Architecture of Distributed Data Aggregation Service

Vlad Serbanescu*, Florin Pop, Valentin Cristea
University Politehnica of Bucharest, Romania
Faculty of Automatic Control and Computers
Emails: vlad.serbanescu@cti.pub.ro, {florin.pop, valentin.cristea}@cs.pub.ro

Gabriel Antoniu
INRIA Rennes-Bretagne Atlantique, France
Email: gabriel.antoniu@inria.fr

*Current affiliation: Centrum Wiskunde & Informatica (CWI),
Amsterdam, Netherlands

***Abstract*—The ever-growing trend of deploying applications over the Internet has resulted in increasingly tougher constraints and requirements. Data management systems are a major concern when it comes to scalability, flexibility and reliability due to being implemented in a distributed way. In this paper we present a Distributed Data Aggregation Service relying on a storage system designed to meet these demands, namely BlobSeer. The primary goal is to serve as a repository backend for complex analysis and automatic mining of scientific data (like bibtex entries). Several requirements, derived from this objective, match BlobSeer's features: versioning used for lock-free access to data and different granularity of read / write operations. We proposed a model to perform the correct translation between BlobSeer's unstructured view of data and the user's structured view. We implemented a client providing a formal description for the data retrieval queries and a specification for a search API. A benchmark tool relying on a performance model of BlobSeer, will be used to automatically determine the best BlobSeer deployment configuration for a specific data aggregation pattern.**

Keywords: Aggregation Service, Data Storage, BlobSeer, Distributed Architecture, Cloud Computing.

I. Introduction

Applications over the Internet **have** continuously evolved into very complex services used for solving difficult and elaborate problems with various requirements and constraints. Distributed systems have now reached Exascale dimension [13] processing very large amounts of data on a regular basis. The distributed data are also at Exascale dimension and storage is a continuous open research problem addressing scalability and high performance for concurrent access.

The emergence of recent infrastructures has provided typical environments for such data-intensive applications with high requirements in terms of performance, cost, availability, ease of use, reliability and others, enabling access to a large number of resources and guaranteeing a predictable Quality of Service (QoS). However, as the exponentially growing data is correlated with an increasing need for fast and reliable data access, data management continues to be a key issue that highly impacts on the performance of applications, as the overall application performance is highly dependent on the properties of the data management service [2]. Therefore data management and aggregation has become a critical requirement in a wide spectrum of research domains, ranging from data-mining, monitoring repositories and digital libraries to high-energy physics [1],climate simulations or matrix equation solving. In this context, the technology above and within the Internet continues to advance reaching now a point where the potential benefits of very large scale are in service for distributed applications.

Any efficient data-management system that is intended for very large scales has to address a series of particular challenges, such as a scalable architecture, data location transparency, high throughput under concurrent accesses, the storage of massive data with fine grain access and has to cope with common problems of large scale distributed systems such as the integration of diverse technologies used in different parts of the distributed systems, tolerance to faults, ensuring confidentiality and protection, efficient use of resources [7]. Additionally, while data-intensive applications have specific demands for storing and retrieving data in large repositories, compute-intensive applications require more processing power, while collaborative applications need the ability to share and co-create knowledge in large organizations.

More specifically, data-management systems work with high volumes of structured data that needs to be stored efficiently and accurately in order to be available for further use. These high volumes of distributed data are always structured in a particular form that needs to be consistent while managing and collecting data adding to the complexity and difficulty of configuring such a system [10]. Furthermore they need to provide methods for finding and retrieving data in a secure and effective manner. Furthermore all these features need to be fulfilled with minimized access and computation costs, while at the same time ensuring high levels of fault tolerance and data consistency.

The main objective of this paper is to to develop a Distributed Data Aggregation Service (DDAS) relying on a distributed data management system, namely BlobSeer. The service will be implemented to respect all the requirements and constraints imposed by data-intensive applications and will utilize multiple features of BlobSeer [11] such as data stripping, distributed metadata management and versioning-based concurrency control. The DDAS will be designed to ensure scalability, fault tolerance and data retrieval performance.

The specific objectives focus on main features and specific scenarios for DDAS. Firstly, all data that is manipulated in this system is represented in long sequences of bytes of unstructured data that is read and written into binary large objects (BLOBS). As our application's main goal is to have a repository back-end for complex analysis and automatic mining of scientific data, its view of the data received from client's will be structured. Therefore a lot of effort will be

put into developing a new a new meta-data management layer needed to make the correct translation between the two views. Secondly a thorough analysis of several of BlobSeer's attributes needs to be done in order to establish the best deployment configuration for a specific data aggregation pattern. This will be done by studying characteristics such as the fixed size of a blob's page or the distribution of related and unrelated data between BLOBS.

The rest of this paper is structured as follows. Section 2 presents a critical overview of the existing solutions for data storage, aggregation and retrieval in Large Scale Distributed Systems. The model for the Distributed Data Aggregation Service together with the model for translating between the two views described in section 3. The architectural model of the entire system analyzing with each layer and component of the solution is analyzed in section 4. Section 5 shows the implementation of the system and the a real use-case scenario for the solution: Application for aggregating scientific data. section 6 draws the conclusions of the work done so far and proposes the future work of this project.

II. Related Work

As mentioned in the previous section there are many issues involving the manipulation of large volumes of distributed data. All operations involving this data, including storage and aggregation, come with very high computation and communication costs. In this section we will look at several solutions for performing these operations while minimizing these costs, as well as tackling all the other problems that appear in Large Scale Distributed Systems. The increasing maturity of cloud computing technology is leading many organizations to migrate their IT infrastructure and/or adapting their IT solutions to operate completely or partially in Cloud.

A. Data Storage Solutions

As applications work with a continuously growing volume of data they require storage that can achieve high achieve scalability and performance. With cloud computing technology evolving at a very fast rate, many IT solutions tend to operate completely or partially in the cloud allowing full usage of its features. Cloud storage provides manageability (the ability to manage a system with minimal resources), a great number of access methods (protocols through which cloud storage is exposed) and multi-tenancy (support for multiple users). At the same time this ensures scalability (ability to scale to meet higher demands or load in an efficient manner), data availability (a measure of a system's uptime) with data being replicated over several systems and storage efficiency (measure of how efficiently the raw storage is used). Furthermore cloud storage has yielded excellent results for system control (configuring for cost, performance, or other characteristics) and (measure of the cost of the storage).

The first solution studied is in [3] which proposes A Robust and Flexible SuperPeered Distributed Hash Table (DHT). The solution provides a simple technique applicable to the majority of existing DHT systems that involves hiding a subset of the nodes that make up the DHT, from the overlay. This enables the use of super-peers in DHT-based networks, while avoiding the deficiencies of a classical DHT which assume that all nodes that make up the DHT can perform all the lookup operations supported at an equal performance level. Additionally it overcomes the high maintenance overhead and single point of failure that improved super-peer DHT has. However this solution comes at the cost of placing high loads of data on the nodes that are visible in the overlay, which may cause very high latencies if these nodes to do not have great computing capabilities.

Another storage system is proposed in [5] and it's main purpose is to handle large amounts of structured data while taking into consideration the varying data size and latency requirements. The model used is that of a a sparse, distributed persistent multi-dimensional sorted map containing uninterpreted array of bytes. It also maintains a versioning system of the data indexed by timestamps. On one hand the main advantages of this solution are that it provides high performance lookup, scalability, high availability of data and it is used in many real applications. On the other hand the it also has disadvantages due to its implementation which requires difficult configurations of the clusters and also places a heavy load on the master server of the solution.

The S3 storage system proposed by Amazon [12] aims to provide storage as a low-cost, highly available service, with a simple 'pay-as-you-go' charging model. The positive aspects of this model are the high-level access control and the global availability of the system. It does however have many negative aspects such as not having any form of Service Level Agreement (SLA) to maintain stored data and also having the possibility of losing all the stored data if something as simple as your email account is compromised.

We also look at the solution presented in [9] where the data storage environment is implemented to handle a very high write throughput and also scale with the number of users. Although the system has many advantages such as the ability to scale incrementally, using replication to ensure high availability and durability and failure detection, Cassandra also has to deal with certain issues such as non-uniform data and load distribution.

Finally we analyze the data storage solution provided by BlobSeer [11]. This solution represents data as BLOBS taking into consideration that most data in circulation is unstructured. This gives the possibility of ensuring scalability using the same BLOB to store large amounts of data by only maintaining the offset of the BLOB. Along with these features BlobSeer also provides the user with a versioning-oriented access interface for manipulating blobs, therefore allowing application-level parallelism as an older version can be read while a newer version is generated. To interact with BlobSeer all that is required is a handle that points to a specific BLOB from which data is extracted or to which data is stored. Minimizing the number of handles that will be created during a request was a major priority in our implementation.

B. Aggregation solutions

In the first solution in [8] a method for aggregating web-service data is presented. This framework employs a set of interconnected aggregation nodes, which cooperate with each other to execute client requests. This aggregation solution provides great response times and high throughput when requests involve a large number of aggregation nodes with each one

handling a low number of requests. However if the request load is not distributed uniformly and a low number of aggregation nodes are used the throughput and performance are very low.

The second solution [14] evaluates the interfaces and implementations for user-defined aggregation in several state of the art distributed computing systems. The User-defined aggregation in Hadoop implementations make user responsible for understanding the defined types and using casts or access functions to fill in the required fields. This apparently adds a lot of complexity to trivial computation, however for more complicated aggregation functions the overhead of casting between system types will be less noticeable, and the benefits of having access to a full-featured high-level language, in this case Java, will be more apparent. The interfaces of User-defined aggregation in a database show the benefits that built-in database functions have when writing an aggregation method, but also the limits of database languages when user-defined functions and types are more complex.

Another solution [6] presents the collective operations implemented in MPI. These operations process data over several processors using functions like MPI-Bcast, MPI-Scatter or MPI-Gather and even aggregate data using MPI-Reduce. These collective functions can also be used to work on a shared object such as a file using MPI collective I/O. The main advantage of these operations is that processes can return from an aggregating call without waiting for the completion of other processes. However,to ensure good response times several request scheduling algorithms have to be tested for a specific problem that uses MPI collective I/O.

Using BlobSeer for an aggregation solution allows the convenient placing of objects to train the DDAS for future operations. The system organizes data in BLOBS, marking each storage operation with a new version, it can prove very effective in the complex aggregation process. With a model that collects objects that match common attributes, our service will send data to BlobSeer such that all retrieval and aggregation operations for a specific pattern will be reduced to reading data from the right location in BlobSeer.

III. Distributed Data Aggregation Service (DDAS) Model

In this section we present the proposed model for the Distributed Data Aggregation Service that will be implemented based on BlobSeer's features. We will describe the view that our service will have of the objects that it handles. Moreover we show how application specific objects with a certain structure are serialized in order to interact with the storage system while at the same time maintaining a view of their structures for all the basic and complex aggregation operations. Also we will follow the steps taken by the DDAS to process each storage and retrieval request.

A. Structured objects view

As was presented in our related work section, we already know that BlobSeer stores data as unstructured blobs. This feature allows our service to handle objects of any type, however we require a method through which applications can describe their objects before they are processed and stored by the DDAS. The main idea is to map each object to at least one scheme. A scheme is represented by a set of key-value pairs that are the properties of the object. In addition, a scheme can have one or more reduce functions. The **main scheme** of an object must contain a **key** that uniquely identifies the object, an **entry** that is a stream that represents the object and all the properties that the objects have in common. For example, for an application that processes publications all objects must have four properties: a key, a format, a digital library from which it comes and the entry containing the actual publication, and a value associated to each of them. Therefore all objects stored in the DDAS must be mapped to this scheme. In addition each storing operation can also map the object to **a scheme that expands** its entry property to more key-value pairs that identify particular properties of the object.

All retrieval operations imply an input scheme which the DDAS uses to identify all of the objects requested. This is known as an **aggregation scheme** and requires a lookup on all the objects stored the first time this operation is called on the DDAS.

B. Object Storage

With a model for retaining each object's structure, the DDAS can store the main scheme of an object into BlobSeer. The DDAS is now responsible for mapping the key of the object to the meta information that points the the object's main scheme in BlobSeer and maintains a catalog for all these mappings known as the **object catalog**. Furthermore it maintains a catalog of all the expansion and aggregation schemes that are each mapped to a list of object meta information that expand to that scheme. This catalog is known as the **metadata catalog**. A big concern of this model is how the size of this metadata catalog will evolve as more retrieval operations add new schemes to it. We therefore proposed a checkpoint feature that stores this metadata catalog in BlobSeer as well and only reads it on demand in order to complete an operation. Finally we attempt to spread objects into blobs as evenly as possible by storing objects that map to a new scheme in a new BLOB, unless the object has already been stored and already fits an existing scheme. This means that when a retrieval operation is requested based on an aggregation scheme, most objects that fit the scheme in the metadata catalog will be stored in the same BLOB. For applications that process very large objects in size but not in numbers, the object catalog would not become a bottleneck due to the low number of keys, however this catalog is stored in a distributed manner, taking advantage of BlobSeer's features, for applications that deal with a growing number of small objects that are frequently accessed and modified.

C. Request Processing Model

The main purpose of the DDAS is to maximize BlobSeer's features without performing extensive lookup throughout an application's entire collection when retrieving data. Therefore aside from when a retrieval on an aggregation scheme that does not exist inside the meta-catalog, all other operations involve only reading and writing in particular blobs of the storage system.

When a request to store new data is made, the new object is stored inside the object catalog and its entry is expanded to

all the existing schemes in the metadata catalog that this new object fits. If the object is stored with a new expansion scheme that is not found in the metadata catalog, all the existing objects are also checked if they fit the new scheme and the metadata catalog is updated accordingly. Although this operation can be very costly, we assumed that storage operations are sparse and focused on reducing the cost of writing operations.

The model for processing retrieval operations simply involves either retrieving the list inside the metadata catalog for the scheme requested or an entire lookup on all objects if the scheme does not exist. Therefore the process of aggregation is continuous whenever a write operation is completed, allowing very efficient data retrieval operations.

IV. Architecture Design

The architecture is made of of several components each with a well-defined purpose to ensure the fast processing of a request. Application requests can either require a large number of objects or huge data size for storage and retrieval operations, therefore we need to efficiently isolate the system's functions when it comes to computation, storage, selection and metadata management. Regardless of the type of operations, the flow of data will go through these functions and will train the DDAS to facilitate future requests.

A. Data Back-end Storage System

This is the actual BlobSeer storage system that we described in our related work and motivated its choice. It only communicates with the DDAS' extended client through read and write requests to store and retrieve specific objects in their serialized form according to the model described in section 3. For write operations, the meta data manager returns the meta-information to identify the location of the written objects, while for read operations it returns the serialized object based on the meta-information input by the DDAS extended client.

B. DDAS

This component is the mediator between BlobSeer and all other data-management applications that require fast and reliable storage and retrieval of data. Its main role is to translate between BlobSeer's unstructured view of data and the different structured objects that each application handles. It is divided into two layers: the metadata management layer and the extended BlobSeer client. Regardless of the type of request or application, the upper layer uses the Collect Gate module to update its scheme-object mappings and performs the operation requested: either storing the serialized object into BlobSeer or retrieving the mapping of an aggregation scheme from BlobSeer and delivering it with the right structure to the client. The extension of the BlobSeer client writes data into BLOBS specified by the information in the metadata catalog. Each scheme is assigned a BLOB to which new objects that fit the scheme will be stored. If an object fits more than one scheme then all schemes will simply point to the same meta-information and the object will only be stored in one BLOB. For an in put aggregation scheme, the DDAS either identifies it in the metadata catalog or if it does not exist, it will request from Collect Gate the list of object keys that fit the aggregation scheme. Using the keys or the existing aggregation scheme the service will send to the extended client the meta-information based on which the retrieval of data from BlobSeer will be made.

C. Collect Gate

This module is a represented by a database of rules and properties that, depending on the application, generates expansion and aggregation schemes that fit one or more objects. Furthermore, Collect Gate creates lists of object keys that map to a scheme to send them to the DDAS in order to fulfill new requests. The idea behind this module is to isolate properties required for aggregation operations and reduction functions from the object itself that can occupy much more space. The role of this component is to perform quick searches and selections based on a few properties while leaving the storage and retrieval of the actual objects to the component designed specifically for this. As storage is required for new objects, Collect Gate's role is to match them to existing schemes from the DDAS's metadata catalog such that, after storage, the matching schemes point to the newly created meta-information.

D. Data Flow

This section studies how objects are handled by each of the components in Figure 1 during a request made by a client to store data or retrieve a set of objects based on an aggregation scheme.

When an application (i.e a crawler) wants to store objects it first collects a massive amount of data from various sources. Then the application makes a storage request to the DDAS with a large set of objects as input. For a new object the DDAS queries Collect Gate for all the existing schemes that fit the new object. Optionally the new object may come with an expansion scheme, which determines an extra query to Collect Gate to obtain all the existing objects that fit the expansion scheme. The metadata management layer is now responsible for two internal operations: mapping the matching schemes to the new object's meta information after it is stored and updating its metadata catalog with the new expansion scheme and the meta-information of all objects that match it. The extended BlobSeer client has to store the new object into the BLOB that was associated to the expansion scheme or one of the BLOBS associated to an existing scheme that matches the objects. Finally BlobSeer executes the write operation and returns the corresponding meta-information to the DDAS. We anticipate that the number of storing requests into our system will be sparse in comparison to the aggregation requests, hence train the system as much as possible during the storage operations.

An aggregation operation requires an client application to input an aggregation scheme in its request to the DDAS. If this aggregation scheme is not already in the metadata management layer of the DDAS, a request to Collect Gate is made to determine the list of object keys that match this scheme. After Collect Gate returns the list or if the scheme already exists in the DDAS, the extended BlobSeer client will read the objects based on the meta-information of each object from the object catalog or meta-information mapped to the existing aggregations scheme. BlobSeer will then read all requested objects and return them to the DDAS. Finally the

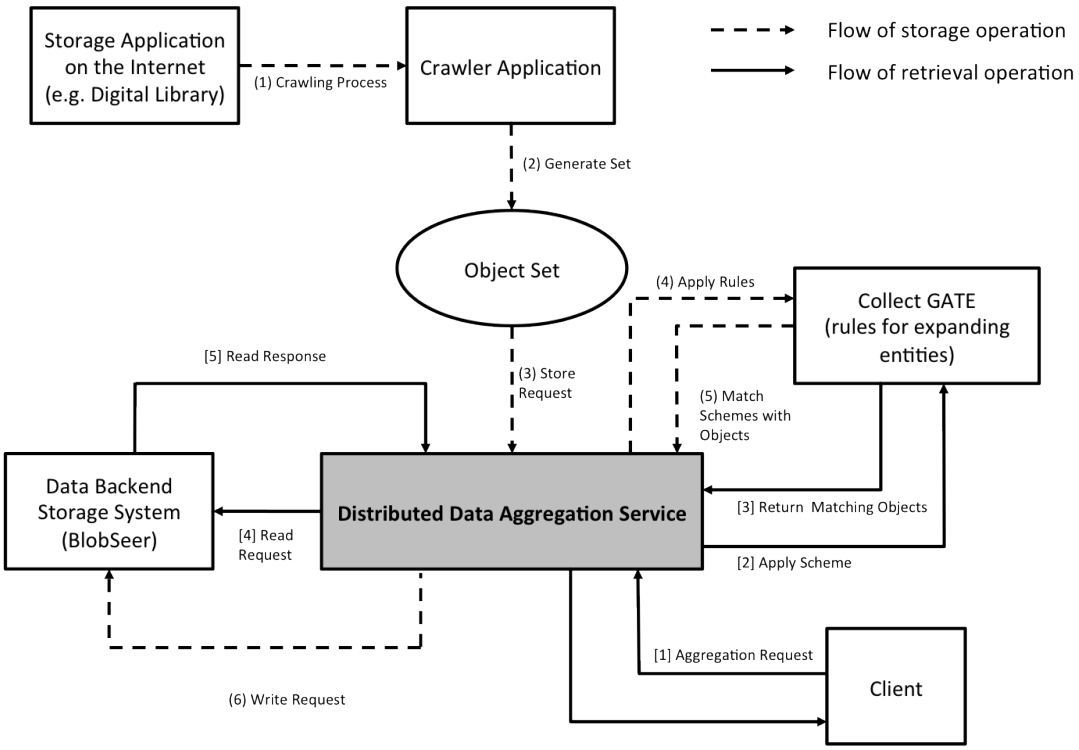

Figure 1. Overview of the System Architecture

DDAS will perform any reduction operations that are included in the aggregation scheme. The list of objects will then be transmitted to the client application in the same format as it was first stored.

V. Experimental Methodology and Results

This section describes how the modules in the architecture are implemented and the test cases used to validate our solution. We consider the data structures used to minimize the lookup overhead introduced by the DDAS, the mechanisms used to handle concurrent requests and how a client or application sends requests using the schemes described in our model. We will describe two real environment applications that will be tested on the DDAS and present the results obtained for the first test scenario.

A. Data Structures and Concurrency Control

A priority for data management is to handle numerous requests from multiple clients all over the cloud, therefore our DDAS has to maintain a consistent metadata catalog and object catalog. Furthermore we need to ensure fast lookup of meta-information, especially for an existing aggregation scheme that does not require Collect Gate's function. Taking this into consideration we implemented the DDAS using JAVA Collections designed specifically for these requirements. The object catalog is represented by a `ConcurrentHashMap` with the object unique identifiers as keys and the meta-information of the objects (as retrieved by the extended client) as values. Additionally, the metadata catalog is represented by the same structure that maps schemes with lists of meta-information that represent the objects which fit the scheme.

B. DDAS Interaction

In order for a client application to interact with the DDAS we need to provide a formal description of the format of the input and output of schemes and objects. We do this through the means of XML files. The main scheme that represents an object is made up of a root tag and inner tags which represent the object's properties with their values as text content. All objects, independent of the application, must have the "key" and "entry" tags. Similarly, an expansion scheme has the same XML format, however we assert that it is significantly smaller in size as it does not contain a tag with the entire object. These two types of XML files are input whenever a storage operation is required with the main object scheme as a compulsory argument and the expansion scheme optional. The following is an example of an XML file with the scheme for a BibTex object used in our first test scenario.

```
<?xml version = "1.0" ?>
<root>
<key> 2082445 </key>
<type> bib </type>
<dl> acm </dl>
<entry>
@inproceedings{2082445,
 author = {Potlog, Alina-Diana and
  Xhafa, Fatos and
  Pop, Florin and
  Cristea, Valentin},
title = {Evaluation of Optimistic
  Replication Techniques for
  Dynamic Files in P2P Systems},
booktitle = {3PGCIC '11: Proceedings
```

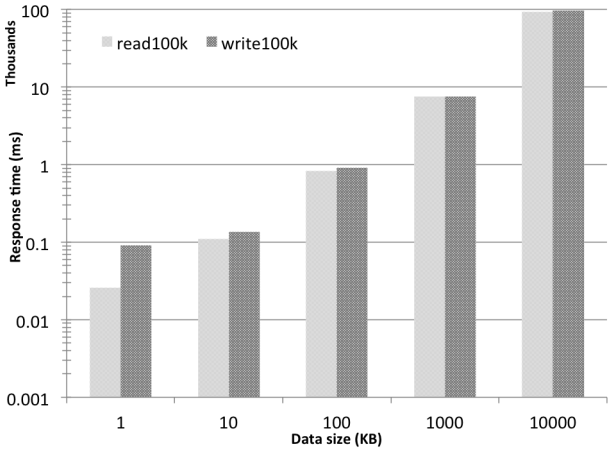

Figure 3. BlobSeer Evaluation Results (response) for read/write different Data size (vertically logarithmic scale).

```
  of the 2011 International
  Conference on P2P, Parallel,
  Grid, Cloud and
  Internet Computing},
year = {2011},
isbn = {978-0-7695-4531-8},
pages = {259--265},
publisher = {IEEE Computer Society},
address = {Washington, DC, USA},
}
</entry>
</root>
```

The aggregation scheme is the argument required by the retrieval request and its XML format contains a root tag with several inner tags with explicit names. First, the ”select” tag contains inner tags with properties and values that objects must match. Second, the ”exclude” tag has inner tags that represent the values for properties that objects must not have in order to be retrieved. Finally, the aggregation scheme can contain multiple ”function” tags which in turn have inner tags describing the property on which the function is applied, the type of the property, the number of operands on which the function works and the result type. The function must also have a name which is the path to one of the DDAS predefined functions. An example of an aggregation scheme is the following:

```
<?xml version = "1.0" ?>
<root>

<select>
 <atribute1> v1 </atribute1>
 <atribute2> v2 </atribute2>
</select>

<exclude>
 <atribute1> v1 </atribute1>
 <atribute2> v2 </atribute2>
</exclude>

<function name="sum">
 <property>property_name</property>
 <propertyType>Int</propertyType>
 <operands>2</operands>
 <result>Integer</result>
</function>

</root>
```

C. Test Scenarios and Evaluation

As we described in the previous section the DDAS is implemented on top of BlobSeer’s meta-data management scheme and storage system. Therefore we first needed to analyze a series of configurations involving the page size of a BLOB in which data is stored and how varying sizes of data affect the overhead of read and write operations. We performed tests by deploying BlobSeer on the Grid’5000 scientific instrument [4] across 20 virtual machines located at three different sites. We varied both the actual data size and the page size of the blobs and obtained the results in Figure 2. Clearly, a large page size outputs better response times for large size data, however an issue appears as most of the data stored such as documents does not not pass a few KB size, therefore a large page size would not be necessary and would be a large overhead for numerous small objects. We established that a page size of about 4-16KB would yield the best results for a large number of objects with a size up to 32KB that will be presented in the first test scenario.

D. Application for aggregating scientific data

Most scientific documents such as articles, books and journals has used until now mostly databases to store, retrieve and aggregate data. The use of databases allowed the maintenance of structured data in relation with attributes and fast searches using indexes. However this data is continuously growing and has finer granularity therefore this model uses disk space inefficiently and creates large additional data through indexes. Our tests on the DDAS for aggregating scientific data evaluate the specific characteristics of this applications such as the granularity for read and write operations, the manageability of large volumes of data and how the documents remain persistent over long periods of time. The results of read/write experiment presented in Figure 3 allow to establish an upper bound for data size to 10MB for 1KB page-size blob. Additionally we look at how this specific application support data faults using BlobSeer’s data replication scheme and how efficiently can a large number of clients retrieve and aggregate data. The integration of DDAS for this specific application in a distributed storage system is presented in Figure 4.

VI. Conclusion

As data management and aggregation continues to evolve in a wide spectrum of research domains and requirements become more specific and complex, the need for high performance solutions for data intensive applications in Large Scale Distributed Systems grows significantly. In this paper we proposed a Distributed Data Aggregation Service (DDAS) relying on BlobSeer. We presented the model of the DDAS in order to ensure a high level of performance in all aspects of a data storage and aggregation solution. Also we emphasized on

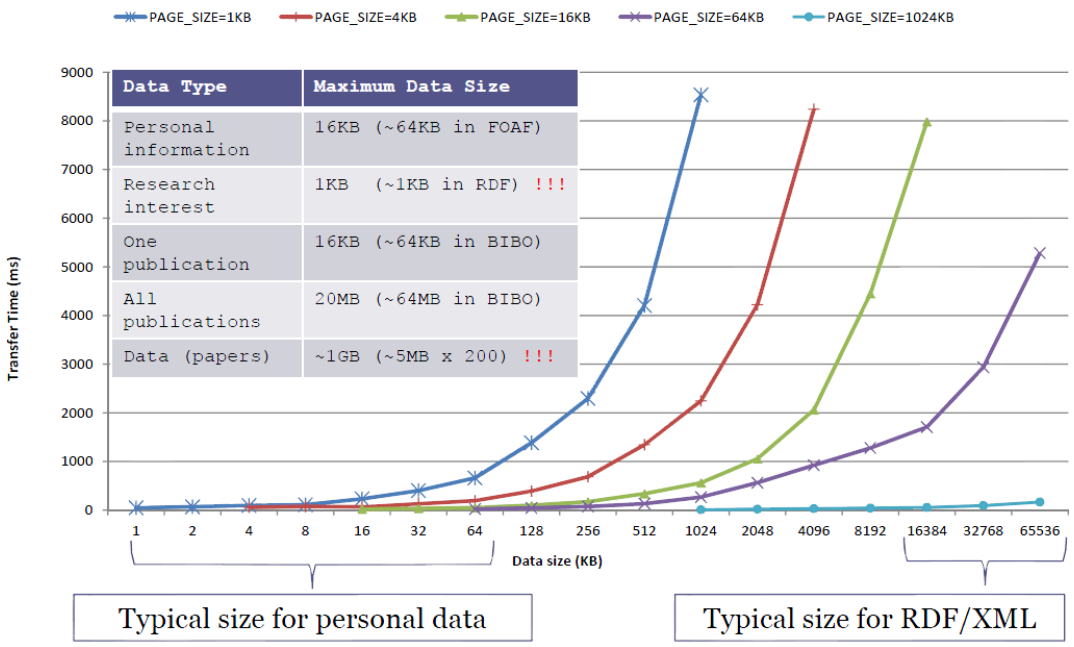

Figure 2. BlobSeer Evaluation Results for different Data type and size.

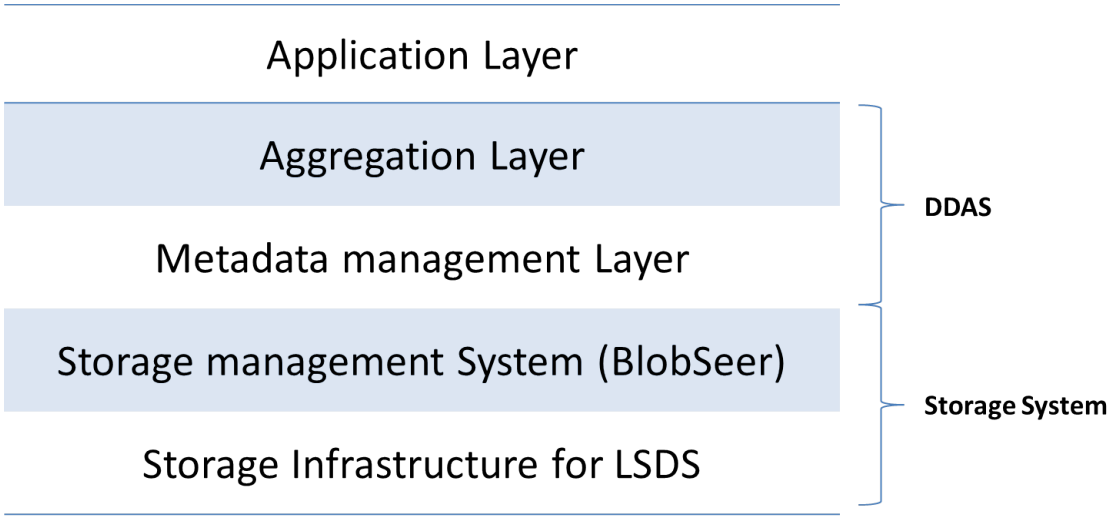

Figure 4. Integrated view of DDAS in a distributed storage system.

how several features of BlobSeer will match the constraints of our solution.

The future work will focus on testing the solution and improving the times obtained by implementing the data structures of the DDAS in a distributed manner. Additionally we will evaluate our solution using the second data-intensive application described in section 5 and finally we plan to run more tests to determine the best BlobSeer deployment configuration for a specific data aggregation pattern.

ACKNOWLEDGMENT

The research presented in this paper is supported by projects: "*SideSTEP - Scheduling Methods for Dynamic Distributed Systems: a self-* approach*", ID: PN-II-CT-RO-FR-2012-1-0084; "*ERRIC - Empowering Romanian Research on Intelligent Information Technologies*", FP7-REGPOT-2010-1, ID: 264207; and by CyberWater grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, project number 47/2012.

The work was developed under the DataCloud@Work associated team between KerData and Myriads teams from INRIA Rennes - Bretagne Atlantique and the Computer Science Department from Politehnica University of Bucharest

The work is partly funded by the EU project FP7-610582

ENVISAGE: Engineering Virtualized Services (Web page: `http://www.envisage-project.eu`)

We would like to thank the reviewers for their time and expertise, constructive comments and valuable insight.

REFERENCES

[1] K. Aamodt *et al.*, "The ALICE experiment at the CERN LHC," *JINST*, vol. 3, p. S08002, 2008.

[2] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "Depsky: dependable and secure storage in a cloud-of-clouds," in *Proceedings of the sixth conference on Computer systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 31–46. [Online]. Available: http://doi.acm.org/10.1145/1966445.1966449

[3] A. Brampton, A. MacQuire, I. A. Rai, N. J. P. Race, and L. Mathy, "Stealth distributed hash table: a robust and flexible super-peered dht," in *Proceedings of the 2006 ACM CoNEXT conference*, ser. CoNEXT '06. New York, NY, USA: ACM, 2006, pp. 19:1–19:12. [Online]. Available: http://doi.acm.org/10.1145/1368436.1368462

[4] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard, "Grid'5000: A large scale and highly reconfigurable grid experimental testbed," in *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, ser. GRID '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 99–106. [Online]. Available: http://dx.doi.org/10.1109/GRID.2005.1542730

[5] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, pp. 4:1–4:26, June 2008. [Online]. Available: http://doi.acm.org/10.1145/1365815.1365816

[6] J. Chen, S. Sehrish, W.-K. Liao, A. Choudhary, and K. Schuchardt, "Improving the average response time in collective i/o," in *Recent Advances in the Message Passing Interface*, ser. LNCS 6090, 2011, pp. 71–73.

[7] T. Glatard, J. Montagnat, and X. Pennec, "Efficient services composition for grid-enabled data-intensive applications," in *Proceedings of the IEEE International Symposium on High Performance and Distributed Computing*, Jun. 2006, pp. 333–334. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00683201

[8] W. Hummer, P. Leitner, and S. Dustdar, "Ws-aggregation: distributed aggregation of web services data," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, ser. SAC '11. New York, NY, USA: ACM, 2011, pp. 1590–1597. [Online]. Available: http://doi.acm.org/10.1145/1982185.1982520

[9] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 35–40, April 2010. [Online]. Available: http://doi.acm.org/10.1145/1773912.1773922

[10] S. Leo and G. Zanetti, "Pydoop: a python mapreduce and hdfs api for hadoop," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 819–825. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851594

[11] B. Nicolae, G. Antoniu, L. Bougé, D. Moise, and A. Carpen-Amarie, "Blobseer: Next-generation data management for large scale infrastructures," *J. Parallel Distrib. Comput.*, vol. 71, pp. 169–184, February 2011. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2010.08.004

[12] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon s3 for science grids: a viable solution?" in *Proceedings of the 2008 international workshop on Data-aware distributed computing*, ser. DADC '08. New York, NY, USA: ACM, 2008, pp. 55–64. [Online]. Available: http://doi.acm.org/10.1145/1383519.1383526

[13] S. Venugopal, R. Buyya, and K. Ramamohanarao, "A taxonomy of data grids for distributed data sharing, management, and processing," *ACM Comput. Surv.*, vol. 38, June 2006. [Online]. Available: http://doi.acm.org/http://doi.acm.org/10.1145/1132952.1132955

[14] Y. Yu, P. K. Gunda, and M. Isard, "Distributed aggregation for data-parallel computing: interfaces and implementations," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 247–260. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629600