World Wide Web Journal manuscript No.
(will be inserted by the editor)

A Formal Method for Rule Analysis and Validation in Distributed Data Aggregation Service

Vlad Serbanescu · Florin Pop · Valentin Cristea · Gabriel Antoniu

Received: date / Accepted: date

Abstract The usage of Cloud Serviced has increased rapidly in the last years. Data management systems, behind any Cloud Service, are a major concern when it comes to scalability, flexibility and reliability due to being implemented in a distributed way. A Distributed Data Aggregation Service relying on a storage system meets these demands and serves as a repository back-end for complex analysis and automatic mining of any type of data. In this paper we continue our previous work on data management in Cloud storage. We present a formal approach to express retrieval and aggregation rules with a compact, yet powerful tool called Rule Markup Language. Our extended solution proposes a standard form to schemes and uses the tool to match the rules to the XML form of the structured data in order to obtain the unstructured entries from BlobSeer data storage system. This allows the Distributed Data Aggregation Service (DDAS) to bypass several steps when processing a retrieval request. Our new architecture is more loosely-coupled with a separate module, the new tool, used for transforming the XML entries to standard XML files which represent the final result. We model the dynamic behavior of the system using this new standard to ensure a simpler and efficient representation of the operations performed by the client while maintaining the

Vlad Serbanescu - Principal Author
PhD Student, Formal Methods Department, Centrum Wiskunde & Informatica, Amsterdam, Netherlands, E-mail: vlad.serbanescu@cwi.nl

Florin Pop - Corresponding Author
Associate Professor, Computer Science Department, Faculty of Automatic Control and Computers, University *Politehnica* of Bucharest, Romania, E-mail: florin.pop@cs.pub.ro

Valentin Cristea
Professor, Computer Science Department, Faculty of Automatic Control and Computers, University *Politehnica* of Bucharest, Romania, E-mail: valentin.cristea@cs.pub.ro

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

constraints imposed by a distributed system running in the Cloud. Furthermore we prove that this method correctly performs the translation between the storage model's unstructured view of data and the client's structured objects.

Keywords Data Aggregation · Data Management · Cloud Storage · Intelligent Cloud Services · Distributed Services · Formal Methods · Rule Markup Language

PACS Distribution theory, 02.50.Ng

Mathematics Subject Classification (2000) 68M20 · 68M14 · 68U20

1 Introduction

As applications work with a continuously growing volume of data they require storage that can achieve high 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. Distributed systems have now reached Exascale dimension [21] processing very large amounts of data on a regular basis. 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) [3]. 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.

In this paper we extend our proposed Distributed Data Aggregation Service (DDAS) [16] using a formal method for representing queries and aggregation operations. As the volume of data stored in Cloud environments grows, we consider that there is a need for formal methods and techniques for QoS analysis and ensuring consistency. With the development of private Clouds authentication, portability and interaction with the Cloud are becoming significant issues that we aim to tackle by adding formality to the user's operations. The motivation of this paper resulted from the heterogeneity of data present in Big Data applications (digital libraries, e-Health applications, Smart Cities Apps, multimedia applications) and the need to have a standard of representing data in an expressive and analyzable format suitable for various tools and services.

Continuing with our approach to represent the structure of objects in XML format along with expanding their attributes, we want to define a standard form for selecting, filtering and aggregating objects using a Rule Markup Language (RML) tool [10,18]. RML was designed for ease of use to write patterns that match certain elements in an XML file and apply transformation rules to the file to produce a desired output. The tool has been tested and validated, therefore we can maintain the fast and reliable data access shown in our previous work without correctness issues as we show in this paper. The analysis of the data elements is done using this formal method by pre-processing data in the RML standard. Queries are defined as XML rules and expansion and aggregation schemes now have a standard format to fit as input for the RML tool. The integration of the tool and its performance impact is based on experimental results.

The scope of our problem now covers a broader area, not only in terms of high requirements for performance, cost, availability, reliability and enabling access to a large volume of resources by a large number of clients, but also in terms of computer-human interaction and ease of programming. RML is designed to be very usable and interoperable such that programmers can learn to use it in a short amount of time without a lot of background or experience in programming. This tool does not add any new technology that needs to be assimilated by clients, with only knowledge in XML and defining rules in RML needed to use the previous Distributed Data Aggregation Service.

As the volume of data and storage devices continues to grow, 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. All these domains have a different view of data and by introducing formal techniques and rules to define objects we can maintain a consistent and, at the same time, correct representation of operations when selecting and filtering data regardless of the application [8].

Our new solution continues to rely on BlobSeer for an efficient storage system that offers 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. However we introduce on top a general-purpose method for XML transformations and apply it to the objects represented in XML form. With this approach it is easy to avoid the complexity and difficulty of configuring the system for diverse structures of data.

Our main objective is to extend the proposed model with rules for representing the aggregation of data. We want to provide a formal analysis and validation of those new rules in order to prove correctness and efficiency in processing data. We aim to use the RML tool to handle the application of the rules to objects represented in XML form and obtain the output required by the DDAS and BlobSeer thereafter to read to objects from the cloud and provide the expected results to the client. Furthermore we want to keep the characteristics validated this far in our solution while providing the user with an easy to understand interface with out system. More specifically we seek to ensure the transparency of the transformations by integrating our RML tool as a separate module in the control flow of operations.

The development of DDAS relies on a distributed data management system, namely BlobSeer. The service is designed to respect all the requirements and constraints imposed by data-intensive applications and utilizes multiple features of BlobSeer [14] such as data stripping, distributed metadata management and versioning-based concurrency control. The DDAS is designed to ensure scalability, fault tolerance and data retrieval performance [16].

Another significant issue we tackled was the way we volatile data was stored while the system was running. First we had to store a mapping between the objects' identifiers and the metadata pointing to the location in BlobSeer; then we had to maintain a metadata catalogue for each expansion and aggregation scheme that maps to a list of all matching object identifiers generating a data explosion when adding new schemes. We had to both distribute the metadata catalogue over several machines and even checkpoint the metadata catalogue itself to BlobSeer to reduce the overhead of the program main memory. As we shall see in Section 3, we now create aggregation schemes on the fly and discard them after the RML tool does the matching process.

This paper has the following research and scientific contributions:

- A formal method for requests made by clients;
- A standard XML format for expansion and aggregation schemes;
- The integration of this standard and formal method into the DDAS.

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 Rule Markup Language solution is also presented in this section. The model for the Distributed Data Aggregation Service together with the formal 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.

2 Related Work

Our starting point for the research work is the Distributed Data Aggregation Service we proposed for manipulation of large volumes of data. In this section we look at some of the issues we had with our proposed model, including testing difficulties, metadata space management and communication. Nonetheless, we continue to make full use of the characteristics proven in our work which include minimized access and computation costs, high levels of fault tolerance and data consistency as well as data persistence and replication, as a good solution to improve QoS in different networks [11,22]. We study how the RML solution fits our model and mitigates some of the problems that arose in our previous work. We also illustrate the proofs of correctness of the aggregation rules that this new tool supports.

2.1 Data Storage and Aggregation Solution

The Distributed Data Aggregation Service was built to complement BlobSeer's features which maintain data in circulation unstructured, ensuring scalability via metadata which specifies an object's location, application-level parallelism using snapshots of an object's location, support for multi-tenancy and various types of objects. However, we needed to have a clear and simple method of processing objects based on an application's actual representation. We did this by mapping objects to a so-called scheme, a representation in XML format with a minimal number of elements. Each element was a key-value pair that together identified an application's view of its objects. To ease the interaction with BlobSeer, we imposed the constraint that each object needed to have a unique identifier element and an entry element with the actual stream representation of the object as its value. 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. Furthermore the application and the service both could expand the entry element to more specific elements in the same XML key - value format. A really big difficulty came whenever we had to create a new scheme for an aggregation operation as we had several steps to perform:

- A lookup on all objects and entry expansions for those that did not have the required scheme mapped to their key.
- A match between each expansion scheme and the required scheme.
- Storing the matching objects in our metadata catalogue with the new scheme as the key.

Therefore we considered that our system was ideal for performing a large number of read operations on already created schemes compared to write and new aggregation operations. In our current extension this assumption is no longer necessary as we modify our algorithm to use RML to define the new scheme and the tool to handle the matching described in Section 3.

Another significant issue we tackled was the way we volatile data was stored while the system was running. First we had to store a mapping between the objects' identifiers and the metadata pointing to the location in BlobSeer; then we had to maintain a metadata catalogue for each expansion and aggregation scheme that maps to a list of all matching object identifiers generating a data explosion when adding new schemes. We had to both distribute the metadata catalogue over several machines and even checkpoint the metadata catalogue itself to BlobSeer to reduce the overhead of the program main memory. As we shall see in Section 3, we now create aggregation schemes on the fly and discard them after the RML tool does the matching process.

The first solution studied is in [4] which proposes a robust and flexible super peered 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 [6] 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 [15] 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 [12] 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 [14]. 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.

2.2 Rule Markup Language (RML)

The main focus of the proposed solution is to establish a standard model for representing aggregation and expansion operations. For this we looked to have a formal approach in parsing the XML descriptions of objects, instead of doing the parsing inside the program. We required something simple that did not introduce new programming techniques for the user, but was sufficient to solve the issues presented in the previous subsection.

The RML that we studied provided the best solution for this. The underlying idea of this tool is to specify transformations in XML documents by means of rules which are formulated in a problem domain XML vocabulary of choice. These rules consist of a mix of XML from the problem domain and the Rule Markup Language. The output produced by this tool remains in the same XML format that we intend to use for the aggregation operation. The RML approach re-uses the problem domain XML as much as possible, with a “programming by example” technique. With this ruleâĂŞbased approach it becomes possible to define transformations in a much simpler way than for example XSLT the official W3C Recommendation for XML transformations.

Another solutions refer to the rules implicitly embedded in Web pages [13] or a single general rule language (REWERSE I1 Rule Markup Language - R2ML), which processes everything in a uniform manner.

Our purpose is to add a set of XML constructs to our XML vocabulary that represents an application’s objects and define define RML rules for that XML vocabulary. Specific RML tools can execute these rules, to transform the input XML according to the rule definition.

2.3 Aggregation Solutions

When applied to the web, aggregation finds applications in meta-searching, search engine comparison, spam fighting and word association techniques [2], and reduces communication overhead [24].

In the first solution in [9] 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 [23] 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 is 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 [7] 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.

3 Distributed Data Aggregation Service Model

In this section we present the application of a formal method to our expansion and aggregation schemes using the Rule Markup Language. As in our previous work, the main schemes remain the identical like the following XML format with a small modification because RML work on attributes and their names so all values of elements have to be attribute names:

```
<?xml version = "1.0" ?>
<root>
<key value ="2082445" />
<type value =  "bib" />
<dl name = "acm" />
<entry value = "
  @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 of the 2011 Int.
                 Conference on P2P, Parallel, Grid, Cloud and
                 Internet Computing},
    year = {2011},
    pages = {259--265},
    publisher = {IEEE Computer Society},
    address = {Washington, DC, USA},
  } "
/>
</root>
```

The first simplification comes with expansion schemes. As attributes are expanded from the entry, they are added with the same expansion scheme instead of creating separate schemes for different attributes.

The main focus of the application is now the attributes it needs to aggregate on, making it easier for a client to associate future aggregation rules with existing aggregation schemes. For example the previous scheme with the "author" and "publisher" added would look like the following:

```
<?xml version = "1.0" ?>
<root>
    <key value = "2082445" />
    <type value = "bib" />
    <dl name = "acm" />
    <author name = "Potlog, Alina-Diana" />
    <author name = "Xhafa, Fatos"/>
    <author name = "Pop, Florin" />
    <author name = "Cristea, Valentin" />
    <publisher name = IEEE Computer Society />
</root>
```

There will be no separate expansion schemes for just the "author" attribute or just the "publisher" attribute. The second enhancement of the system comes in the definition of rules for aggregation schemes. A former selection scheme looked like this:

```
<?xml version = "1.0" ?>
<root>
    <select>
        <author name ="Pop, Florin" />
        <publisher name=IEEE Computer Society />
    </select>
</root>
```

For this simple scheme we had to expand all entries to match it or not. Even the already expanded scheme would not actually match and an expansion from the main scheme was required.

Furthermore, for each attribute value a separate scheme needed to be created. With the RML tool all that is needed is to define 2 simple rules to match the attributes "author" and "publisher" and bind the key of the object such that it can be output if it matches.

```
<div class = "rule">
    <div class = "antecedent">
        <root>
            <key value = "rml-A">
            <author name ="Pop, Florin" />
            <publisher name=IEEE Computer Society />
        </root>
    </div>
    <div class = "consequence">
        <key value = "rml-A">
    </div>
</div>
```

This approach only constrains the application to have an agreement on the representation of both expansion schemes and subsequent aggregation schemes while allowing as many attributes as the user wants in the same expansion scheme regardless of the amount of selections, exclusions and aggregation that may be requested subsequently. This significantly reduces lookup time as all objects are ran through the same tool on the fly, as well as main memory space as the number of expansion schemes is reduced and the aggregation schemes are discarded as soon as the output is generated.

In this section we present the proposed model for the Distributed Data Aggregation Service that is implemented based on BlobSeer's features. We describe the view that our service has on 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 follow the steps taken by the DDAS to process each storage and retrieval request. The structured objects view, object storage model and request processing model were presented in [16].

3.1 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 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. 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.

3.2 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 catalogue for all these mappings known as the **object catalogue**. Furthermore it maintains a catalogue of all recurring aggregation schemes that are each mapped to a list of object meta information that expand to that scheme. This catalogue is known as the **metadata catalogue**.

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 catalogue is stored in the same BLOB. For applications that process very large objects in size but not in numbers, the object catalogue would not become a bottleneck due to the low number of keys, however this catalogue 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.

3.3 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-catalogue, 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 catalogue. The model for processing retrieval operations simply involves either retrieving the list inside the metadata catalogue 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.

4 Proposed Model

The newly designed flow model (see Figure 1) undergoes several modifications that integrate the RML tool in the system as a separate loosely coupled component. We also supress the role of Collect Gate as parsing of XML files and expanding them is no longer necessary, completely reducing the overhead of this step in the aggregation process. Like all other components in the system the RML tool will have to handle a large number of XML files to apply the transformation, therefore we need to efficiently isolate it's function when it comes to input and output. The only learning mechanism that the DDAS will now need to have is to map recurring rules to a list of unique identifiers such that common aggregation operation won't even need to go through the tool.

4.1 Data Back-end Storage System

The Data Backend Storage System is represented by BlobSeer, which was studied and motivated in our previous work for its excellent storage capabilities. It stores the "entry" field of an XML file and interacts only with the DDAS through standard read and write requests. For a read operation it accepts as input the serialized form of the "entry" field and returns the metadata (BLOB id, BLOB version, page, offset and size) associated to the storage location. For a write request, it accepts the metadata pointing to the object requested and outputs the serialized form of the object.

4.2 DDAS Overview

This component is the main result of our previous work. It represents the mediator component between BlobSeer and all other data-management applications that require fast and reliable storage and retrieval of data; moreover it also mediates communication with the RML tool. Its main role is to translate between BlobSeer unstructured view of data and the XML formatted objects that each application handles and our new tool processes. 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 RML tool to transform the schemes of the objects using the defined rule and obtain the identifiers of the objects that match. Its role is to use these identifiers to retrieve the metadata necessary for BlobSeer. Interaction with the RML tool is no longer required when performing a store operation.

The extension of the BlobSeer client writes data into BLOBS specified by the information in the metadata catalogue. 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 input aggregation scheme, the DDAS either identifies it in the metadata catalogue or if it does

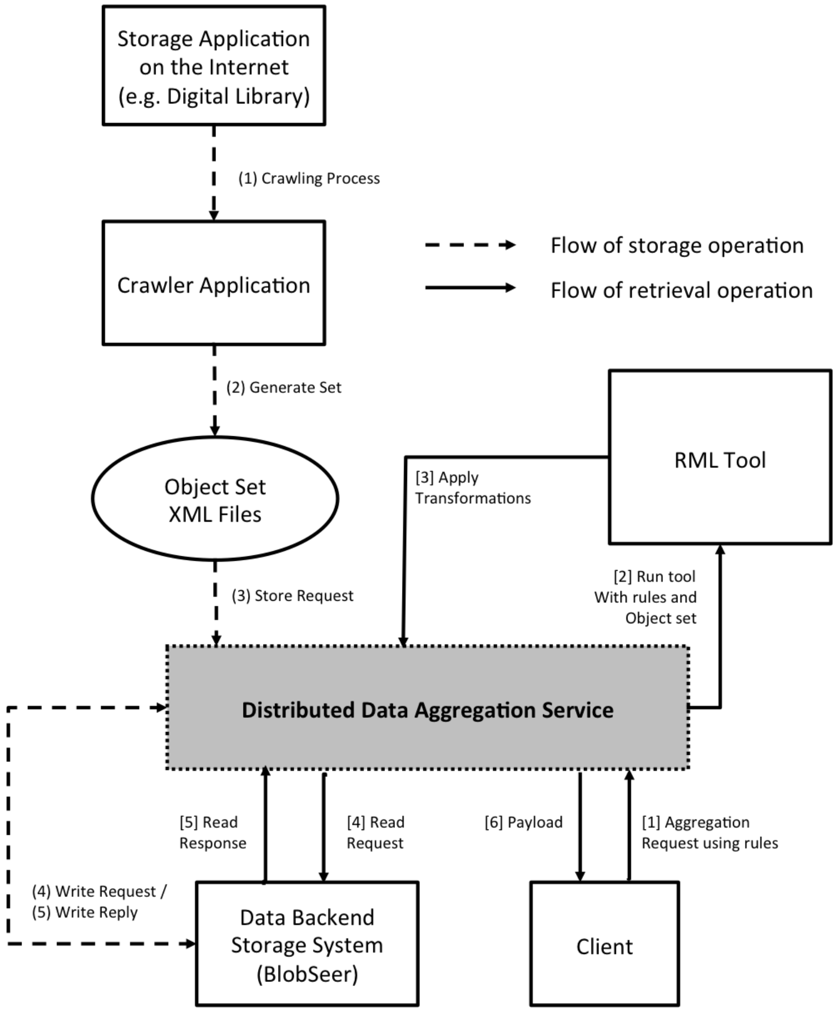
Storage Application on the Internet (e.g. Digital Library)
(1) Crawling Process
Crawler Application
Flow of storage operation
Flow of retrieval operation
(2) Generate Set
RML Tool
Object Set XML Files
[3] Apply Transformations
[2] Run tool With rules and Object set
(3) Store Request
Distributed Data Aggregation Service
[5] Read Response
[4] Read Request
[6] Payload
[1] Aggregation Request using rules
(4) Write Request / (5) Write Reply
Data Backend Storage System (BlobSeer)
Client

Fig. 1 Rule Analysis and Validation Flow for DDAS.

not exist, it will request from the RML tool the list of object keys that fit the aggregation scheme and then discard it, unless it recurrs past a threshold which will cause the DDAS to cache it. 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.

4.3 RML Tool used for Validation in DDAS

This is the novel addition to the architecture that mainly simplifies the aggregation process and surpresses the role of CollectGate. Because RML works with XML formats there is no need to parse the input anymore and the tool directly proceeds with the tranfsormation whose output will always be XML files containing the identifiers of the objects whose input files match the given rule. The role of this component is to perform quick searches and selections based on a few properties and offer a standard for defining rules and object properties that are easy to understand and follow. RML was designed to perform pattern matching in XML and therefore can do all aggregation operations on the fly without the need to do any matching when new objects are stored.

4.4 Rule Analysis and Validation Flow for DDAS

We now explain how objects are handled and flow through each of the components in Figure 1 during a request made by an application to store data or a client to 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. Each new object can be stored with an expansion scheme with all attributed that the application considers significant (without the need to create more than one) and directly written into BlobSeer. The expansion scheme in XML format, the object's unique identifier and its BlobSeer metadata are stored in the DDAS. These operations will no longer be as costly despite the large amount of data being handled, as there is no interaction with ant other component of the system.

An aggregation operation requires a client to input a rule defined in RML. Upon receiving this rule, the DDAS looks for the rule in its cache to possbily skip invoking the RML tool at all. If the rule is found then the list of matching identifiers is searchedin the object catalogue to find each object's metadata. The extended BlobSeer client will then read the objects based on this meta information. However, if the rule is not cached the it will be passed along with all the expansion schemes of the objects set stored to be processed by the RML tool. The tool will provide a list of XML files containing the identifiers of the objects that mtch the rule and based on these identifiers the interaction with BlobSeer will take place as previously stated. At the end of the request the read objects from BlobSeer will be provided to the user.

5 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.

5.1 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 catalogue and object catalogue. 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 catalogue 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 catalogue is represented by the same structure that maps schemes with lists of meta-information that represent the objects which fit the scheme.

5.2 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, the 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 of the 2011 Int.
                 Conference on P2P, Parallel, Grid, Cloud and
                 Internet Computing},
    year = {2011},
    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>
```

We tested DDAS with an aggregation scenario of objects with a large size domain ranging from 4KB to 1MB. We set the BlobSeer page size for all operation to 1KB as to minimize the amount of padded data that will be added to the BLOB. We obtained the results in Figure 5 for the transfer rate of the system. We can observe that regardless of the high number of objects, as long as the entry size is small there is no overhead in the processing of the aggregation rules, the only spikes resulting from network communication. The only overhead introduced in the system appears when object sizes grow past a threshold (256KB) and this is due to the page size being too small and implying a large number of reads and writes to BlobSeer, reducing the transfer rate proportionally to the increasing object size.

5.3 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 [5] across 20 virtual machines located at three different sites.

Figure 2 presents the performance times for aggregating objects of varying dimensions (from 1KB to 64MB) with different setups of the DDAS, that is a variation of the Page Size set in BlobSeer when writing and reading data. As objects size grows it is evident that an operation becomes more costly at a fine level of granularity. If the ratio between object and page size is large the time taken to complete an aggregation operation exponentially grows due to the overhead generated by a large number of requests for reading or writing on multiple small pages.

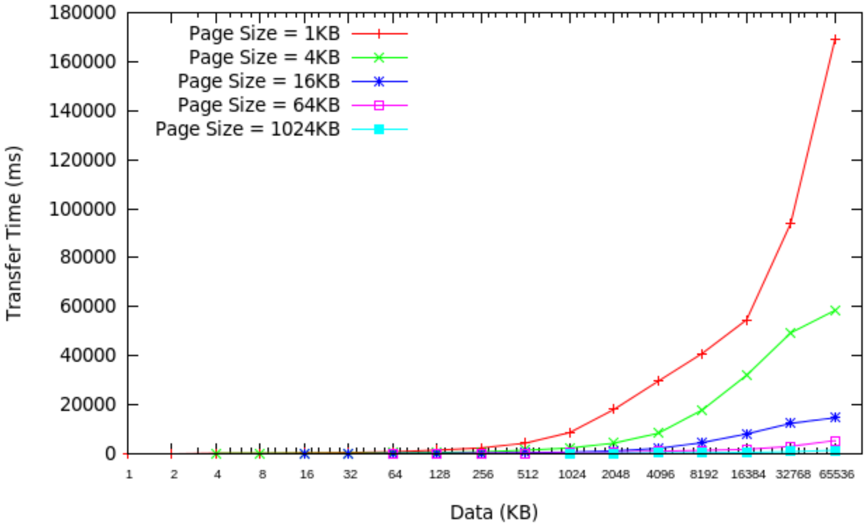

Fig. 2 Performance times for aggregating objects of varying dimensions (from 1KB to 64MB) with different setups of the DDAS.

This evaluation is underlined by the transfer rate measurements shown in Figure 3 for aggregating objects in the same size range. The results show a significant growth for a Page Size of 1MB, therefore it is suitable when aggregating large objects.

A more detailed comparison of Page Sizes is shown in Figure 4. We observe that the transfer rate for a Page Size set at 1KB stops growing significantly when objects go over 256KB. Similarly, for a Page Size set at 4KB the transfer

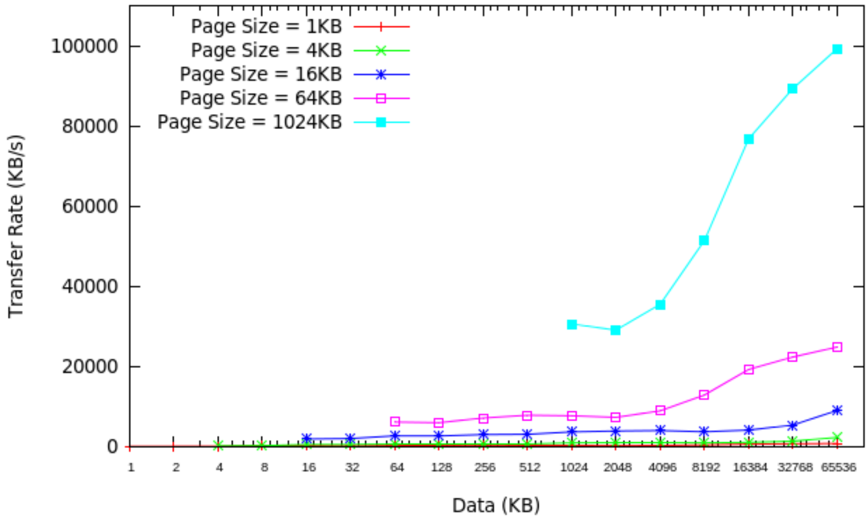

Fig. 3 Transfer rate for aggregating objects.

rate peaks around 2MB and 4MB, at which point it begins to decrease. Taking into account network anomalies we can generalize that for a ratio of up to 512 between Page Size and Object Size the fine granularity does not affect performance. After this peak in the ratio the Page Size inside BlobSeer must be dynamically adjusted to suit the objects sizes.

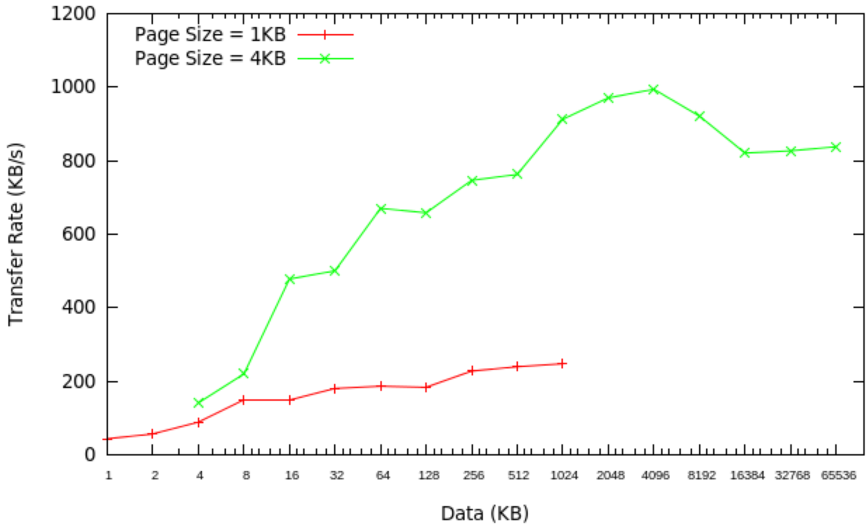

Fig. 4 Comparison of transfer rate for different values of Page Sizes.

These two measurements were performed in parallel, but we also wanted to investigate what happens when varying the objects size only slightly with the whole network dedicated to a client performing operations on a very fine granularity (Page Size = 1KB). The results in Figure 5 show that aside from a few errors due to the network the transfer rate is affected as soon as objects grow past 256KB, even when size grows at a minimal rate.

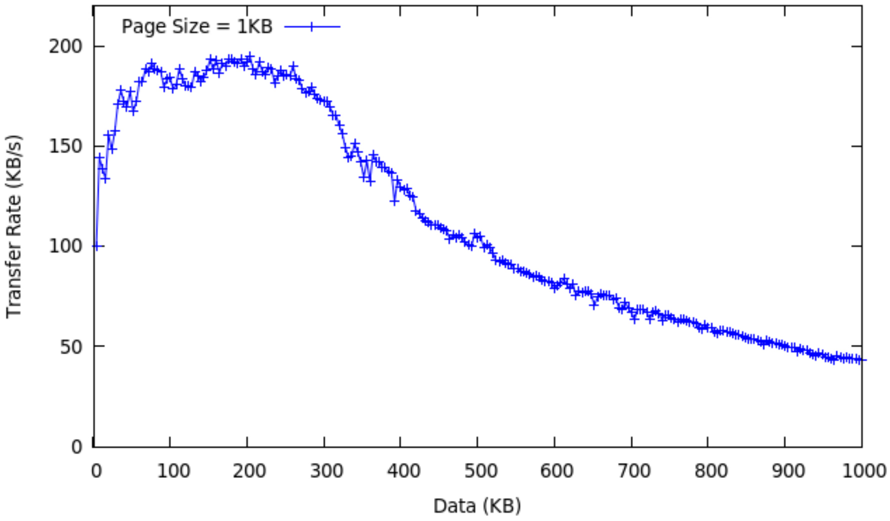

Fig. 5 Transfer rate evolution for Page Size = 1KB. The transfer rate is affected as soon as objects grow past 256KB.

After making a thorough analysis of the previous results we decided to investigate how consistent the object:page ratio is and determine the range of object sizes for which a certain page size is suitable. The results are presented in Figure 6. For example a Page Size of 1KB is suitable for objects up to 64KB. For objects between 64KB and 512KB a Page Size between 4 and 16 KB yields good performance times, while a Page Size of 64KB is suitable for objects up to 4MB. Finally we can conclude that for very large objects a Page Size of 1MB is the best setup in BlobSeer. A very important observation is that it is not sufficient to select a large Page Size regardless of the object size due to the fact that the Page Size determines the minimum amount of memory that will be used by one aggregation operation.

We varied both the actual data size and the page size of the blobs and obtained the results in Figure 6. 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

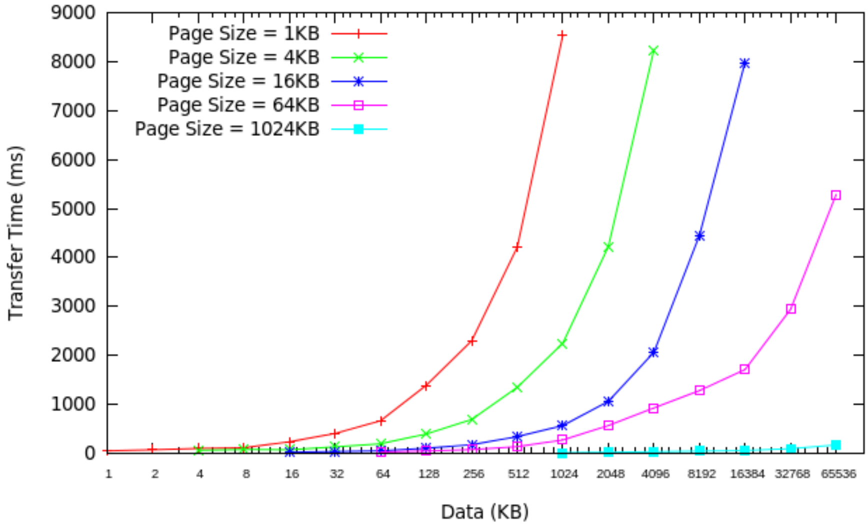

Fig. 6 Determination the range of object sizes for which a certain page size is suitable.

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.

5.4 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 [17]. 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 Table 1 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 overhead of generating new schemes on the fly is minimal while eliminating the need to store schemes in temporary or permanent memory. Therefore the RML tool needs to be replicated only to maintain reliability of the service, and does not need to be scaled in terms of resource provisioning. Furthermore, because the schemes and rules now have a standard XML format, the DDAS is suitable for working in a federated cloud environment. The ag-

Table 1 Performance times of DDAS operations

Number of concurrent requests for 100K objects	Read requests times (ms)	Write requests times (ms)
1	26	91
10	110	137
100	831	917
1000	7521	7550
10000	93046	97165

gregation schemes output by the RML tool and sent as queries to the storage layer can easily be formatted as context variables when requesting data from a resource deployed on a different cloud. The RML tool is now the top layer of the DDAS and is responsible for handling the input rules and generating the keys which are bound to those rules and passing them to the metadata management layer. This layer matches the keys with the metadata used for identifying the objects in BlobSeer. Finally the BlobSeer layer retrieves the serialized data from the physical layer.

6 Conclusions

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. At the same time validation of results that can also reach Exascale dimension is required In this paper we proposed an extension to the Distributed Data Aggregation Service (DDAS) previously proposed using a formal method for analysing and validating the aggregation process. We presented the new model of the DDAS in order to ensure a high level of performance in all aspects of a data storage and aggregation as well as representation of object properties. Also we emphasized again on 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 was supported by projects: "*SideDOWN: Smart Internet Data Downloader and Aggregator*", ID: PN-II-IN-CI-2012-1-0324; *CyberWater* grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, project number 47/2012; *MobiWay*: Mobility Beyond Individualism: an Integrated Platform for Intelligent Transportation Systems of Tomorrow - PN-II-PT-PCCA-2013-4-0321; *clueFarm*: Information system based on cloud services accessible through mobile devices, to increase product quality and business development farms - PN-II-PT-PCCA-2013-4-0870.

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 (`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*, 3:S08002, 2008.
2. M. M. Sufyan Beg and Nesar Ahmad. Soft computing techniques for rank aggregation on the world wide web. *World Wide Web*, 6(1):5–22, March 2003.
3. Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: dependable and secure storage in a cloud-of-clouds. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 31–46, New York, NY, USA, 2011. ACM.
4. Andrew Brampton, Andrew MacQuire, Idris A. Rai, Nicholas J. P. Race, and Laurent Mathy. Stealth distributed hash table: a robust and flexible super-peered dht. In *Proceedings of the 2006 ACM CoNEXT conference*, CoNEXT '06, pages 19:1–19:12, New York, NY, USA, 2006. ACM.
5. 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*, GRID '05, pages 99–106, Washington, DC, USA, 2005. IEEE Computer Society.
6. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26:4:1–4:26, June 2008.
7. Jin Chen, Saba Sehrish, Wei-Kang Liao, Alok Choudhary, and Karen Schuchardt. Improving the average response time in collective i/o. In *Recent Advances in the Message Passing Interface*, LNCS 6090, pages 71–73, 2011.
8. Tristan Glatard, Johan Montagnat, and Xavier Pennec. Efficient services composition for grid-enabled data-intensive applications. In *Proceedings of the IEEE International Symposium on High Performance and Distributed Computing*, pages 333–334, June 2006.
9. Waldemar Hummer, Philipp Leitner, and Schahram Dustdar. Ws-aggregation: distributed aggregation of web services data. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 1590–1597, New York, NY, USA, 2011. ACM.
10. Joost Jacob. A rule markup language and its application to uml. In *Leveraging Applications of Formal Methods*, pages 26–41. Springer, 2006.
11. Elis Kulla, Evjola Spaho, Fatos Xhafa, Leonard Barolli, and Makoto Takizawa. Using data replication for improving qos in manets. In *Proceedings of the 2012 Seventh International Conference on Broadband, Wireless Computing, Communication and Applications*, BWCCA '12, pages 529–533, Washington, DC, USA, 2012. IEEE Computer Society.

12. Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44:35–40, April 2010.
13. Jae Kyu Lee and Mye M. Sohn. The extensible rule markup language. *Commun. ACM*, 46(5):59–64, May 2003.
14. Bogdan Nicolae, Gabriel Antoniu, Luc Bougé, Diana Moise, and Alexandra Carpen-Amarie. Blobseer: Next-generation data management for large scale infrastructures. *J. Parallel Distrib. Comput.*, 71:169–184, February 2011.
15. Mayur R. Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. Amazon s3 for science grids: a viable solution? In *Proceedings of the 2008 international workshop on Data-aware distributed computing*, DADC '08, pages 55–64, New York, NY, USA, 2008. ACM.
16. Vlad Serbanescu, Florin Pop, Valentin Cristea, and Gabriel Antoniu. Architecture of distributed data aggregation service. In *Proceedings of the 2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, AINA '14, pages 727–734, Washington, DC, USA, 2014. IEEE Computer Society.
17. Shaoxu Song and Lei Chen. Indexing dataspaces with partitions. *World Wide Web*, 16(2):141–170, March 2013.
18. A. Stam, J. Jacob, F.S. de Boer, M.M. Bonsangue, and L. van der Torre. Using xml transformations for enterprise architectures. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods*, volume 4313 of *Lecture Notes in Computer Science*, pages 42–56. Springer Berlin Heidelberg, 2006.
19. Dorian Gorgan, Victor Bacu, Denisa Rodila, Florin Pop, and Dana Petcu. (2010). Experiments on ESIPâĂŤEnvironment oriented satellite data processing platform. Earth Science Informatics, 3(4), 297-308.
20. Florin Pop, Claudiu Gruia, and Valentin Cristea. (2007, July). Distributed algorithm for change detection in satellite images for Grid Environments. In Parallel and Distributed Computing, 2007. ISPDC'07. Sixth International Symposium on (pp. 41-41). IEEE.
21. Srikumar Venugopal, Rajkumar Buyya, and Kotagiri Ramamohanarao. A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Comput. Surv.*, 38, June 2006.
22. Fatos Xhafa, Vladi Kolici, Alina-Diana Potlog, Evjola Spaho, Leonard Barolli, and Makoto Takizawa. Data replication in p2p collaborative systems. In *Proceedings of the 2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 3PGCIC '12, pages 49–57, Washington, DC, USA, 2012. IEEE Computer Society.
23. Yuan Yu, Pradeep Kumar Gunda, and Michael Isard. Distributed aggregation for data-parallel computing: interfaces and implementations. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 247–260, New York, NY, USA, 2009. ACM.
24. Ji Zhang, Xiaohui Tao, and Hua Wang. Outlier detection from large distributed databases. *World Wide Web*, 17(4):539–568, July 2014.