StashCache: A Distributed Caching Federation for the Open Science Grid

Derek Weitzel
Marian Zvada
dweitzel@unl.edu
zvada@unl.edu
University of Nebraska – Lincoln
Lincoln, Nebraska

Ilija Vukotic
Rob Gardner
ivukotic@uchicago.edu
rwg@uchicago.edu
University of Chicago
Chicago, Illinois

Brian Bockelman
bbockelman@morgridge.org
Morgridge Institute for Research
Madison, Wisconsin

Mats Rynge
rynge@isi.edu
Information Sciences Institute
Los Angles, California

Edgar Fajardo Hernandez
emfajard@ucsd.edu
San Diego Super Computer Center
La Jolla, California

Brian Lin
Mátyás Selmeci
blin@cs.wisc.edu
matyas@cs.wisc.edu
University of Wisconsin – Madison
Madison, Wisconsin

arXiv:1905.06911v1 [cs.DC] 16 May 2019

ABSTRACT

Data distribution for opportunistic users is challenging as they neither own the computing resources they are using or any nearby storage. Users are motivated to use opportunistic computing to expand their data processing capacity, but they require storage and fast networking to distribute data to that processing. Since it requires significant management overhead, it is rare for resource providers to allow opportunistic access to storage. Additionally, in order to use opportunistic storage at several distributed sites, users assume the responsibility to maintain their data.

In this paper we present StashCache, a distributed caching federation that enables opportunistic users to utilize nearby opportunistic storage. StashCache is comprised of four components: data origins, redirectors, caches, and clients. StashCache has been deployed in the Open Science Grid for several years and has been used by many projects. Caches are deployed in geographically distributed locations across the U.S. and Europe. We will present the architecture of StashCache, as well as utilization information of the infrastructure. We will also present performance analysis comparing distributed HTTP Proxies vs StashCache.

ACM Reference Format:
Derek Weitzel, Marian Zvada, Ilija Vukotic, Rob Gardner, Brian Bockelman, Mats Rynge, Edgar Fajardo Hernandez, Brian Lin, and Mátyás Selmeci. 2019. StashCache: A Distributed Caching Federation for the Open Science Grid. In *Practice and Experience in Advanced Research Computing (PEARC '19), July 28-August 1, 2019, Chicago, IL, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3332186.3332212

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
PEARC '19, July 28-August 1, 2019, Chicago, IL, USA
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7227-5/19/07…$15.00
https://doi.org/10.1145/3332186.3332212

1 INTRODUCTION

The Open Science Grid (OSG) provides access to computing resources distributed throughout the U.S. to perform data analysis. A portion of users on the OSG are opportunistic users: they do not own the computing and storage resources which they use. While they continue to use the computing resources of the OSG, the data sizes have increased faster than the existing infrastructure can support.

Non-opportunistic users of the OSG are members of large experiments such as the Large Hadron Collider's ATLAS [1] and CMS [6]. These experiments own and maintain resources accessed through the OSG. Each experiment manages the distribution of data to dozens of storage sites near the computing resources. Since the experiments own storage, it is clear to users where to store their data and how to access data from the experiment. Further, the large experiments have written complex software systems to manage the distributed storage.

On the other hand, opportunistic users do not have dedicated resources for their storage needs and therefore must use storage resources they do not own. There are competing interests in the use of opportunistic resources, the resource owners and the opportunistic users. The resource owner may want to reclaim space from the opportunistic user. If the owner reclaims the space, the opportunistic user must discover that their data has been removed. Further, the opportunistic user is responsible to transfer the data to multiple sites and to periodically check if the data is still available at these sites. The opportunistic user has an interest to acquire the required storage for their workflow while not over-burdening the storage owner.

When a user's processing requires significant input data, they can transfer the data from the submit host or from a centralized server. Transferring the data with each job puts network and I/O load on the submit node, starving it of resources to serve it's primary purpose to maintain the workflows run by the user. As the user increases the number of jobs running and requesting data, the submit node becomes the bottleneck. Transferring data from a central service

has similar bottlenecks as transferring from the submit host, though it leaves the submit host to only manage the user workflow.

Users of the OSG may use reverse HTTP proxies maintained by the large experiments such as CMS [3]. The HTTP proxies are distributed across the OSG near the computing resources. But, the HTTP proxies have well known limitations [15]. One such limitation is that the proxies are optimized for small files such as software [4] and experiment conditions [11] rather than the multi-gigabyte files that some users require. In this paper, we will present performance metrics for the use of StashCache over HTTP proxies for large files.

StashCache provides an easy interface for opportunistic users to distribute data to their processing. It provides a caching layer between the researcher data location and the execution hosts requiring the data. The caches are distributed throughout the OSG, and are guaranteed to have at least 10Gbps networking and several TB's of caching storage.

A cache provides many advantages over persistent storage space. For users, a cache is simple to maintain since the data stored is transient and does not require maintenance. The resource provider can reclaim space in the cache without worry of causing workflow failures when processing next tries to access the data.

2 BACKGROUND

StashCache is comparable to a Content Distribution Network (CDN), which is used by hosting services to provide fast delivery of data to clients. A CDN consists of a collection of (non-origin) servers that attempt to offload work from origin servers by delivering content on their behalf[16]. Popular commercial CDNs are Akamai[18], Cloudflare[7] and Fastly[12].

Content Distribution Networks have many of the same goals as StashCache:

- Cache data from a set of origins
- Provide fast delivery of data to clients
- Cache near the client

Most CDNs are optimized for the delivery of web assets such as web pages, images and videos. These items are usually small in size, less than a gigabyte. For example, Cloudflare will only cache files up to 512MB [8] and the cloud CDN Azure - Akamai service will by default only cache files up to 1.8GB [17]. For some users of StashCache, the average file size is over 2GB. Further, commercial CDNs do not have access to the high speed research networks within the U.S.

Accessing data available from many sources has been implemented in large experiments with frameworks such as Any data, any time, anywhere[2] and Federating ATLAS storage using XRootD [14]. Federated storage allows clients to access data from several sources in a consistent namespace. A client requests a file's location from a data location service, which will query storage servers to determine where the file is located.

3 IMPLEMENTATION

The StashCache infrastructure consists of four components shown in Figure 1:

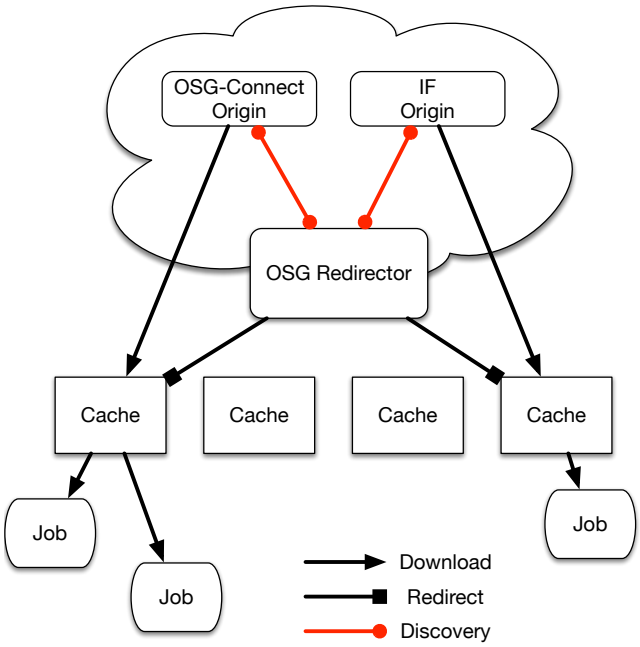

Figure 1: StashCache Architecture: Jobs request data from caches, which in turn query the redirector for the data location. Data is transferred from the origin to the cache, and then to the job.

- **Data Origin:** The authoritative source of data within the StashCache federation. Each organization or experiment could have an origin.
- **Data Caches:** Regional servers that cache data from the origins before sending to the clients.
- **Redirector:** Serves as the data discovery service. Caches query the redirector for the location of data.
- **Clients:** Requests data from the data caches. The clients are responsible for finding the nearest cache using GeoIP.

Data origins are installed on the researcher's storage. The origin is the authoritative source of data within the federation. Each Origin is registered to serve a subset of the global namespace. Caches will contact the origin in order to retrieve and cache the data when it has been requested by a client. The origin is a service, built upon the XRootD [10] software that exports datasets to the caching infrastructure. The origin subscribes to a central XRootD redirector that is used by the caches to discover which origin has the requested data.

Caches also use XRootD to capture data requests from clients, download data from the origins, and to manage the cache space. The caches receive data requests from the client, check the local cache, and if necessary locate and download the requested data from the origins. The caches locate the data by contacting the XRootD redirector, which in turn asks the origins if they have the requested data. Figure 2 shows the locations of the caches in the U.S.

The redirector serves as the data discovery service. Caches query the redirector to find which origin contains the requested data. The redirector will query the origins in order to find the data and return

the the hostname of the origin that contains the data to the caches. The redirector is a central service within StashCache hosted by the Open Science Grid. There are two redirectors in a round robin, high availability configuration.

3.1 Clients

Two clients are used to read from the StashCache federation. The CERN Virtual Machine File System (CVMFS) [5] and stashcp [23]. In a previous paper, we highlighted how the Laser Interferometer Gravitational-Wave Observatory (LIGO) used the CVMFS client to access the StashCache federation [22].

CVMFS provides a read-only POSIX interface to the StashCache federation. It appears to users as a regular filesystem on the execute hosts. When a user reads data from CVMFS, it will request the data from the caches and cache the data locally. CVMFS determines which cache to contact by using its built-in GeoIP locator.

The posix interface that CVMFS provides allows it to view exactly what data the application requires. CVMFS will download the data in small chunks of 24MB. If an application only reads portions of a file, CVMFS will only download those portions.

CVMFS is configured to only cache 1GB on the local hard drive. There are two reason for this low cache amount. The working set size of data transferred through the CVMFS client is expected to be too large to be reasonably accommodated on an execute node. Also it is expected that transferring from a nearby cache is fast enough to serve the data to applications.

In order to provide a posix interface to remote storage, metadata such as the directory structure and size of files must be determined. We wrote an indexer which will scan a remote data origin and gather metadata about the files. Metadata includes:

- File name and directory structure
- File size and permissions
- Checksum of files along the chunk boundaries

The indexer will detect changes to files by checking the file modification time and file size. If it differs from the previously indexed file, it will reindex the file. The indexer must scan the entire filesystem each iteration, causing a delay proportional to the number of files in the filesystem.

The second client, `stashcp`, is useful when CVMFS is not installed on the execute host or the delay caused by the indexing procedure is unacceptable for the user. It is a simple script that will download data from the nearest cache, utilizing the CVMFS GeoIP infrastructure to find the nearest cache. The interface to `stashcp` is similar to the Linux command `cp`.

`stashcp` attempts 3 different methods to download the data:

(1) If CVMFS is available on the resource, copy the data from CVMFS
(2) If an XRootD client is available, it will download using XRootD clients.
(3) If the above two methods fail, it will attempt to download with `curl` and the HTTP interface on the caches.

If CVMFS is available, then it should be used first as it has the most redundant features, including: built-in GeoIP locating, rate monitoring, and fallback in failures. XRootD client is used as well because it has efficient multi-threaded, multi-stream transfers of data from caches. The fallback is using `curl`, which has the least features of the clients available.

In contrast to CVMFS, it does not provide a POSIX interface to the StashCache data federation. `stashcp` cannot watch the application's read behavior to know what parts of files are required. Instead, `stashcp` will copy the entire file from the cache to the execute machine.

Both the caches and origins are packaged by the Open Science Grid. They are provided in RPM packages[20], Docker containers[9], and Kubernetes[13] configurations.

Further, we have responded to feedback from experienced StashCache administrators and created automatic configuration generation using a registration service. An administrator simply specifies the experiments that the opportunistic storage should support and the configuration for a cache and origin is generated on-demand.

Figure 2: StashCache Locations within the U.S. They are located at six universities and 3 Internet2 PoPs.

StashCache caches are installed at six universities and three Internet2 backbones. In addition, a StashCache cache is installed at the University of Amsterdam.

3.2 Monitoring

We use XRootD's detailed monitoring in order to monitor and account for usage of StashCache. Each StashCache cache sends a UDP packet for each file open, user login, and file close. The collector of this information is complex since each packet contains different information.

- **User Login:** The user login information includes the client hostname, the method of logging in, such as HTTP or xrootd protocol. It also includes whether it was logged in with IPv6 or IPv4. The user is later identified by a unique user ID number.
- **File Open:** Contains the file name, total file size, and the user ID which opened the file. The file is later referred to by a unique file ID number.
- **File Close:** Contains the total bytes read or written to the file, as well as the number of IO operations performed on the file. It contains the file ID from the file open event.

Figure 3 shows the flow of monitoring information. Each XRootD server sends binary UDP packets to the central monitoring collector.

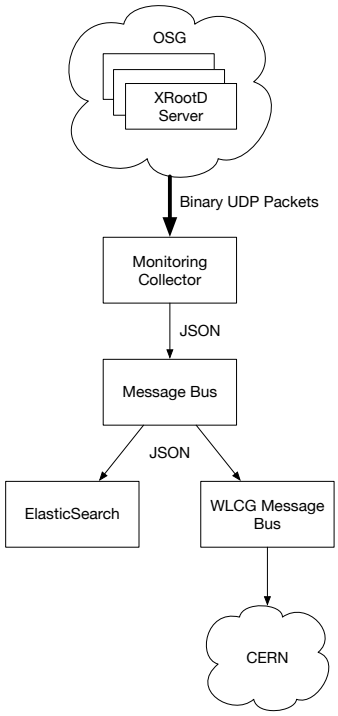

Figure 3: StashCache monitoring flow

The collector combines the different UDP packets to fill in full information for each file transfer. On each file close packet, the collector combines the data from the file open and user login packets and sends a JSON message to the OSG message bus.

The OSG message bus distributes the file monitoring to databases in the OSG and the Worldwide LHC Computing Grid (WLCG). The message is stored in the database for aggregation and analytics.

Figure 4 shows the utilization of StashCache over the last year, from April 2018 to April 2019.

4 EVALUATION

The benefits using StashCache are perceived by both the researchers and the resource owners.

Caches are located at sites that have volunteered opportunistic storage and on the Internet2 network backbone as part of the National Research Platform initiative. Some sites have volunteered storage in order to help opportunistic users. One site, Syracuse University, noticed wide area network usage to other caches and wanted to install a cache locally in order to minimize the network usage of outbound data requests. After installation of a StashCache cache, the site noticed reduced wide area network connections.

Figure 5 shows the Syracuse WAN network bandwidth before and after StashCache was installed. The bold red line shows when the StashCache server was installed. Without the StashCache, Syracuse was downloading 14.3 GB/s of data. After StashCache was installed, the network bandwidth reduced to 1.6 GB/s. Not all of the data being monitored in this graph is StashCache data, other research data was captured; but the reduction in WAN network transfers is clear.

Many researchers use StashCache simply because there is no other way to transfer data to distributed computing resources. They do not own storage resources that could scale to several thousand requests. And they do not have relationships with sites to allow them to borrow storage.

Table 1 shows the top users of StashCache for 6 months ending in February 2019.

Experiment	Usage
Open Gravitational Wave Research	1.079PB
Dark Energy Survey	709.051TB
MINERvA (Neurtrino Experiment)	514.794TB
LIGO	228.324TB
Continuous Testing	184.773TB
NOvA	24.317TB
LSST	18.966TB
Bioinformatics	17.566TB
DUNE (Neutrino Experiment)	11.677TB

Table 1: StashCache Usage

4.1 Comparing StashCache to Distributed HTTP Proxies

We tested the performance of StashCache and of HTTP proxies distributed at sites. We theorize that HTTP proxies will have better performance with small files, while StashCache will be more efficient at large files.

For a test dataset, we created files with sizes representing the file size percentiles from the monitoring. We queried the monitoring for the percentiles from the last 6 months of usage from October 2018 to April 2019. The percentiles are shown in Table 2. Since the 95th and 99th percentile are the same value, we did not create a test file for the 99th percentile. In addition to the percentiles, we also tested with a 10GB file which will show future potential for larger files.

Percentile	Filesize
1	5.797KB
5	22.801MB
25	170.131MB
50	467.852MB
75	493.337MB
95	2.335GB
99	2.335GB

Table 2: StashCache filesize percentiles

The test dataset was hosted on the Stash filesystem at the University of Chicago. The files are available by HTTP and StashCache downloads while on the same filesystem. There are many users of the filesystem, network, and data transfer nodes during our tests which provided realistic infrastructure conditions.

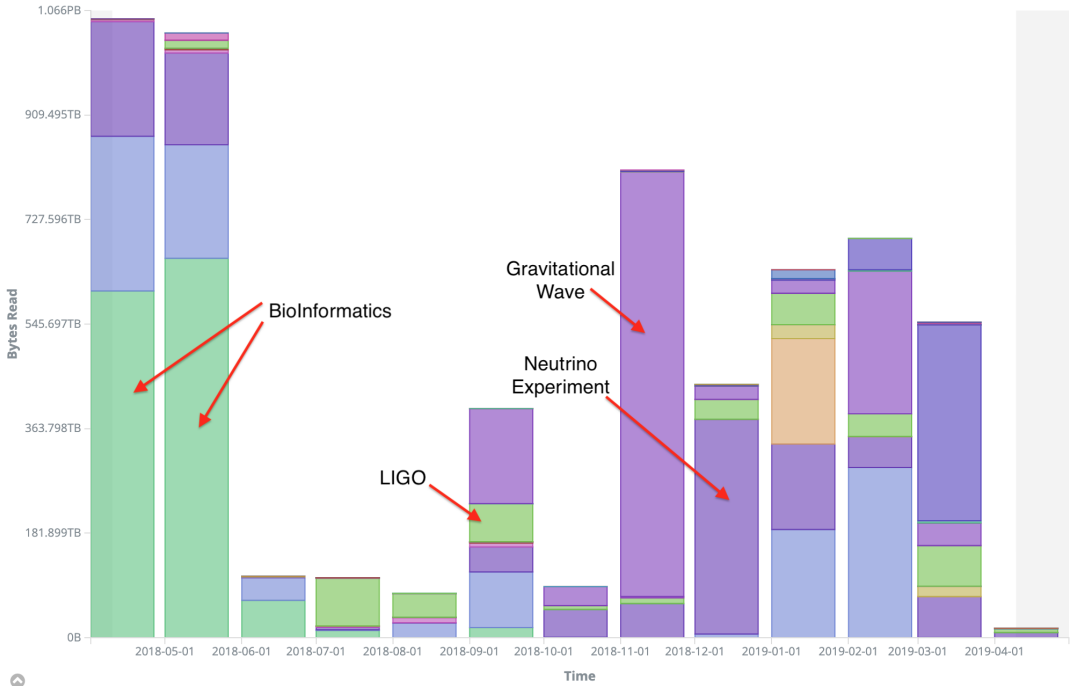

Figure 4: Last 1 Year of StashCache usage.

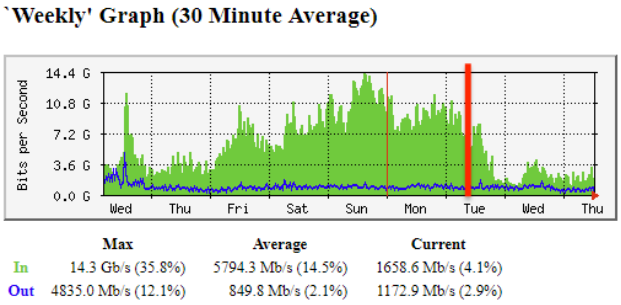

Figure 5: Syracuse Network Usage

We chose to run against the top 5 sites providing opportunistic computing in the last six months on the OSG. Those sites are Syracuse University, University of Colorado, Bellarmine University, University of Nebraska – Lincoln, and the University of Chicago.

We created an HTCondor DAGMan workflow to submit the jobs to each site, without two sites running at the same time. We avoided running two sites at the same time so there was no competition at the data origin for the files.

Each job downloads all files four times. The first time it uses curl to download through the HTTP cache. Since this is the first time downloading this unique file, it is assumed and verified that the first time is a cache miss. It then downloads the file again through the HTTP proxy which will be a cache hit.

The third download is through `stashcp` and the StashCache federation. This will download the file through a cache. The fourth download is again using `stashcp`, but it should be cached.

5 RESULTS & DISCUSSION

During the tests, there were many observations. While running the experiments we experienced expiration of files within the HTTP proxies. Our initial design of the experiments would loop through the list of download files, then loop again to download from the HTTP proxy. After downloading the last large file, the first files were already expired within the cache and deleted.

The HTTP proxies at sites are configured to not cache large files. In all of our tests, the 95th percentile file and the 10GB file were never cached by the HTTP proxies. But they were cached by the StashCache caches.

The tests made clear that some sites have very fast networking from the wide area network from to worker nodes, while others did not. For example, see Figure 6. The bandwidth between the worker nodes and the HTTP proxy is very good. But, the bandwidth to the StashCache caches is consistent but lower than HTTP proxy performance. Some sites prioritize bandwidth to the HTTP proxy, and between the proxy and worker nodes. For example, they have larger bandwidth available from the wide area network to the HTTP proxy than to the worker nodes.

Compare Colorado's performance to Syracuse's performance in Figure 7. You will notice that the cached StashCache is always better than the non-cached. Also, for large data transfers, StashCache is faster than HTTP proxies. This indicates that the worker nodes have higher or equal bandwidth to the StashCache cache and the HTTP proxy.

From the tests, it is clear that HTTP proxies provide superior transfer speeds for small files. In Figure 8 you can notice that StashCache transfer speed for small files is always slower than the HTTP

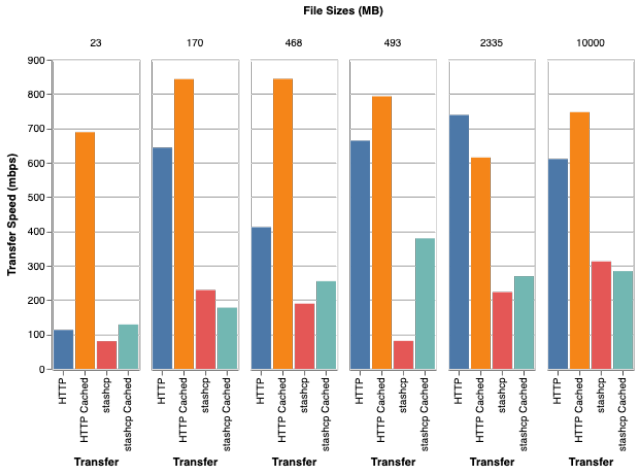

Figure 6: Colorado Cache Performance. Higher is better. Using the HTTP Proxies provide faster download speeds than using StashCache in all filesizes. This could be because the HTTP proxy has fast networking to the wide area network, while the worker nodes have slower networking to the nearest StashCache cache.

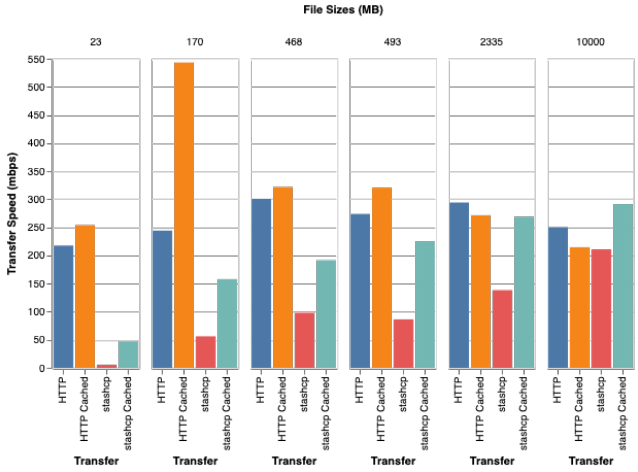

Figure 7: Syracuse Cache Performance. Higher is better. StashCache provides faster downloads for large files, but not for smaller files.

proxy. `stashcp` has a larger startup time which decreases it's average performance. The `stashcp` has to determine the nearest cache, which requires querying a remote server, then can start the transfer. In contrast, the HTTP client has the nearest proxy provided to it from the environment.

For most of the tests, the very large file was downloaded faster with StashCache than a HTTP Proxy. In table 3, you can see the decrease in time it takes to download larger files with StashCache vs. HTTP Proxy. In the table, a negative value indicates a decrease in time to download the file through StashCache. As mentioned above,

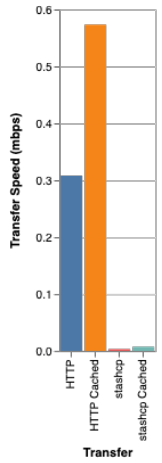

Figure 8: Small File Performance. Higher is better. This is downloading a 5.7KB file. For this small of a file, HTTP performance is much better than StashCache.

Colorado stands alone with very fast performance for downloading through the HTTP proxy.

Site	2.3GB	10GB
Bellarmine	-68.5%	-10.0%
Syracuse	+0.9%	-26.3%
Colorado	+506.5%	+245.9%
Nebraska	-12.1%	-2.1%
Chicago	+30.6%	-7.7%

Table 3: StashCache HTTP Proxies vs StashCache. This is the percent difference between the HTTP Proxy and StashCache. Negative values indicate the time to download decreased when using StashCache

The tools and analysis notebooks are available [21].

6 CONCLUSIONS AND FUTURE WORK

StashCache provides a scalable and transparent method for data distribution for opportunistic users of cyberinfrastructure. From our analysis, we have shown that StashCache has better performance than HTTP proxies for large files. But for small files less than 500MB, HTTP proxies provide better performance.

StashCache provides features not available when using HTTP proxies, such as a posix interface with CVMFS. Also, CVMFS calculates checksums of the data, which guarantees consistency of the data which HTTP proxies do not provide. Also, the HTTP proxy cache can expire rapidly which can cause re-downloads of the data from the origin.

Future testing will be needed to confirm our findings in a variety of infrastructure conditions. We ran our tests over the course of several days and the results may be different in the future. We have no visibility into the resource contention of the network, caches, proxies, or origin server. Future tests should be run over a longer

period of time which will show the performance of StashCache over many conditions.

In the future, we hope to add more capabilities to the StashCache federation. Specifically, we plan to support output data in a write-back cache configuration. Writeback cache will allow users to write output files to a cache rather than back to the origin. Once the files are written to StashCache, writing to the origin will be scheduled in order to not overwhelm the origin.

ACKNOWLEDGMENTS

This research was done using resources provided by the Open Science Grid [19], which is supported by the National Science Foundation award 1148698, and the U.S. Department of Energy's Office of Science. This work was supported by the National Science Foundation under Cooperative Agreement OAC-1836650.

REFERENCES

[1] Georges Aad, JM Butterworth, J Thion, U Bratzler, PN Ratoff, RB Nickerson, JM Seixas, I Grabowska-Bold, F Meisel, S Lokwitz, et al. 2008. The ATLAS experiment at the CERN large hadron collider. *Jinst* 3 (2008), S08003.

[2] Kenneth Bloom, Tommaso Boccali, Brian Bockelman, Daniel Bradley, Sridhara Dasu, Jeff Dost, Federica Fanzago, Igor Sfiligoi, Alja Mrak Tadel, Matevz Tadel, et al. 2015. Any data, any time, anywhere: Global data access for science. In *2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC)*. IEEE, 85–91.

[3] Barry Blumenfeld, David Dykstra, Lee Lueking, and Eric Wicklund. 2008. CMS conditions data access using FroNTier. In *Journal of Physics: Conference Series*, Vol. 119. IOP Publishing, 072007.

[4] B Bockelman, J Caballero Bejar, J De Stefano, John Hover, R Quick, and S Teige. 2014. OASIS: a data and software distribution service for Open Science Grid. In *Journal of Physics: Conference Series*, Vol. 513. IOP Publishing, 032013.

[5] Predrag Buncic, C Aguado Sanchez, Jakob Blomer, Leandro Franco, Artem Harutyunian, Pere Mato, and Yushu Yao. 2010. CernVM–a virtual software appliance for LHC applications. In *Journal of Physics: Conference Series*, Vol. 219. IOP Publishing, 042003.

[6] Serguei Chatrchyan, EA de Wolf, et al. 2008. The CMS experiment at the CERN LHC. *Journal of instrumentation.-Bristol, 2006, currens* 3 (2008), S08004–1.

[7] Cloudflare. 2019. Cloudflare: The Web Performance & Security Company. https://www.cloudflare.com/

[8] Cloudflare. 2019. What's the maximum file size Cloudflare will cache? https://support.cloudflare.com/hc/en-us/articles/200394750-What-s-the-maximum-file-size-Cloudflare-will-cache-

[9] Docker. 2019. Enterprise Application Container Platform | Docker. https://www.docker.com/

[10] Alvise Dorigo, Peter Elmer, Fabrizio Furano, and Andrew Hanushevsky. 2005. XROOTD-A Highly scalable architecture for data access. *WSEAS Transactions on Computers* 1, 4.3 (2005).

[11] Dave Dykstra and Lee Lueking. 2010. Greatly improved cache update times for conditions data with Frontier/Squid. In *Journal of Physics: Conference Series*, Vol. 219. IOP Publishing, 072034.

[12] Fastly. 2019. Fastly: The edge cloud platform behind the best of the web. https://www.fastly.com/

[13] The Linux Foundation. 2019. Production-Grade Container Orchestration. https://kubernetes.io/

[14] Robert Gardner, Simone Campana, Guenter Duckeck, Johannes Elmsheuser, Andrew Hanushevsky, Friedrich G Hönig, Jan Iven, Federica Legger, Ilija Vukotic, Wei Yang, et al. 2014. Data federation strategies for ATLAS using XRootD. In *Journal of Physics: Conference Series*, Vol. 513. IOP Publishing, 042049.

[15] Gabriele Garzoglio, Tanya Levshina, Mats Rynge, Chander Sehgal, and Marko Slyz. 2012. Supporting shared resource usage for a diverse user community: the OSG experience and lessons learned. In *Journal of Physics: Conference Series*, Vol. 396. IOP Publishing, 032046.

[16] Balachander Krishnamurthy, Craig Wills, and Yin Zhang. 2001. On the use and performance of content distribution networks. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. ACM, 169–182.

[17] Microsoft. 2019. Large file download optimization with Azure CDN. https://docs.microsoft.com/en-us/azure/cdn/cdn-large-file-optimization

[18] Erik Nygren, Ramesh K Sitaraman, and Jennifer Sun. 2010. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review* 44, 3 (2010), 2–19.

[19] Ruth Pordes, Don Petravick, Bill Kramer, Doug Olson, Miron Livny, Alain Roy, Paul Avery, Kent Blackburn, Torre Wenaus, Frank Würthwein, et al. 2007. The open science grid. In *Journal of Physics: Conference Series*, Vol. 78. IOP Publishing, 012057.

[20] RPM. 2019. RPM Package Manager. https://rpm.org/contribute.html

[21] Derek Weitzel. 2019. djw8605/Pearc19-StashCache-Tools: v1.0 Release of tools. https://doi.org/10.5281/zenodo.2635588

[22] Derek Weitzel, Brian Bockelman, Duncan A Brown, Peter Couvares, Frank Würthwein, and Edgar Fajardo Hernandez. 2017. Data Access for LIGO on the OSG. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*. ACM, 24.

[23] Derek Weitzel, Brian Lin, Tony Aburaad, caseylargent, Robert Illingworth, Suchandra Thapa, Brian Bockelman, Rob Gardner, Marian Zvada, Lincoln Bryant, and eharstad. 2017. opensciencegrid/StashCache: Multi-Origin Support. https://doi.org/10.5281/zenodo.557111