAV server is custom only in the sense that it supports http redirects (providing horizontal scaling) and that the authentication module has, as back end, a proxy delegation chain that can be used by the cache nodes to retrieve files from the grid. The prototype was deployed at Nikhef and tested at a scale of several terabytes of data and approximately one hundred fast cores of computing. Both small and large files were tested, in a number of scenarios, and with various numbers of cache nodes, in order to understand the scaling properties of the system. For properly-dimensioned cache-node hardware, the system showed speedup of several integer factors for the analysis-type use cases. These results and others are presented and discussed.

1. Introduction

In 2010, the Worldwide LHC Computing Grid (WLCG) collaboration formally started investigating how to evolve the strategies, technologies, and frameworks for data and storage management used by the LHC experiments. *Content Delivery Networks* was one of the technologies identified as potentially interesting during the kickoff workshop of this investigation.

At the time of the workshop, data "on the grid" was generally being pre-placed at computing centers around the world, in preparation for sending the grid jobs to process these data: the jobs were being run under the assumption that the data were located on-site and were available. If for some reason the data were not available — for example, a disk server was temporarily down or overloaded — the job would fail. Statistics on data access showed that a large fraction of the pre-placed data were never accessed. Two consequences of the prevailing data management and access strategy were then a relatively high failure rate, and a rather poor usage profile for the scarce and expensive disk space available to the experiment collaborations.

Content Delivery Networks (CDNs) often operate in the reverse fashion: there is no pre-placement. Data are delivered to the user via the network when requested, and any of a number of algorithms are used to decide when data are used "often enough" to warrant replication to sites "close" (in a network sense) to the user. The "missing data" of the previous paragraph is,

in a CDN, a normal phenomenon, the network being designed to deal with remote access and replication.

CDNs deal with both the problems we listed with grid data management: application failure due to missing data does not occur (unless the master copy of the data has been removed!) and space is not wasted, as data that are not used are never placed. The challenge that CDNs must, as a consequence, face is the problem of efficient delivery of data – data that an application requests, but is not located on-site – to the user's application without the user needing to take any special action. In the most general case, the application uses a single identifier (*e.g.,* URL) for each piece of data, regardless of the where the application is running.

CDNs generally are implemented using two main components. The first is a network of proxy caches, located strategically "close" to the places where the applications are being run. The second is a network of DNS servers, that when queried for a DNS lookup resulting from an application trying to access a data URL, returns to the application the IP address of a suitably close proxy cache.

This paper describes a prototype design and implementation of such a proxy cache. What is specifically new to this proxy cache, is that it is designed to be able to proxy access to data located on a grid storage element, and that it propagates permissions from the grid world onto the cache layer. Note that this prototype only addresses the design and implementation of a single proxy cache; networking these caches, and constructing the associated DNS network, only makes sense if a suitable proxy cache for grid data can be demonstrated. The intent of this paper is to convince the reader that such a demonstration has been accomplished.

2. Constraints and Design Decisions

The project was to develop a prototype, and this should be achieved in a nine-month period due to external circumstances. We made this possible to achieve by limiting the design space very early in the project. The following choices were made:

(i) the identifiers by which data will be accessed, are their logical file names as registered in the grid file catalog (LFC)

(ii) the access protocol is https

(iii) the cache should also be mountable via the webDAV protocol

(iv) access permissions should be identical to those using the standard WLCG tools. WLCG uses grid proxies extended by VOMS (Virtual Organization Membership Service) attributes as the basis for authorization, hence our system must understand these attributes.

(v) the system design should allow for scaling in throughput and number of connections

An additional decision was to attempt to write as little code as possible, using existing software unless absolutely necessary due to some missing functionality. This decision is motivated by the observation that the "home-grown" data access protocols and software used in the HEP world generate a sizeable support load, in terms of person power necessary to maintain the codes.

3. Initial Exploration of Design Space

Rough prototypes were made for many possible versions of the system, looking at *e.g.* different paradigms for load balancing, different mechanisms for handling permissions, different web servers on which to base the system. Readers wishing a full description of these explorations and our conclusions should consult the technical report of the project [1]. In the interest of space and clarity we present here only some information on previous work (especially why it was not sufficient for our purposes) and the final design of the prototype.

4. Previous Work

The generic term for the product we describe here is a "reverse proxy cache" [2]. We also include the additional functionality that the system should be mountable as a file system.

A prototype of the reverse proxy cache functionality was already available in Scalla/xrootd, see for example [3]. However when this project started (early 2011), the xrootd system only supported its own protocol, there was no support for http(s). The xrootd proxy cache system was undergoing rapid evolution at the time, hence we chose not to attempt extending it to support https as a protocol.

Both `squid` and `varnish` are well known and supported products supporting reverse proxy functionality in the http(s) world. If the grid storage elements had all supported https access at the time, these could have been used; however this was not the case. It turned out to be quite involved to extend squid or varnish to properly handle grid proxies, especially when delegation of a user credential is concerned. In our case this is what is needed, the cache is intended to serve multiple VOs, hence the system needs to be able to present a delegated user credential to the LFC or a grid storage element when retrieving data via gridFTP.

Much of the needed functionality regarding grid authentication was already present in Apache plus GridSite. Adding caching functionality and a webDAV server as CGI scripts turned out to be much less effort than adapting one of the available proxy cache products to support the necessary grid security.

Concerning the "mount the grid as a filesystem", there have been several previous projects. Most of them, for example ELFI [4], gridfs [5], and GS^3 [6] are based on use of the FUSE kernel module and hence provide (some) caching only at the client level, and are limited to linux systems. dCache already supported webDAV access at the beginning of this project [7], however a large fraction of the data in the grid is hosted on non-dCache storage elements.

5. System Design

Figure 1 shows a high-level representation of the system design. A webDAV server handles the connections to the system via webDAV, allowing a user to "mount" or "browse" the grid storage as seen by the grid file catalog (LFC). If data are requested, the request is redirected to one of a cooperating group of cache nodes that speak https. This node then a) fetches the data from the grid (if it is not already in cache), b) serves it to the user, and c) caches it for later use.This arrangement is intended to allow for the required scaling in throughput and number of client connections. For a compact installation, however, a single node can function as both webDAV server and cache node.

Figure 2 shows a protocol view of how the various components interact.

6. Control and Data Flow for Common Operations

This section describes how the various system components interact with each other. This provides an opportunity to illustrate the design choices we made regarding load balancing, which we consider one of the important aspects of the system. Where possible, we took a "passive" approach in which normal system behavior results in a natural balancing under load, rather than an active system that attempts to monitor the load and takes specific actions to redistribute it. Our experience is that the former approach results in higher system stability, and is also more forgiving when it comes to unexpected usage patterns as well as unexpected failure or degradation in parts of the system. Passive stability is a frequent feature of control systems [8]; a simple everyday example is the relative placement of the steering axis *vs.* the rotational axis for the front wheels of a car — the placement chosen yields passive stability, which is why (if your car is properly maintained) it is possible to remove your hands from the steering wheel for a few moments without the car veering off the road.

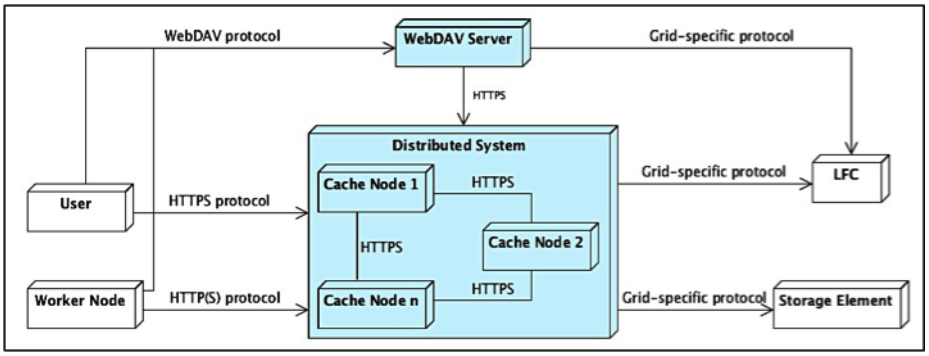

Figure 1. Architecture diagram. The system we constructed is shaded blue in this figure. The arrows show the direction in which requests flow between the various components when initiated by a user request.

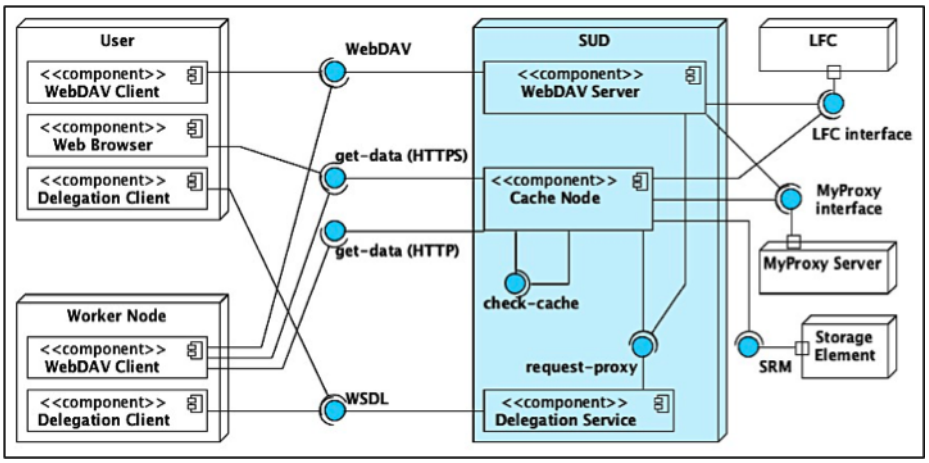

Figure 2. Illustration of communication protocols between various parts of the system and with external components.

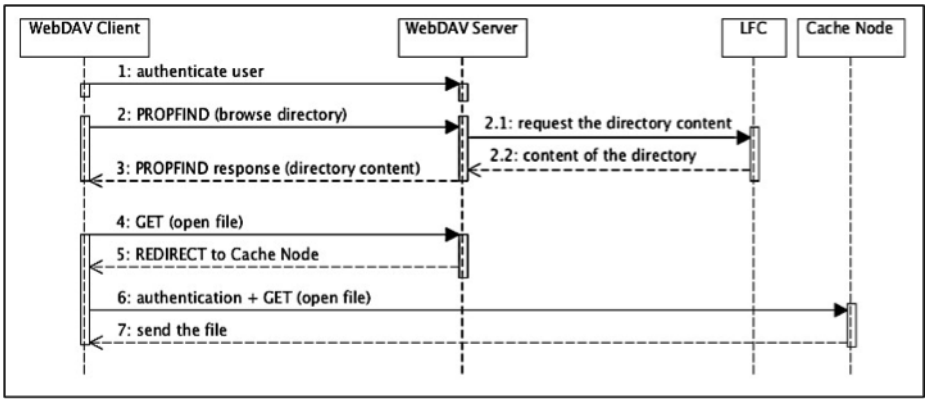

Figure 3. Sequence diagram corresponding to client interaction with the system via the webDAV protocol.

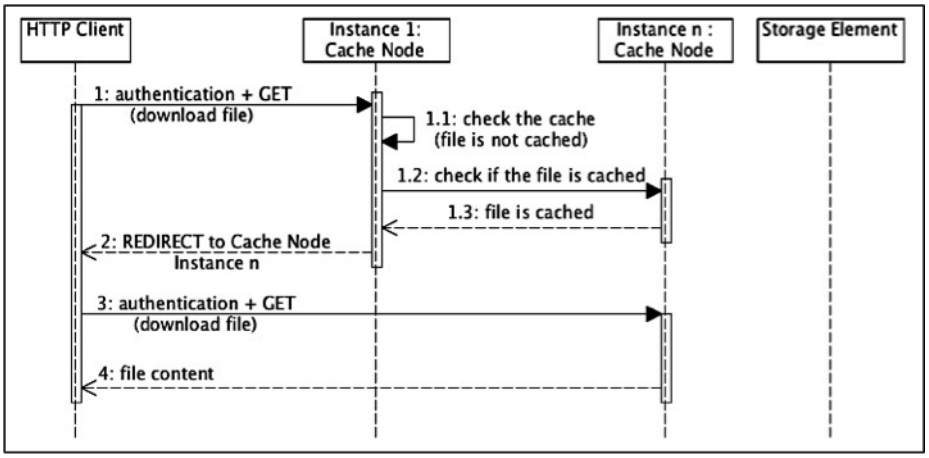

Figure 4. Sequence diagram for client access via https to one of the cache nodes, for the case where the data are cached on one of the other cache nodes.

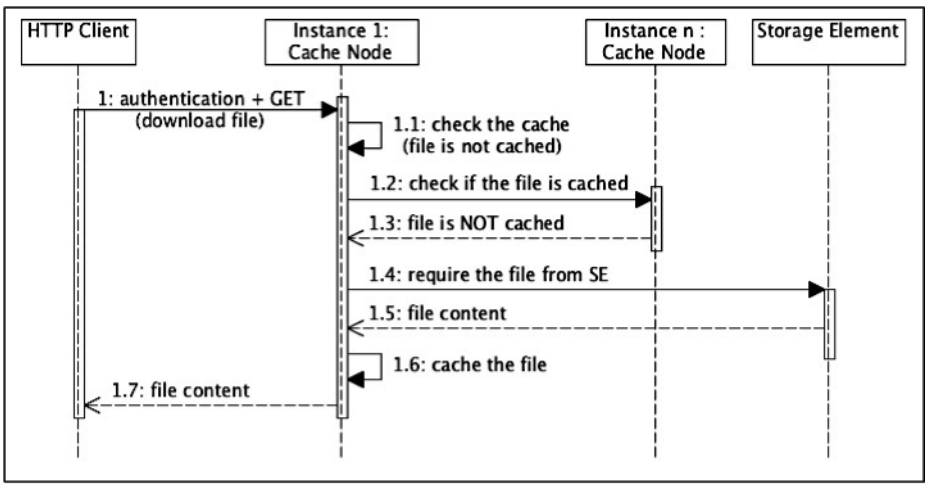

Figure 5. Sequence diagram for client access via https to one of the cache nodes, for the case where the data are not cached on any of the cache nodes.

Figure 3 shows the sequence of interactions following from directory browsing and file access via a webDAV client. The diagram depicts the case in which the client supports http REDIRECT (not all webDAV clients do). For this case, the important load-balancing feature is that the REDIRECT (step 5 in the figure) is to a *random* cache node, this evenly distributes load (throughput and client connections) across all available cache nodes. Furthermore this random redirect, as opposed to consulting a central catalog, avoids having the central webDAV server itself be a bottleneck. Finally, this choice results in excellent behavior in the case of a burst of requests for a particular "new" file : due to the random redirects, all cache nodes in the system will wind up retrieving a copy of the file, handling the burst in the most effective manner.

For clients not supporting REDIRECT, the webDAV server proxies the data from the cache node to the client.

A further level of load balancing takes place between the cache nodes themselves. For this please refer to figures 4 and 5, which expand upon steps 6 and 7 in figure 3 for two cases: the

data being requested is resident in the cache (figure 4) or not (figure 5). When the randomly chosen cache node receives a request for data, the following steps are taken:

(i) the cache node checks to see whether it holds a copy of the data. If so, it serves the data to the user. If not:
(ii) the cache node broadcasts a query to all other cache nodes, asking they hold a copy of the data
(iii) the client is redirected to the first node that answers "yes" within a specified timeout, this node serves the data
(iv) if no answer is received, the cache node retrieves the file from the grid, and serves it to the user.

This sequence responds correctly to various load situations in the following ways:

(i) under low loads, a cache node holding a copy of the requested data will respond promptly to the broadcasted query, this results in a redirect to that cache node.
(ii) if that particular cache node holding the copy is too slow to respond, an additional copy (on a different node) will be fetched from the grid
(iii) in the case that multiple nodes hold a copy of the file, the least loaded of them will serve the file, as it will be the one responding most quickly to the broadcast.

The timeout was chosen by observing the effect on performance; the value used for the tests described below was 2 seconds. While we did not perform any dedicated measurements that we could present here, we informally observed a slowing of a node's response to queries, as the number of connected clients increased. Each connected client consumes bandwidth, disk i/o, and cpu, the combination of which presumably results in the slower response.

7. Implementation

The prototype consist of a number of readily-available open source components, plus about 700 lines of glue code. Apache2 was used as a basis for implementing both the webDAV server and the cache nodes, using the mod_gridsite plugin to deal with the GSI authentication. The webDAV server was written as a FCGI application under Apache2. The reason to write a server, rather than reuse one, was our requirement of support for REDIRECT. The caching is also managed by an FCGI application under Apache2.

Cache management in this prototype is rather simple and is triggered in two cases. In the first case, the total amount of required data (actively being used) at a particular moment exceeds the allowed size of the cache. In this case, all transfers in process are aborted, a "file not found message" is returned to the clients, and all partial transfers are deleted. In the second case, a request is received for a file not in cache, but the amount of space left in the cache area is not sufficient to hold this file, the system starts deleting data by last-access time (oldest first) until there is enough space for the new file.

Permissions management was also handled by a rather simple mechanism. For the case where the file requested is not yet cached in the system, the mechanism is extremely simple: it is the user's delegated credential which is used to retrieve the file from a storage element. If the user does not have permission to access the file, the transfer fails and the user does not get the file. If the user does have permission to get the file, this permission is cached as metadata to the SURL for a tunable period (in our prototype, five minutes).

For cases where the file is on cache, it is possible that the permissions may have changed, which is why the permission is only cached for a short period. It might also be the case that the user asking for the file is not the one whose request brought it in cache. In that case, or if the cached permission has expired, the permission is checked by querying the SE whether the

Table 1. Hardware characteristics of the proxy cache nodes. Each contained two Intel Xeon L5520 processors running at 2.27GHz.

Proxy cache nodes tbn{21,22,28}.nikhef.nl			
three nodes	Total Cores	8	
	RAM Memory	24	GB
	Local Disk	493	GB
	: throughput (read)	120	MB/s
	: throughput (write)	40	MB/s
	Network connectivity	10	Gb/s

Table 2. Hardware characteristics of the worker nodes used to run the jobs. Each contained two Intel Xeon L5520 processors running at 2.27GHz.

Worker nodes tbn{23,24..27,29...35,36}.nikhef.nl			
thirteen nodes	Total Cores	8	
	RAM Memory	24	GB
	Local Disk	493	GB
	: throughput (read)	120	MB/s
	: throughput (write)	40	MB/s
	Network connectivity	10	Gb/s

requesting user may access the SURL in question. If the answer from the SE is "no" then the system does not serve the file to the user.

8. Performance Testing

The system was deployed at Nikhef and subjected to medium-scale testing: the test cluster could support up to 104 concurrent clients. For the grid data, a local storage element (serving files from two disk server machines) was used; for the cache, we tested configurations of one, two, and three cache nodes. Tables 1, 2, and 3 show the characteristics of the machines participating in the tests. Please note that the cache nodes are not specially optimized for their function; the disk IO rates are in particular not exceptional. For our medium-scale tests of this prototype, this choice is not unreasonable; discovering bottlenecks in an optimized system might require testing at a larger scale. Unless otherwise specified, the files being retrieved were each 1 GB in size.

The storage element was running the glite "DPM" software. For both the lcg-cp and webDAV transfers, proper authentication and authorization was carried out, however the bulk data transfer was in both cases unencrypted.

Figure 6 depicts the setup graphically and shows the network bandwidths available between all these systems.

Four main tests were performed. At the core, each test consisted of multiple clients (grid jobs) that perform an md5 checksum of a grid file. For reference purposes, each test was first performed with the grid data file being retrieved to local disk by the glite `lcg-cp` command, and running the md5 checksum on the local copy. For all other tests using the cache system, the job followed these steps:

Table 3. Hardware characteristics of the storage element disk server nodes. Each contained two Intel Xeon L5520 processors running at 2.27GHz.

Storage Element disk server nodes hooimaand-{01,02}.nikhef.nl			
two nodes	Total Cores	8	
	RAM Memory	72	GB
	Local Disk	240	GB
	: throughput (read)	600	MB/s
	: throughput (write)	100	MB/s
	Network connectivity	10	Gb/s

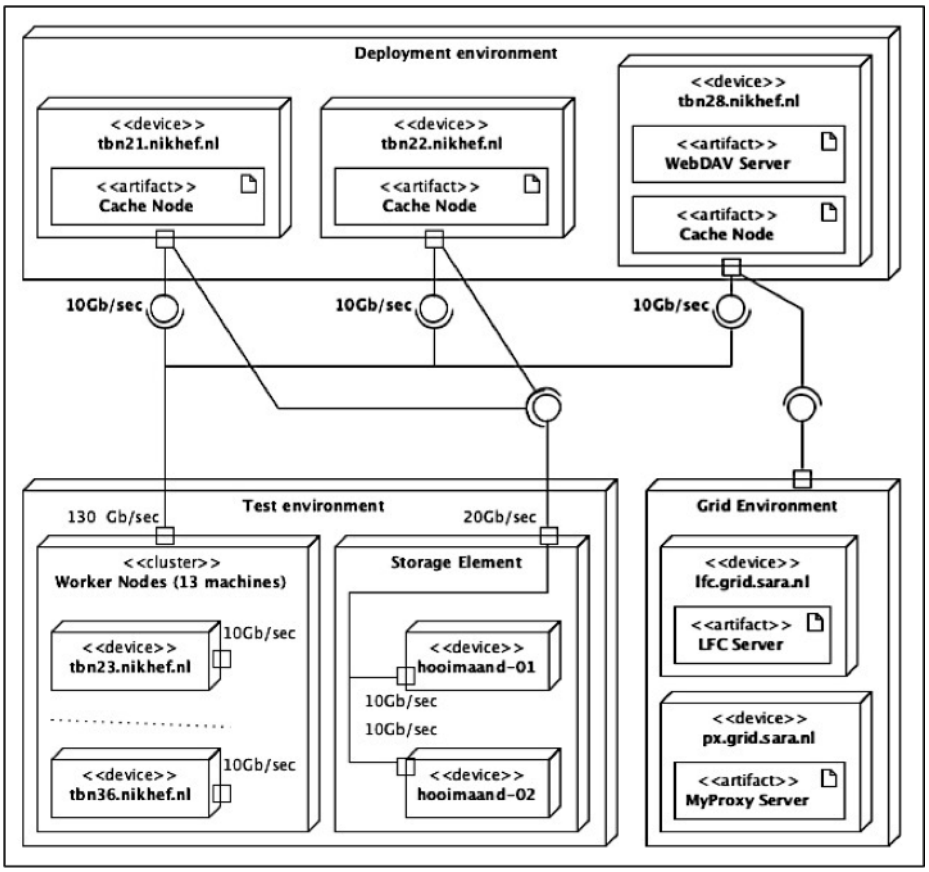

Figure 6. Illustration of the test environment in which the results described below were measured. The proxy cache system is in the box at the top. The lower left box contains the worker nodes and storage element disk servers used for the test; the lower right box shows external grid services such as the LFC and MyProxy servers. The connecting lines illustrate the network bandwidth available between components. The bandwidth limitation between the worker nodes and storage element is completely determined in this case by that of the storage element (10 Gb/s per disk server).

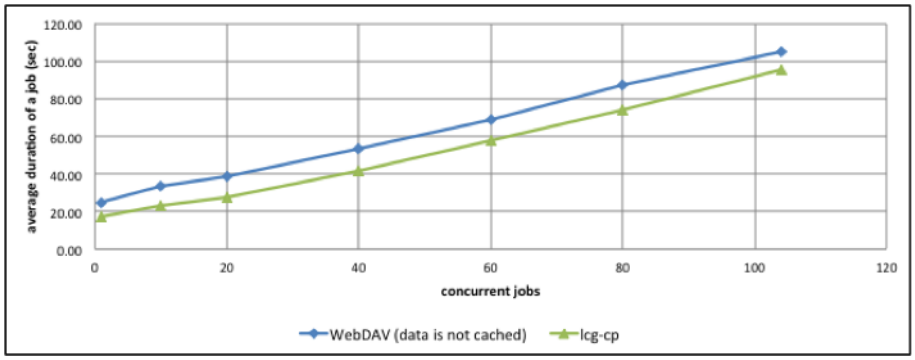

Figure 7. Results for average job duration (seconds) as a function of the number of concurrent jobs run, for the case of a single cache node, all jobs requesting the same file, and the file not being present on cache at the start of the test.

(i) delegate the grid credential from the WN to the webDAV server
(ii) mount the webDAV view on the WN
(iii) run the md5 sum over the file via the posix mountpoint
(iv) unmount the repository

Average job durations were measured, as a function of the number of concurrent jobs, with either a) all jobs requesting the same data, or all requesting different data and b) a single node functioning as both webDAV server and cache node, vs. the addition of two cache-only nodes. Furthermore, each test was done for the cases of data already cached or not. Finally, each test was repeated using lcg-cp (retrieval by the job, directly from the storage element) for comparison.

8.1. Single cache node, all clients request same file
Figures 7 and 8 show the measured performance of the system for the case when all clients request the same data file (one gigabyte) and the system includes a single cache node. One can see for the case where the data is not already cached, our system takes about 13 seconds longer than does a direct lcg-cp to copy the file from the grid. This overhead of 13 seconds is mostly due to the time needed to retrieve a copy of the data from the grid onto the cache node.

For the case where the data is already resident on the cache node, the performance is identical to that of lcg-cp. This is only so because the network bandwidth between the cache node and worker nodes is the same as that between the storage element disk servers and the worker nodes. If the SE were connected via a slower (WAN) network, or had been heavily loaded, the cache probably would have performed much better than lcg-cp based access. On the other hand, the penalty for a cache miss (listed above as 13 seconds) would be increased.

8.2. Multiple Cache Nodes, all clients request same file
Figure 9 shows the measurements for the case of all clients requesting the same file, from a system of three cache nodes (of which one doubles as the webDAV server). Note that for the case where the data are not cached, at the start of the run, none of the cache nodes have a copy of the requested file, hence due to the load-balancing properties described earlier, all three cache nodes retrieve a copy of the file.

For the case in which the data is already present in cache, access speed via our system is superior to that via lcg-cp at all scales tested. This can be simply explained as arising from the

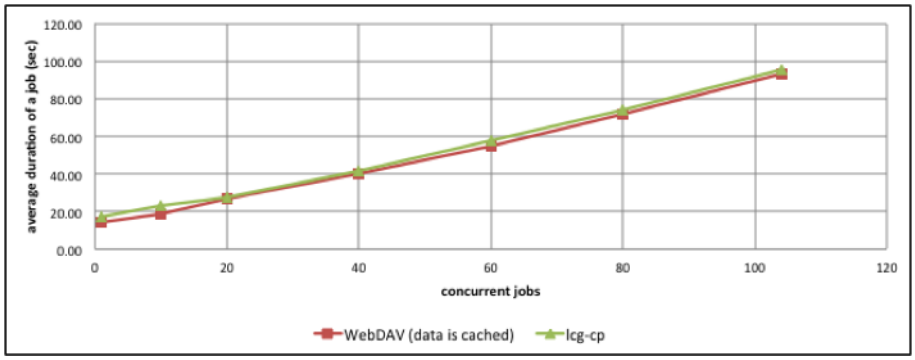

Figure 8. Results for average job duration (seconds) as a function of the number of concurrent jobs run, for the case of a single cache node, all jobs requesting the same file, and the file already being present on cache at the start of the test.

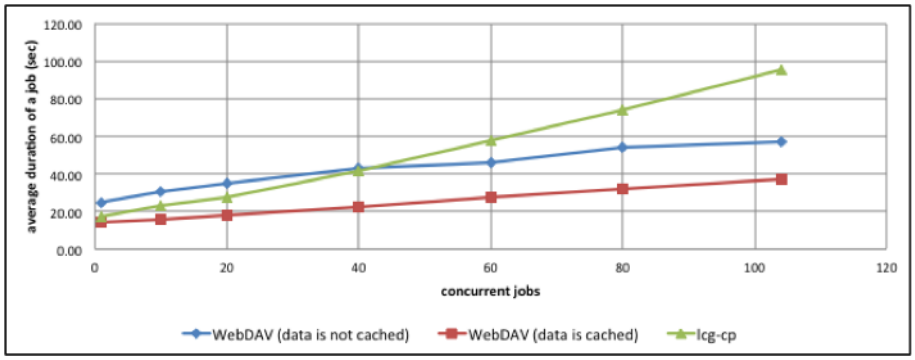

Figure 9. Results for average job duration (seconds) as a function of the number of concurrent jobs run, for the case of three cache nodes, and all jobs request the same file. The blue line shows the measurement for the file not being present in the cache system at the start of the run, for the red line, the file was already present at the start.

larger network bandwidth offered by three cache nodes (30 Gb/s) compared to a single storage element disk server (10 Gb/s). For the case of the data not yet being cached, the picture is a bit more complex. For low numbers of concurrent clients, lcg-cp is slightly better, since use of the cache requires the file to be first retrieved from the grid, after which the transfer of data to the user can start. The break-even point is at 40 concurrent clients, after which the cache outperforms lcg-cp, again due to the larger aggregate bandwidth: the speed increase in serving the data to the job on the worker node more than compensates for the delay incurred in transferring the file from the grid to the cache.

8.3. Single Cache Node, all clients request a different file

Figure 10 shows the performance measurements for the case where each job requests a unique file. These files are distributed evenly across the two disk servers (see figure 6). Note that the results for lcg-cp are essentially the same as in the previous tests, despite being served by two disk servers compared to one. This appears to result from disk i/o becoming a limiting factor on the storage elements' disk servers. In the previous case, the single requested data file need only be read from disk once, it could be served to the clients directly from filesystem cache in

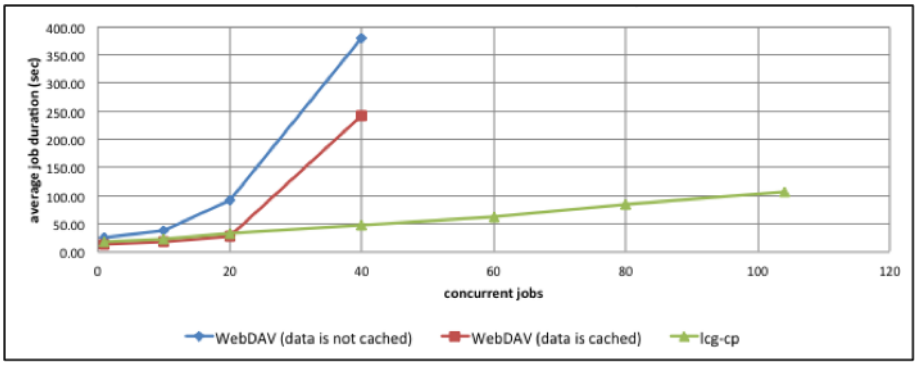

Figure 10. Results for average job duration (seconds) as a function of the number of concurrent jobs run, for the case of a single cache node, and all jobs request unique (different) files. The blue line shows the measurement for the file not being present in the cache system at the start of the run, for the red line, the file was already present at the start.

memory.

Comparing the cache system to lcg-cp, we see that when the data are not cached, the peformance when using the cache system already starts to degrade at about 10 concurrent clients. This degradation is however not seen for the case of the data already present on the cache node.

This degradation accelerates after about 20 concurrent clients, in both cases (data already cached, or not cached). The initial degradation appears to result from the poorer disk performance of the cache node compared to the disk server nodes; it is not present when the data are already present on the cache node, due to the the data still being in OS cache from the previous tests. The second degradation appeared to be linked to

- the disk performance of the cache nodes. The maximum throughput for reading is 120 MB/s. Forty jobs reading a 1 GB file from a single server at this rate will take at least 330 seconds (compare the observed 390).
- the size of installed RAM on the cache node, in this case 24 GB.

For 20 jobs, 20 GB of data must be transferred. This 20 GB can all fit into the operating system's file-system cache, minimizing the amount of disk i/o required. For more than 20 jobs, the operating system cache becomes increasingly less effective and performance is increasingly limited by the disk I/O rates.

To conclude, in the case of high demand of varied data from the system, the performance is very low compared with a typical grid storage element (running on suitable hardware), when the amount of active data exceeds the RAM size of the cache node. When the amount of active data less than RAM size, and the data is cached, the system performance is similar to that of the storage element (lcg-cp test). The performance could also be improved by substituting a system with better disk I/O performance.

8.4. Multiple Cache Nodes, all clients request a different file

Figure 11 shows the same test using three cache nodes instead of one. The behavior is the same, except that now the "serious" performance degradation starts somewhere between 60 and 80 concurrent clients. For the three cache nodes, the total RAM size is 72 GB, hence the behavior here is consistent with the previous measurement: when the data are already on cache, and the total size of the requested data is less than the total available RAM, the cache

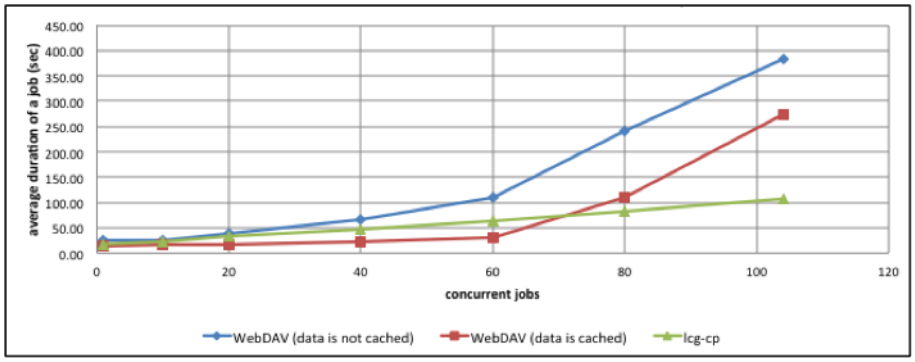

Figure 11. Results for average job duration (seconds) as a function of the number of concurrent jobs run, for the case of three cache nodes, and all jobs request unique (different) files. The blue line shows the measurement for the file not being present in the cache system at the start of the run, for the red line, the file was already present at the start.

system outperforms a grid storage element in serving the data. This was further verified by repeating the "single cache node" test with half of the RAM disabled; performance then began to deteriorate at 10 concurrent clients instead of 20.

9. Conclusions

We have shown that it is possible to construct a grid proxy cache system using mostly standard software like the Apache2 webserver. The system preserves the authorization restrictions present on the data in the grid domain.

For the worst cases – when the requested data are not already in cache, and there are no repeat requests – the system does not perform significantly worse than direct use of a grid storage element, as long as the hardware limits of the cache system have not been reached. We have also shown that these hardware limits can be extended by simply adding more cache nodes.

In the best case – frequent requests for the same files – we have shown data access via the cache has the same performance as direct access for a single-cache-node system; addition of more cache nodes has an essentially linear effect on performance when the number of clients is large enough. Note that our tests did not include cached access to data residing on remote storage elements; it is likely that in that case, even a single-node cache will outperform direct grid access.

10. Further Work

One of the most difficult practical aspects of the current project was the lack of support for GSI proxy certificates in "standard" software like Apache2. Many of these tools do support X.509 proxy certificates. The system would be less complex if it only had to support X.509 and not GSI.

The teams behind many of the grid storage-element software products are now working on support for https access to the data. Such support would create an opportunity to make the current system simpler and more performant. The "retrieve file from the grid" step would become simpler (use of standard protocol). The more important advantage is that, when a user requests a file that is not present on cache, the system can now offer, in addition to the current "retrieve file and serve to the user" service, a redirection of the client to the off-site location of the file (using a standard http REDIRECT). This is faster for the user (one transfer instead of two). Heuristics can be used to decide, based on access patterns, whether it is advantageous to

retrieve a copy to place on the cache.

References

[1] Cirstea T C 2011 *Grid Data Access: Proxy Caches and User Views* (*Eindhoven University of Technology SAI Technical Report*) ISBN 978-90-444-1067-9.
[2] *Reverse Proxy*, `http://en.wikipedia.org/wiki/Reverse_proxy`
[3] *USCMS T3 Xrootd Architecture* , `https://twiki.cern.ch/twiki/bin/view/Main/T3Xrootd`
[4] Corso E, Cozzini S, Leto A, Messina A, Murri R, Di Meo R, and Terpin A 2006 *1st Int. Workshop on Grid Technology for Financial Modeling and Simulation*
[5] Maad S, Coghlan B, Quigley G, Ryan J, Kenny E, and O'Callaghan D 2007 *Future Generation Computer Systems* **23** 123
[6] Cunsolo V D, Distefano S, Puliafito A, and Scarpa M L 2010 *J. Grid Computing* **8** 391
[7] Berhmann G 2010 `http://www.dcache.org/articles/i,article-20100114001.html`
[8] *Passivity (engineering)*, `http://en.wikipedia.org/wiki/Passivity_(engineering)#Stability`