BlobSeer: Efficient Data Management for Data-Intensive Applications Distributed at Large-Scale

Bogdan Nicolae, Gabriel Antoniu, Luc Bougé

BlobSeer: Efficient Data Management for Data-Intensive Applications Distributed at Large-Scale

Bogdan Nicolae
University of Rennes 1
IRISA
Rennes, France
bogdan.nicolae@irisa.fr

Advisor: Gabriel Antoniu
INRIA Rennes
IRISA
Rennes, France
gabriel.antoniu@inria.fr

Advisor: Luc Bougé
ENS Cachan, Brittany
IRISA
Rennes, France
luc.bouge@bretagne.ens-cachan.fr

I. Introduction

A. Problem description

As the rate, scale and variety of data increases in complexity, the need for flexible applications that can crunch huge amounts of heterogeneous data fast and cost-effective is of utmost importance.

Such applications are *data-intensive*: in a typical scenario, they continuously acquire massive datasets (e.g. by crawling the web or analyzing access logs) while performing computations over these changing datasets (e.g. building up-to-date search indexes). In order to achieve scalability and performance, data acquisitions and computations need to be *distributed at large scale* in infrastructures comprising hundreds and thousands of machines.

As these applications focus on data rather then on computation, a heavy burden is put on the storage service employed to handle data management, because it must efficiently deal with massively parallel data accesses. In order to achieve this, a series of issues need to be address properly: *scalable aggregation of storage space* from the participating nodes with minimal overhead, the ability to store *huge data objects*, *efficient fine-grain access* to data subsets, *high throughput even under heavy access concurrency*, *versioning*, as well as *fault-tolerance* and a high *quality of service* for access throughput.

B. Our approach: BlobSeer

This PhD proposal introduces BlobSeer, an efficient distributed data management service that addresses the issues presented above. In BlobSeer, long sequences of bytes representing unstructured data are called *blobs* (*Binary Large OBject*).

1) Access interface: A client of BlobSeer manipulates a blob through a simple access interface that enables creating a blob, reading/writing a subsequence of *size* bytes from/to the blob starting at *offset* and appending a sequence of *size* bytes to the blob. This access interface is designed to support versioning explicitly: each time a write or append is performed by the client, a new snapshot of the blob is generated rather then overwriting any existing data (but physically stored is only the difference). This snapshot is labeled with an incremental version and the client is allowed to read from any past snapshot of the blob by specifying its version.

In a typical setting, a large number of clients access the same blob concurrently. With respect to this, BlobSeer provides strong consistency guarantees: all operations are *linearizable* [1].

2) Architecture: BlobSeer consists of a series of distributed communicating processes. Each blob is made up of fixed-sized *chunks* that are distributed among *data providers*. *Metadata* is used to map ranges defined by *(offset, size)* to the data providers where the corresponding chunks are stored, for any existing version of a blob. This metadata is stored and managed by *metadata providers* through a decentralized, DHT-based infrastructure. A central *version manager* is responsible of assigning versions to writes and appends and exposing these versions to reads in such way as to ensure consistency. Finally, a *provider manager* decides which chunks are stored on which data providers when writes or appends are issued by the clients.

3) Key design choices: Three key design factors enable BlobSeer to address these requirements presented in Section I-A: *data striping*, *distributed metadata management* and *versioning-based concurrency control*.

Data striping: Each blob is split into chunks of a fixed size which is specified at the time the blob is created. Typically the size of the chunks matches or is a multiple of the size of the data the client is expected to process in one step. These chunks are distributed all over the storage space providers of BlobSeer. A configurable chunk distribution strategy is employed when writes and appends are issued, in order to maximize the benefits of data distribution with respect to the desired goal (for example, round-robin can be used to achieve load-balancing). The data distribution strategy has a major role in sustaining a high throughput when concurrent clients access different parts of the blob.

Metadata decentralization: Since each blob is spread across a large number of storage space providers, BlobSeer needs to maintain metadata that maps subsequences of the blob to the corresponding chunks. In traditional approaches, this is

the job of a centralized metadata server that is responsible for metadata maintenance and consistency. In contrast, BlobSeer uses a distributed metadata management scheme for two main reasons: firstly it avoids the bottleneck of accessing the same centralized node for metadata queries under heavy access concurrency and secondly it avoids having the metadata server act as a single point of failure.

We organize metadata as a versioning-oriented, distributed segment-tree based structure. To favor efficient concurrent access to metadata, the tree nodes are distributed in a fine-grain manner among the metadata providers, which form a DHT (Distributed Hash Table). This metadata organization scheme has a significant impact on the global throughput, as demonstrated in [2]. A detailed description of the algorithms used to manage metadata can be found in [3].

Versioning-based concurrency control: Versioning is not only provided at client level through the access interface, but is also leveraged by the concurrency control to increase the number of operations performed in parallel by avoiding synchronization as much as possible. The key idea is simple: thanks to versioning, *no existing data or metadata is ever overwritten*: new data chunks and metadata tree nodes are always added and never modified. This enables the readers to be completely decoupled from the writers: concurrent readers and writers will never interfere with each other because writers never modify an existing blob snapshot, not at data nor at metadata level (read/write concurrency). Thus from the reader point of view the blob snapshot is at all times in a consistent state and no synchronization is necessary, despite the concurrent writers.

Because data is never overwritten and only the difference with respect to the previous version is stored, writers/appenders also can send their chunks to the storage space providers independently of each other (write/write concurrency). It is at metadata level where the synchronization takes place and the newly written chunks are integrated into a new version. The detailed algorithms are available in [3].

II. Significance of the research

This research resulted in the BlobSeer prototype, an efficient and scalable large-scale storage service that serves the needs of data-intensive applications. As the storage service is a key factor that impacts overall performance of data-intensive applications, understanding the issues involved and coming up with a design that overcomes these issues is thus crucial. This work not only provides efficient support for features commonly used to exploit parallelism at data level, but also explores a set of new features that can be leveraged to further improve parallel data access.

Processing massive amounts of unstructured data (such as web pages, online transaction records, access logs, etc.) in huge datacenters is very common nowadays. Several paradigms such as MapReduce [4] and Dryad [5] have been proposed to address this need. These paradigms facilitate exploiting parallelism at data level *explicitly*. The advantage of this approach is the fact that once the application is cast into the framework, the scheduling, distribution and data transfers are performed automatically. This greatly simplifies application design, but favors massively parallel data access that has to be handled efficiently by the underlying storage service. In this context, results from this work are highly relevant.

Global file systems are an attractive choice for the high-performance computing community, as they federate large-scale distributed storage resources while offering transparent access to data through a shared file namespace. Transparency at large scale however raises a lot of challenges with respect to scalability and performance as issues such as data placement, caching, replication, concurrent accesses and access to historic data need to be addressed. This work brings insight into many of these issues and proposes efficient solutions to cope with them. Results from this work have already been successfully applied to the Gfarm grid file system [6].

Data-intensive applications form an important class of applications for popular execution infrastructures, such as desktop grids, clouds and service-based architectures. Desktop grids for example are highly popular in the e-Science community, which needs access to large datasets, wide-area transfers and broad distribution of Terabytes of data. Clouds and service-based architectures are highly popular in the industry, where cost-effective processing of data at large scale under quality of service guarantees is an issue. In all these scenarios, scalable and performant data management is a key factor for the success of the whole underlying infrastructure.

III. Related work

Work with respect to large-scale distributed storage solutions has been carried out in the area of parallel and distributed file systems [7]. Traditional approaches [8], [9] target to emulate as closely as possible the behavior of a general purpose POSIX file system running on a single machine. This approach however limits the possibility of exploiting data parallelism explicitly as it forces concurrent accesses to the same files to obey POSIX semantics.

Specialized file systems have been introduced that target the needs of data-intensive oriented paradigms that exploit data parallelism explicitly. Google developed GoogleFS [10], a distributed file system that meets the needs of their MapReduce paradigm. Hadoop, an open-source MapReduce implementation relies on a specialized storage layer as well: HDFS [11]. Compared to these systems, BlobSeer efficiently supports concurrent writes at arbitrary positions in the same blob, a feature that is either poorly supported in GoogleFS or not supported at all in HDFS. Moreover, BlobSeer introduces several novel techniques, among which metadata decentralization and versioning based concurrency-control to push the limits of exploiting parallelism at data-level even further.

Several explicit transfer protocols have been established, such as the GridFTP [12] extensions to the File Transfer Protocol. These transfer protocols define a general purpose mechanism for secure, reliable and high-performance data movement. While this approach enables efficient fine-grain

data transfers at large scale, unlike in the case of BlobSeer, data management transparency is missing, as the user must be aware of data location and manage transfers explicitly.

With the emergence of cloud computing, storage solutions specifically designed to fit in this context appeared. Amazon S3 is such a storage service that provides a simple web service interface to write, read, and delete an unlimited number of objects of modest sizes (at most 5 Gigabytes). Its simple design features high scalability, reliability and access speed. BlobSeer on the other hand introduces a set of advanced features, such as support for huge objects, fine-grain access, striping, concurrent writes to the same object. These features form a solid basis for advanced storage services on clouds.

IV. Preliminary results

A. Transparent fine-grain access to massive data

Our initial efforts concentrated on enabling efficient data sharing at global scale in grids. As managing huge amounts of data explicitly at a very large scale makes the design of applications much more complex, one key issue we addressed is *transparency* with respect to data localization and data movements. In this context, one important goal is to provide mechanisms allowing to manage massive files (e.g., of several Terabytes), while providing efficient fine-grain access to small parts of the file. The initial BlobSeer prototype, featuring RAM-based storage only while leveraging a simplified segment-tree based metadata management scheme was developed for this purpose.

Preliminary experiments performed on Grid'5000 [13] testbed demonstrated this approach to scale well, both in terms of metadata overhead and in terms of concurrent reads and writes. The results have been published in [14].

As a next step, we extended this work to show how our approach to transparent fine-grain access to massive data applies to real-life grid applications. We considered an application in the field of astronomy, supernovae detection. In this context, huge data strings representing the view of the sky are shared and accessed by concurrent clients in a fine-grain manner in an attempt to find supernovae in parts of the sky. We targeted efficient fine-grain access by eliminating the need to lock the string itself. Our results show good concurrent access performance and also underline the benefits of metadata caching on the client side. The results have been published in [15].

B. Efficient Versioning for Large Object Storage

We targeted large-scale data-intensive distributed applications built on top of paradigms that exploit data parallelism explicitly. In this context applications continuously acquire massive datasets (mostly writes) while performing computations over them (mostly reads). It is thus a crucial issue to minimize synchronization between data acquisition and data processing in order to proceed as much as possible in parallel. Versioning in this context is highly beneficial as it enables processing to be performed on a stable snapshot, while the data acquisition builds a newer snapshot in background.

For this purpose we further refined and formalized our versioning-oriented access interface for blob-based data management. We introduced concurrent append support by further developing our segment-tree-based distributed metadata management scheme to accommodate the appends efficiently, while keeping the same level of performance for reads and writes. We also introduced persistent data and metadata storage while keeping our initial RAM-based storage scheme as an underlying caching mechanism.

We conducted preliminary large-scale experimentation on the Grid'5000 testbed evaluating append performance. Results suggest a good scalability with respect to the data size and to the number of concurrent accesses. There results have been published in [3].

C. High Write Throughput in Desktop Grids

We evaluated BlobSeer in its role as a storage service for *write-intensive* applications running in Desktop Grids that have high *output* data requirements and where the access grain and the access pattern may be random.

In this context, the main challenge is to deal with *heavy write concurrency* (both writes and appends) in an efficient way. We addressed this challenge by leveraging data striping and our decentralized versioning-oriented metadata structure build on top of distributed segment trees.

We conducted extensive experimentation on the Grid'5000 testbed, evaluating both the impact of data decentralization and metadata decentralization. We insisted in a final large scale experiment on the importance of the latter on sustaining high write throughput when under heavy write concurrency. Results suggest clear benefits of using a decentralized metadata approach. They have been published in [2].

D. High Throughput under Heavy Concurrency for Hadoop Map/Reduce Applications

We targeted to provide high data access throughput even under heavy access concurrency for MapReduce data-intensive applications. For this purpose we chose Hadoop, an open-source software framework that implements the MapReduce paradigm, and replaced its default storage layer, Hadoop Distributed File System (HDFS), with a new storage layer based on BlobSeer.

To this end, we implemented a fully-fledged distributed file system on top of BlobSeer, BSFS, that manages a hierarchical directory structure, mapping files to blobs which are addressed in BlobSeer using a flat scheme. We also had to implement the streaming access API of Hadoop in BSFS which raised issues such as buffering and prefetching. Finally, since Hadoop tries to place the computation close to the data we had to extend BlobSeer to expose the data location and then integrate this into BSFS through a Hadoop-specific API.

Again, we conducted extensive experimentation on the Grid'5000 testbed. We used synthetic MapReduce access patterns that go further than our previous experiments both in scale and scope and we show clear benefits of using BlobSeer over Hadoop's original back-end, especially in the case of

concurrent accesses to the same huge file. Moreover, we performed experimentations with real MapReduce applications and demonstrated clear improvements when using BlobSeer as well. Our results have been recently accepted for publication [16].

E. Improving QoS in Large-scale Distributed Data Storage Services

To adequately support the requirements of distributed data-intensive applications on large-scale infrastructures, maintaining a high quality of service for the data storage system is crucial. Dealing with this issue involves reasoning about the behavior of the storage service, however this is an inherently difficult task given the large number of factors involved: highly-concurrent data access patterns, long periods of service uptime, failures of physical components, the highly distributed nature of the storage service itself.

We proposed an offline analysis approach to improve the quality of service in distributed storage systems based on global behavior modeling combined with client-side quality of service feedback. It automates the process of identifying dangerous behavior patterns in storage services, which makes reasoning about potential improvements much easier. With this occasion, BlobSeer saw important improvements with respect to fault tolerance: we added configurable per-blob data replication capabilities.

We demonstrated our approach by using GloBeM [17] (a global behavior modeling technique based on monitoring data analysis and machine learning) to improve the quality of service in BlobSeer. Extensive experimentations under hard conditions were conducted on the Grid'5000: highly-concurrent data access patterns for long periods of service up-time while supporting failures of the physical storage components. Our results show a substantial improvement in quality of service by sustaining a higher and more stable data access throughput. They have been recently submitted for publication.

V. Conclusions

We have successfully demonstrated the benefits of using several key design choices: data striping, distributed metadata management and versioning-based concurrency control in a storage service to enable efficient data management for data-intensive applications at large scale. To this end we performed extensive experimentations at large scale using our fully-functional BlobSeer prototype. Results obtained confirm substantial gain in data access throughput and quality of service for a wide range of data-intensive scenarios.

As this is a work in progress, further development is targeted in the near future with respect to emerging hot-topic areas for large-scale data-intensive applications.

Data availability under fault tolerance is important because as datacenters grow to huge sizes, failures are rather the norm that the exception. We have implemented preliminary replication-based fault tolerance techniques into BlobSeer, but further refinement is necessary in this direction.

Cloud computing gains popularity as a cost-effective way to leverage computing and storage resources. The majority of applications running in this context are data-intensive. Adapting BlobSeer to a cloud middleware (such as Nimbus) to offer scalable and performant cloud storage (i.e., for use as virtual machine management in a highly-available IaaS or as an advanced data sharing service for applications running on top of clouds) is another important direction we intend to pursue.

References

[1] M. P. Herlihy and J. M. Wing, "Linearizability: a correctness condition for concurrent objects," vol. 12, no. 3. New York, NY, USA: ACM, 1990, pp. 463–492, concurrency control.

[2] B. Nicolae, G. Antoniu, and L. Bougé, "Enabling high data throughput in desktop grids through decentralized data and metadata management: The blobseer approach," in *Proc. 15th International Euro-Par Conference on Parallel Processing (Euro-Par '09)*, ser. Lect. Notes in Comp. Science, vol. 5704. Delft, The Netherlands: Springer-Verlag, 2009, pp. 404–416.

[3] ——, "BlobSeer: How to enable efficient versioning for large object storage under heavy access concurrency," in *Proc. 2nd Workshop on Data Management in Peer-to-Peer Systems (DAMAP'2009)*, Saint Petersburg, Russia, Mar. 2009, held in conjunction with EDBT'2009.

[4] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[5] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 59–72, 2007.

[6] V.-T. Tran, G. Antoniu, B. Nicolae, and L. Bougé, "Towards a grid file system based on a large-scale blob management service," in *CoreGRID ERCIM Working Group Workshop on Grids, P2P and Service computing*, 2009.

[7] T. D. Thanh, S. Mohan, E. Choi, S. Kim, and P. Kim, "A taxonomy and survey on distributed file systems," in *NCM '08*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 144–149.

[8] F. B. Schmuck and R. L. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *FAST '02: Proceedings of the Conference on File and Storage Technologies*. USENIX Association, 2002, pp. 231–244. [Online]. Available: http://portal.acm.org/citation.cfm?id=651312

[9] PVFS, "Parallel virtual file system, version 2," http://pvfs2.org/.

[10] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *SIGOPS - Operating Systems Review*, vol. 37, no. 5, pp. 29–43, 2003.

[11] "HDFS. The Hadoop Distributed File System," http://hadoop.apache.org/common/docs/r0.20.1/hdfs_design.html.

[12] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The globus striped gridftp framework and server," in *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2005, p. 54.

[13] Y. Jégou, S. Lantéri, J. Leduc, M. Noredine, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and T. Iréa, "Grid'5000: a large scale and highly reconfigurable experimental grid testbed." *International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, November 2006.

[14] B. Nicolae, G. Antoniu, and L. Bougé, "Distributed management of massive data: An efficient fine-grain data access scheme," in *VECPAR*, 2008, pp. 532–543.

[15] ——, "Enabling lock-free concurrent fine-grain access to massive distributed data: Application to supernovae detection," in *CLUSTER*, 2008, pp. 310–315.

[16] B. Nicolae, D. Moise, G. Antoniu, L. Bougé, and M. Dorier, "Blobseer: Bringing high throughput under heavy concurrency to hadoop map/reduce applications," in *IPDPS '10*, 2010, p. In press.

[17] J. Montes, A. Sánchez, J. J. Valdés, M. S. Pérez, and P. Herrero, "Finding order in chaos: a behavior model of the whole grid," *Concurrency and Computation: Practice and Experience*, p. In press., 2009.