Harmonizing Sequential and Random Access to Datasets in Organizationally Distributed Environments

Michał Wrzeszcz[1,2], Łukasz Opioła[1,2], Bartosz Kryza[2], Łukasz Dutka[2], Renata G. Słota[1], and Jacek Kitowski[1,2]

[1] AGH University of Science and Technology, Faculty of Computer Science, Electronics and Telecommunications, Dep. of Computer Science, Kraków, Poland
{wrzeszcz,lukasz.opiola,bkryza,renata.slota,jacek.kitowski}@agh.edu.pl
[2] Academic Computer Centre CYFRONET AGH, Kraków, Poland
lukasz.dutka@cyfronet.pl

Abstract Computational science is rapidly developing, which pushes the boundaries in data management concerning the size and structure of datasets, data processing patterns, geographical distribution of data and performance expectations. In this paper we present a solution for harmonizing data access performance, i.e. finding a compromise between local and remote read/write efficiency that would fit those evolving requirements. It is based on variable-size logical data-chunks (in contrast to fixed-size blocks), direct storage access and several mechanisms improving remote data access performance. The solution is implemented in the Onedata system and suited to its multi-layer architecture, supporting organizationally distributed environments – with limited trust between data providers. The solution is benchmarked and compared to XRootD + XCache, which offers similar functionalities. The results show that the performance of both systems is comparable, although overheads in local data access are visibly lower in Onedata.

Keywords: random access, variable-size block, distributed file system, organizationally distributed environment

1 Introduction

Recent years have brought significant advances in computational science and rapid development of data centers, which keep growing and employing modern, distributed storage technologies. Big institutions are getting dedicated network links and the throughput of network infrastructures is increasing. This technological progress aligns well with the trends in computational science that push towards globalization and distributed computing. The idea of e-Science [11] allows scientists from different fields and organizations to cooperate without borders, performing parallel, distributed analysis on large, shared datasets. However, current data access and sharing solutions can only partly fulfill this vision. The reason behind the lack of suitable solutions is the challenging nature of

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_22

data access globalization. Some relevant issues are: autonomy of data providers, geographical distribution of vast datasets, complicated maintenance of network-based communication, data security and privacy or decentralized authorization. Among them there is efficient and cost-effective access and processing of distributed datasets in such decentralized environments.

Analysis of different use-cases show that scientists use very diverse methods to process their datasets. Quite often the data is stored in giant (sometimes sparse) files, which are read or written by variable-size chunks in a seemingly arbitrary order – consider for example the popular HDF5 [9] format that can hold multi-dimensional data. While sequential access is usually well handled, the case of random read & write is a pitfall for most of network-based file systems. When data is located on remote storage systems or distributed, these operations trigger transfers of whole files or large blocks between storage clusters that can generate unnecessary costs. Transfer management and optimization can be very challenging, especially when files are accessed in a random manner.

In this paper we present our solution for harmonizing performance of sequential and random access to local and remote datasets in organizationally distributed environments. The solution was implemented as a part of data access system called Onedata [16, 17], evaluated and compared to commonly used XRootD virtual filesystem.

2 Related Work

Below is a summary of existing solutions related to efficient access to large, distributed datasets: distributed data access systems, solutions optimizing random access performance of network-based storage and tools for large data transfers.

The need of unified data access is apparent as more and more initiatives [22, 25] and products appear, trying to fulfill those requirements. For example, IBM offers Active File Management (AFM) [12] as an additional layer over their GPFS storage to achieve caching of data originating from remote sites (home-and-cache model) with support for data modifications. By creating associations between data clusters, one can implement a single namespace view across sites around the world, though this requires full trust between them. XRootD [3] is a commonly used, open-source alternative to GPFS + AFM, which embraces a very similar model when coupled with XCache [7]. XRootD can be used to unify access to different storage systems into a single virtual endpoint, accessible from anywhere. XCache is essentially an XRootD service employing a caching plugin, which manages a local cache of data read from remote sites for faster consecutive reading. Like in IBM's solutions, XRootD/XCache requires all the sites to be federated. However, in contrast to AFM, XCache does not support remote write operations. DataNet Federation Consortium (DFC) [5] aims to implement a national data management infrastructure to streamline scientific development. The prototype supports three types of federation mechanisms: (1) tightly coupled federations, realized by federating iRODS data grids, (2) loosely coupled federations, i.e. external services offering certain datasets for retrieval

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_22

and querying and (3) asynchronous federations where queries to external services are processed in an asynchronous (message queue based) manner. This approach promotes (read-only) integration with open data services, rather than unifying data access to distributed data providers. DFC employs iRODS [18] to create a single federation, but it can be also used to achieve cross-federation data access. However, it does not implement location transparency of the stored data. The files must be manually moved/copied between iRODS Zones. It requires certain administrative effort to set up cross-federation data access – user accounts pointing to their home Zone must be created in remote Zones.

As mentioned before, random access performance is a weakness of most file systems, and especially problematic in network-based storage systems. There have been attempts to overcome these limitations or introduce optimization mechanisms. For example, in [29] the authors propose three methods to optimize random queries on HDFS [20] and guarantee the performance of sequential access. All the methods are based on network-level optimizations and yield satisfactory results. Another attempt to adjust HDFS to random access profile is presented in [15], where the authors introduce some low-level modifications to make the filesystem better suited for computations in the field of High Energy Physics (HEP). The next example is VarFS [10] – a filesystem build on Ceph [24] especially for the purpose of random write operations. The general idea is that instead of using objects or blocks of fixed-size as most file systems do, this layer uses variable-size objects while remaining POSIX compatible. This way, random write performance can be greatly increased and the overheads of sequential write operations are acceptable. The conclusion is that it is possible to achieve reasonable random access performance in a commonly used distributed data access systems such as HDFS or Ceph, however these solutions are designed for federated environments.

Data transfers are an inherent aspect of distributed data access systems. Whenever a file is accessed remotely, it must be pushed through the network between data sites. In the great majority of data access systems, fixed-size block is used as the basic unit of data and only the required blocks are transferred. The stability and latency of the network link have a significant impact on efficiency – hence various optimization techniques are employed. They include prefetching of blocks, caching the data locally, tuning the network etc. Pre-staging used to be a reasonable choice in some scenarios, but with the growth of data volumes, replication of whole datasets is becoming unviable. Still, there are many cases where large files are moved between data clusters as part of the scientific process. The choice of tools for managing data transfers is wide, see products from Signiant [21], Axway [2], IBM [13] or Serv-u [19] as commercial examples. Non-commercial solutions include mdtmFTP [28], FDT [8] or GridFTP [1], which is extensively used in scientific communities. A common approach is to embrace parallel network links between clusters to speed up file moving – this idea was formalized in the Parallel FTP protocol [4].

The choice of tools for data access and transfers is wide, but there is a lack of integrated solutions for efficient data access in organizationally distributed

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_22

environments. Such solution should hide away the complexity of manual data management between autonomous sites and offer a unified, transparent view on all user's datasets, at the same time ensuring the performance sufficient for scientific computing.

3 Data Access in Organizationally Distributed Environments

Our solution for harmonizing performance of sequential and random access to local and remote datasets is a part of Onedata – an eventually consistent distributed data access system. Onedata aims to provide access to distributed data under a single namespace [27]. Its main goal is to achieve truly transparent, efficient, scalable and cost-effective data access to autonomous data providers, despite the inherent lack of trust between them [17].

The Onedata system is based on a multi-layer architecture (see Figure 1). Onezones provide an Authentication and Authorization Infrastructure, and mediate in cooperation of Oneproviders, which realize access to datasets stored in different organizations. Oneclients, subject to Oneproviders, employ FUSE (filesystem in userspace [23]) to implement POSIX data access and seamless integration with filesystems. While Onezones are key to overcome the lack of trust in organizationally distributed environments, Oneproviders and Oneclients are responsible for handling data access.

Users access their data through the Oneclient software using logical paths pointing at logical files. To provide efficient data access without significant overheads, Oneclient accesses the data directly on the storage system whenever possible. Otherwise, proxy mode is used – the data is read/written through a network connection to Oneprovider. Prior to direct data access, Oneclient gains knowledge about logical files from metadata managed by Oneprovider. The metadata includes such information as logical filenames, permissions, access types and a registry of data-chunks – as decribed below.

3.1 Data-chunks

Onedata introduces data-chunk as the basic unit of data. The content of each logical file consists of one or more data-chunks, each representing a range of bytes. Data-chunks have similar role as blocks in a standard filesystem, although they can correspond to a series of blocks or other entities (e.g., objects) on the underlying storage system. Thus, data-chunk handling is a vital factor in data access scalability, performance and cost-effectiveness. Without appropriate models for metadata consistency and synchronization [26], file metadata that includes the registry of data-chunks can become a bottleneck of the whole data access system (e.g. [6, 14, 24]).

Datasets differ in characteristics – in extreme cases the files may be small and numerous or large and sparse. While small data-chunk size would result in creation of numerous data-chunks for big files, large data-chunk size would

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_22

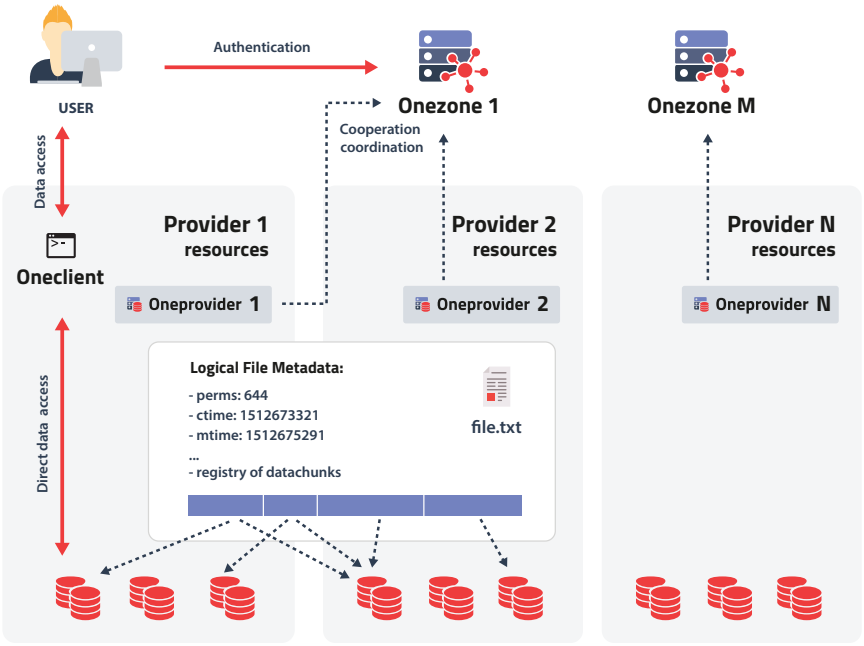

Figure 1. Multi-layer Onedata architecture and logical file matadata with data-chunks.

cause synchronization of large data pieces even when a single byte is read. Thus, Onedata uses variable-size data-chunks – in specific cases, a data-chunk can represent a single byte or the whole file.

Oneprovider services synchronize file metadata including the registry of data-chunks (see Figure 1). When a data-chunk is overwritten by a Oneprovider, the remaining Oneproviders mark the modified data-chunk as invalid. As the actual file content is not exchanged, this is a lightweight mechanism even for large files. Data transfers are performed only when a data-chunk being read is absent from the Oneprovider that handles the reading. As needed, data-chunks are split on the fly to limit the transfer size to the missing data range only. Therefore, the variable-size data-chunks minimize the cost of remote data access.

All things considered, there are several advantages of variable-size block management in the context of highly distributed systems, which align well with our concept of data-chunks:

- universal fit for small and large files,
- limiting the network and storage cost to possible minimum,
- flexibility and ability to dynamically adapt to circumstances,
- seamless integration with different underlying storage systems, no matter their blocksize or type (file/object-based etc.).

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_22

3.2 Models for metadata consistency and synchronization

Metadata access overheads can be related to round trip times, which are often impossible to reduce. For this reason, the metadata is replicated between Oneproviders and cached by Oneclients. Whenever possible, it is processed locally and/or asynchronously. As a result, most of the metadata (including the registry of data-chunks) is eventually consistent and adopts last-write-wins conflict resolution. The causal consistency model is applied only to metadata managed by Onezones that is crucial for cooperation and security.

Since the overheads of synchronization grows with the number of entities that exchange metadata, only Oneproviders that store parts of the particular dataset are involved in processing and replication of the corresponding metadata.

3.3 Data access performance

One of the basic assumptions for Onedata is direct access to storage systems whenever possible to retain the performance they offer. Scalability is achieved by handling multiple underlying storage systems in parallel and limiting the metadata processing overheads as much as possible, by using appropriate consistency and synchronization models (see 3.2).

Data access performance can be further improved by employing specialized mechanisms that support particular data access patterns when data is not accessible directly. However, the sequential and random data access patterns require different optimization strategies to limit the negative impact of the network and data access latency.

4 Harmonizing Random and Sequential Access

There are three main factors when considering efficient data access:

- operation: read/write,
- data location: local/remote,
- access pattern: sequential/random.

The write operation on the side of Oneclient works in the same way, regardless if local or remote. The data is written directly to the local storage system and events are produced that update the data-chunk registry asynchronously. This process is depicted in Figure 2 – initially, the file is stored only in the second Oneprovider. Oneclient overwrites a part of the file content within the first Oneprovider, a registry update is broadcasted and the data-chunk is invalidated in the second Oneprovider.

Similarly to the write operation, local data read is performed directly on the storage system. In both cases, the only overheads are caused by fetching the file metadata, which stays cached for faster consecutive operations. Essentially, the efficiency of write and local read operations depends roughly on the local storage system performance. Thus, to harmonize the performance of data access, we focused mostly on mechanisms that support sequential or random data reading from remote sites, discussing local operations for reference only.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_22

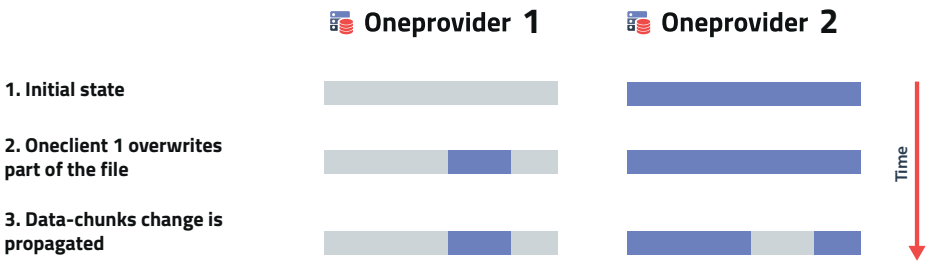

Figure 2. Remote write causing a data-chunk invalidation.

4.1 Sequential Remote Read

During sequential remote reading, Oneclient requests its Oneprovider to transfer the missing data-chunks to the local storage as needed and reads the file block by block. While certain data-chunks of the file may be distributed between many remote Oneproviders, the key factor of sequential remote read performance is reduction of the delay in access to non-local data-chunks. It is achieved using a prefetching mechanism. For each opened file handle, Oneclient continuously detects the access pattern (sequential vs. random), based on comparing the read offsets on consecutive read operations. When a file is detected to be accessed sequentially, Oneclient requests transfer from remote Oneprovider of more data in advance to be immediately accessible as reading proceeds. Data transfers are prioritized so that the prefetch requests do not hinder the transfers of data needed instantly. Moreover, all Oneclients operating on a particular logical file are asynchronously notified of any prefetched data-chunks in order to minimize the number of transfer requests. In summary, Onedata employs three mechanisms that support sequential remote read: prefetching, prioritization of transfer requests and broadcasting of information about synchronized data blocks.

4.2 Random Remote Read

Random read performance is cumbersome to optimize in any file system, especially when data is stored in a remote location. The introduction of data-chunks greatly limits the network traffic caused by transfers, but reading a file remotely (especially by small blocks) results in creation of numerous small data-chunks. This causes the data-chunks registry to grow, increasing the costs of processing and synchronization between Oneproviders. Moreover, the prefetching mechanism in Oneclient is undesirable during random read, as it hampers the performance by transferring unneeded bytes. For these reasons, we introduced several optimizations to data-chunk management and Oneclient behaviour to suite them to remote data access.

The data-chunks registry includes information about data-chunks stored in local and remote Oneproviders. The registry is updated whenever one of the

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_22

following takes place: an event is received from Oneclient reporting data modification, an update of certain data-chunks appears from another Oneprovider or a transfer request is completed. As random read can result in thousands of small transfer requests per second, the overheads of synchronizing the data-chunks registry with other Oneproviders become considerable. For this reason, we introduced the distinction of public (instantly advertised) and private (stored only locally) data-chunks. Public data-chunks are created as a result of data modifications, so that other Oneproviders are quickly informed about any changes in the file content. Private data-chunks are a result of replicating fragments of data to the local storage. There is no need to broadcast them quickly – it is done only after they are merged to a larger data-chunk and made public.

To minimize the costs of processing and synchronizing the data-chunks registry, it is based on a tree structure with fast offset-based access. Consequently, the registry processing time grows logarithmically with size, and upon any modification, only the changed parts of the tree are broadcasted to other Oneproviders.

In case Oneclient determines that the file is not read in sequential access pattern, it assumes that the file is accessed randomly and the prefetching algorithm works differently. Rather than requesting consecutive parts of data, it discovers which fragments (if any) of the logical file are accessed frequently and prefetches them (see Figure 3). Such behavior is beneficial for two reasons. Firstly, it is probable that further read operations will appear within such fragment and will be handled much faster. Secondly, such aggregation merges several data-chunks into a larger public one, decreasing the overall data-chunks number and triggering a broadcast of the aggregated data-chunk.

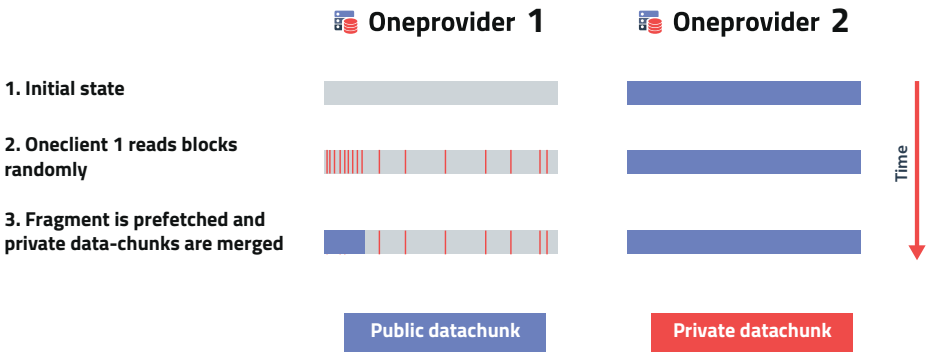

Figure 3. During random read, private data-chunks are merged into a bigger one, prefetched and published.

To summarize, the following mechanisms support random remote read in Onedata: private data-chunks, data-chunk merging, selective tree-based processing and broadcasting of the data-chunk registry, automatic discovery of ran-

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_22

dom access pattern and frequently accessed fragments of logical files that trigger prefetching of the whole fragment.

4.3 Influence of Random and Sequential Read on One Another

Due to their nature, sequential and random read require different optimizations. Some of the mechanisms dedicated for one read type have an opposite effect on the other. For this reason, Oneclient performs continuous detection of data access pattern and adjusts its behaviour accordingly.

Besides access pattern recognition, Onedata harmonizes random and sequential access when many Oneclients operate on logical files sequentially and randomly in parallel. In such case, randomly reading Oneclients trigger merging of smaller data-chunks into larger ones, which makes certain fragments of files better suited to sequential read. On the other hand, sequentially reading Oneclients trigger prefetching, which does not block random read (due to lower priority) but increases the chances of hitting already prefetched data during further reading.

5 Evaluation

We have performed benchmarks to verify the read and write performance and estimate overheads introduced by the Onedata software. For reference, an installation of XRootD and XCache with standard settings has been tested using the same underlying storage and benchmarks. The environment consisted of two identical virtual machines: 12 CPU x 2GHz and 40GB RAM. All test cases were based on a 64KB block and the test file size was 200GB. The network link between the machines yielded about 5.2 Gb/s. The presented results have been obtained from several runs with repeatable measurements.

5.1 Local Data Access

The purpose of testing local data access was mainly to estimate the overheads of virtualization. Data was read or written to a test file by one process on one host in three cases:

- directly on the storage system,
- via Oneclient connected to the Oneprovider on the host and with direct access to the storage,
- via XRootDFS (FUSE-based client for XRootD) connected to the XRootD server on the host.

Results of the tests are presented in Figures 4 and 5. Thanks to the fact that Oneclient operates directly on the storage system and communicates with Oneprovider only to fetch the required metadata, the measurements are close to the underlying file system performance. XRootDFS uses a network link to the XRootD server to read/write file data and yields lower results, despite the fact that both client and server were located on the same machine. As a consequence, when scalability is concerned, XRootDFS depends on the network capacity, and Oneclient depends on the underlying storage scalability.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_22

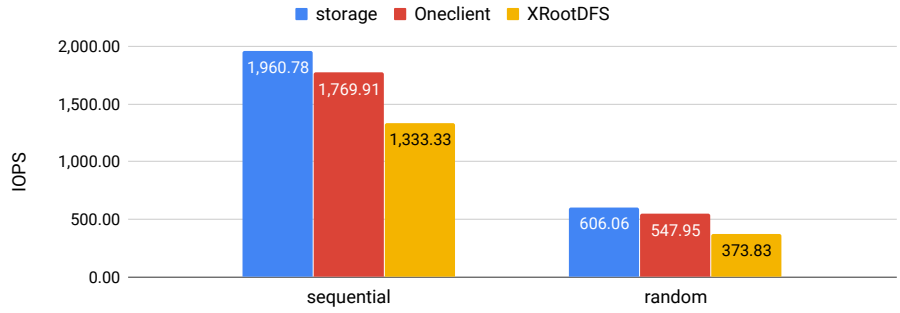

Figure 4. Local read performance.

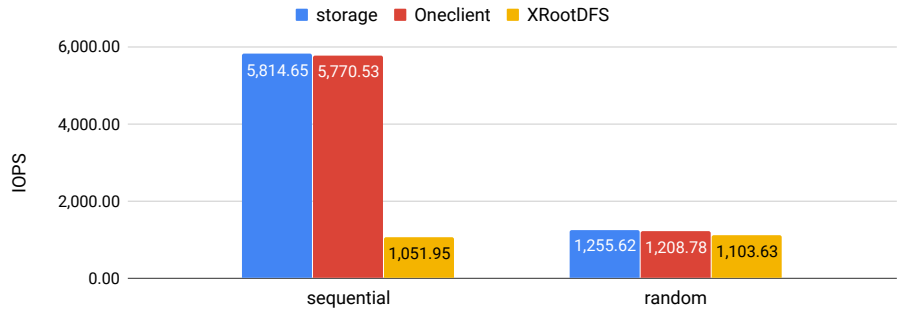

Figure 5. Local write performance.

5.2 Remote Data Access

The environment for remote data access tests consisted of two machines, hosting one of the following setups:

- Oneprovider on the first host, Oneprovider + Oneclient (with direct storage access) on the second host,
- XRootD server on the first host, XCache + XRootDFS on the second host.

The tests included only remote read benchmarking. Remote write was not tested, because it is not supported by XCache, and in Oneclient it works the same way as local write and yields the same performance – the data is written locally and overwritten data-chunks are invalidated in remote providers.

The test file was placed on the first host, and read by the client software on the second host, via the caching layer (second Oneprovider / XCache). Effectively, reading the file caused data trasfers between the Oneproviders or XRootD and XCache. The file was read following three different patterns: sequential, random and hybrid (starting from a random offset every time, a 20MB fragment was read). The results are shown in Figure 6.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_22

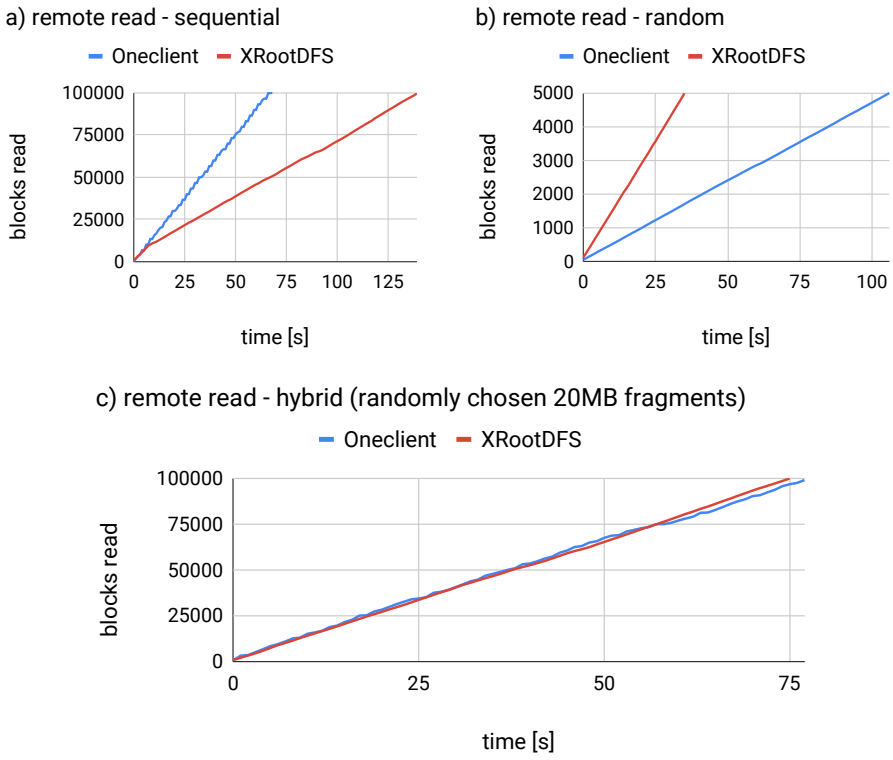

Figure 6. Remote read performance – (a) sequential, (b) random, (c) hybrid.

Figure 6a shows that the prefetching mechanism in Oneclient works effectively. The chart adopts a stairstep-like shape, depicting where the prefetching or reading proceeded ahead of each other – the data is transferred by Oneprovider to the local storage and then read directly by Oneclient. XCache was configured with prefetching enabled and served the file in a slower, but stable manner.

Random remote read (Figure 6b) is the most pessimistic case for any filesystem, as no prediction-based optimizations can be done when the access is completely random. This is where XCache is faster, thanks to its simpler architecture. The data is fetched from remote XRootD, served to the client and cached locally at the same time. On the other hand, Oneprovider transfers the data and writes it to the storage before informing the client that it is ready to be read. Nevertheless, consecutive reading of the same blocks (if required) would be faster via Oneclient (as shown in local read tests), and the total transfer size is about 15 times less for Oneclient thanks to variable-size data-chunks (64KB vs. 1MB default block size in XRootD). In this case, prefetching in XCache was disabled – it is worth mentioning that while Oneclient detects the reading pattern automatically, XCache needs to be restarted when prefetching settings change, making it

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_22

less universally applicable. Moreover, the case of complete random read is quite rare, usually there is a pattern that causes some file fragments to be read more frequently than others.

Considering the above-mentioned, we have tested the two solutions in a more likely scenario when the file is read by bigger fragments (20MB), but randomly chosen every time (Figure 6c). Here, Oneclient and XRootDFS yielded similar measurements – a result of a compromise between prefetching and random read performance.

5.3 Discussion

The presented tests were performed on an elementary environment, where both systems were running on default settings and without any tuning. The purpose was to assess the impact of using variable-size data-chunks in comparison to a fixed-size block approach in XRootD, as well as the mechanisms introduced in Onedata in order to harmonize sequential and random data access. That said, these benchmarks should not be perceived as an absolute comparison of Onedata and XRootD + XCache performance. The results show that our data-chunk based solution achieves performance comparable to a state-of-the-art virtual filesystem based on a fixed block size, while offering additional features such as remote write and ensuring lower overheads in local data access. These tests should be treated as a proof-of-concept and their satisfying results are our incentive to further refine and optimize the proposal. We plan to perform tests in larger scale, on more complex environments, taking into account various parameters and other state-of-the-art filesystems for reference.

6 Conclusions and Future Work

In this paper we present our approach for harmonized data access in organizationally distributed environments. The solution has been implemented in a data access system called Onedata. The purpose is to provide a universal solution for data access that offers a satisfactory compromise between sequential and random read/write performance. It is achieved by combining a multi-layer architecture with Onezones with a novel approach to logical mapping of distributed file content – variable-size data-chunks, a layer over physical file blocks, objects etc. The system achieves good performance owing to support for direct storage access, which minimizes overheads during local data read and write, and a series of optimizations and mechanisms for efficient data-chunk management that boost remote data access: prefetching, prioritization of data transfers, event-based notifications of data-chunk registry changes, reading pattern recognition, detection of popular file fragments and data-chunks merging.

The universal character of Onedata means that it will not perform better than solutions dedicated for certain use-cases. Nevertheless, scientific computing is constantly developing, along with the diversity of datasets and data access patterns, which promotes integrated solutions that can cover various use-cases

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_22

with sufficient performance. Furthermore, the sizes of datasets are ever-growing, which gradually makes pre-staging and full data replication obsolete, and encourages data access based on smaller file fragments. The proposed data-chunks are well suited to those needs, and have the great advantage of minimizing network and storage costs. The performance is comparable to the state-of-the-art XRootD virtual filesystem, coupled with XCache, at the same time the Onedata system offers more features, such as support for organizationally distributed environments and remote write, which are desired in scientific collaboration.

The future work includes further tests in larger scale and comparing more filesystems, following further optimizations: lowering the overheads of data transfer management, better access pattern recognition and increasing adaptability of the prefetching mechanisms to minimize the idle time during sequential reading.

Acknowledgments

This work has been partially supported by the funds of Polish Ministry of Science and Higher Education assigned to AGH University of Science and Technology and 2018-2020's research funds in the scope of the co-financed international projects framework (projects no. 3958/H2020/2018/2 and no. 3905/H2020/2018/2).

References

1. Allcock, W., Bester, J., Bresnahan, J., et al.: GridFTP: Protocol Extensions to FTP for the Grid. Global Grid Forum, GFD-RP **20**, 1–21 (2003)
2. Axway AMPLIFY, https://www.axway.com/en/products/amplify
3. Bauerdick, L., Benjamin, D., Bloom, K., Bockelman, B., Bradley, D., Dasu, S., Ernst, M., Gardner, R., Hanushevsky, A., Ito, H., et al.: Using Xrootd to Federate Regional Storage. In: Journal of Physics: Conference Series. vol. 396 (4), p. 042009. IOP Publishing (2012)
4. Bhardwaj, D., Kumar, R.: A Parallel File Transfer Protocol for Clusters and Grid Systems. In: e-Science and Grid Computing, 2005. First International Conference on. pp. 7–254. IEEE (2005)
5. DataNet Federation Consortium, http://datafed.org/
6. Dong, D., Herbert, J.: FSaaS: File System as a Service. In: IEEE 38th Annual Computer Software and Applications Conference (2009)
7. Fajardo, E., Tadel, A., Tadel, M., Steer, B., Martin, T., Würthwein, F.: A Federated Xrootd Cache. In: Journal of Physics: Conference Series. vol. 1085, p. 032025. IOP Publishing (09 2018)
8. Fast Data Transfer, http://monalisa.cern.ch/FDT/
9. Folk, M., Heber, G., Koziol, Q., Pourmal, E., Robinson, D.: An Overview of the HDF5 Technology Suite and Its Applications. In: Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases. pp. 36–47. AD '11, ACM, New York, NY, USA (2011). https://doi.org/10.1145/1966895.1966900, http://doi.acm.org/10.1145/1966895.1966900
10. Gong, Y., Hu, C., Xu, Y., Wang, W.: A Distributed File System with Variable Sized Objects for Enhanced Random Writes. The Computer Journal **59**(10), 1536–1550 (2016)

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_22

11. Hinrich, P., Grosso, P., Monga, I.: Collaborative Research Using eScience Infrastructure and High Speed Networks. Future Generation Computer Systems **45**(C), 161 (2015)
12. IBM Active File Management, https://www.ibm.com/support/knowledgecenter/en/STXKQY_4.1.1/com.ibm.spectrum.scale.v4r11.adv.doc/bl1adv_afm.htm
13. IBM MFT, https://www.ibm.com/customer-engagement/supply-chain/managed-file-transfer
14. Leong, D.: A new revolution in enterprise storage architecture. IEEE Potentials **28**(6), 32–33 (2009)
15. Li, Q., Sun, Z., Wei, Z., Sun, G.: A New Data Access Mechanism for HDFS. In: Journal of Physics: Conference Series. vol. 898, p. 062018. IOP Publishing (10 2017)
16. Onedata, https://onedata.org
17. Opioła, Ł., Dutka, Ł., Słota, R.G., Kitowski, J.: Trust-driven, Decentralized Data Access Control for Open Network of Autonomous Data Providers. In: 16th Annual Conference on Privacy, Security and Trust (PST). pp. 1–10. IEEE (Aug 2018). https://doi.org/10.1109/PST.2018.8514209
18. Röblitz, T.: Towards Implementing Virtual Data Infrastructures – a Case Study with iRODS. Computer Science (AGH) **13**(4), 21–34 (2012), http://dblp.uni-trier.de/db/journals/aghcs/aghcs13.html#Roblitz12
19. Serv-u, https://www.serv-u.com
20. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The Hadoop Distributed File System. In: Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on. pp. 1–10. Ieee (2010)
21. Signiant, https://www.signiant.com
22. Słota, R., Dutka, Ł., Wrzeszcz, M., Kryza, B., Nikolow, D., Król, D., Kitowski, J.: Storage Management Systems for Organizationally Distributed Environments PLGrid PLUS Case Study. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) Parallel Processing and Applied Mathematics. pp. 724–733. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
23. Szeredi, M.: Fuse: Filesystem in Userspace. http://fuse. sourceforge. net (2010)
24. Weil, S.A., Brandt, S.A., Miller, E.L., Long, D.D., Maltzahn, C.: Ceph: a Scalable, High-performance Distributed File System. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation. pp. 307–320. USENIX Association (2006)
25. Wrzeszcz, M., Kitowski, J., Słota, R.: Towards Trasparent Data Access with Context Awareness. Computer Science **19**(2), 201–221 (2018)
26. Wrzeszcz, M., Nikolow, D., Lichoń, T., Słota, R., Dutka, Ł., Słota, R.G., Kitowski, J.: Consistency Models for Global Scalable Data Access Services. In: International Conference on Parallel Processing and Applied Mathematics. pp. 471–480. Springer (2017)
27. Wrzeszcz, M., Trzepla, K., Słota, R., Zemek, K., Lichoń, T., Opioła, Ł., Nikolow, D., Dutka, Ł., Słota, R., Kitowski, J.: Metadata Organization and Management for Globalization of Data Access with Onedata. In: International Conference on Parallel Processing and Applied Mathematics. pp. 312–321. Springer (2015)
28. Zhang, L., Wu, W., DeMar, P., Pouyoul, E.: mdtmFTP and its Evaluation on ESNET SDN Testbed. Future Generation Computer Systems **79**, 199–204 (2018)
29. Zhou, W., Han, J., Zhang, Z., Dai, J.: Dynamic Random Access for Hadoop Distributed File System. In: Distributed Computing Systems Workshops (ICDCSW), 32nd International Conference on. pp. 17–22. IEEE (2012)

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_22