Mahasen: Distributed Storage Resource Broker

K. Perera, T. Kishanthan, H. Perera, D. Madola, Malaka Walpola, Srinath Perera

Mahasen: Distributed Storage Resource Broker

K.D.A.K.S.Perera[1], T Kishanthan[1], H.A.S.Perera[1], D.T.H.V.Madola[1], Malaka Walpola[1], Srinath Perera[2]

[1] Computer Science and Engineering Department, University Of Moratuwa, Sri Lanka.
{shelanrc, kshanth2101, ashansa.perera, hirunimadola, malaka.uom}@gmail.com
[2] WSO2 Lanka, No 59, Flower Road, Colombo 07, Sri Lanka
srinath@wso2.com

Abstract. Modern day systems are facing an avalanche of data, and they are being forced to handle more and more data intensive use cases. These data comes in many forms and shapes: Sensors (RFID, Near Field Communication, Weather Sensors), transaction logs, Web, social networks etc. As an example, weather sensors across the world generate a large amount of data throughout the year. Handling these and similar data require scalable, efficient, reliable and very large storages with support for efficient metadata based searching. This paper present Mahasen, a highly scalable storage for high volume data intensive applications built on top of a peer-to-peer layer. In addition to scalable storage, Mahasen also supports efficient searching, built on top of the Distributed Hash table (DHT)

1 Introduction

Currently United States collects weather data from many sources like Doppler readers deployed across the country, aircrafts, mobile towers and Balloons etc. These sensors keep generating a sizable amount of data. Processing them efficiently as needed is pushing our understanding about large-scale data processing to its limits.

Among many challenges data poses, a prominent one is storing the data and indexing them so that scientist and researchers can come and ask for specific type of data collected at a given time and in a given region. For example, a scientist may want to search for all Automated Weather data items collected in Bloomington area in June 15 between 8am-12pm.

Although we have presented meteorology as an example, there are many similar use cases. For instance, Sky server [1] is one of the best examples that illustrate the use case of large data generation. This project expects to collect 40 terabytes of data in five years. In its data collection, the photometric catalog is expected to contain about 500 distinct attributes for each of one hundred million galaxies, one hundred million stars, and one million quasars. Similarly many sciences, analytic processing organizations, data mining use cases etc., would want to store large amount of data and process them later in a selective manner. These systems often store data as files

and there have been several efforts to build large scale Metadata catalogs [2][3] and storage solutions[4][5] to support storing and searching those data items. One such example is AMGA metadata catalog [6] which was an effort to build replication and distribution mechanism for metadata catalogs.

As we discuss in the related work section, most of the metadata catalog implementations use centralized architectures and therefore have limited scalability unlike Mahasen. For example, Nirvana Storage [7] has a centralized metadata catalog which only supports scalability through vendor's mechanism such as Oracle Real Application clusters. XML Metadata Concept catalog (XMC Cat) [8] is another centralized metadata catalog which stores hierarchical rich metadata. This paper presents Mahasen, a scalable metadata catalog and storage server built on top of a P2P technology. Further, it is built by distributing an open source centralized Data registry (WSO2 Registry).

Mahasen (Distributed Storage Resource Broker) is a Data Grid Management System (DGMS) that can manage a large volume of distributed data. It targets high volume data intensive applications. The architecture of Mahasen has been designed to present a single global logical namespace across all the stored data, and it maintains a metadata structure which can be used to search files based on its' attributes. It is a network of storage servers that plays the dual purpose of a metadata catalog and a storage server. Mahasen will solve the huge data storage problem and fault tolerance in data intensive computing through aggregating low cost hardware while having both metadata and actual resources distributed without single point of failure. Metadata management will ensure the capability of searching files based on attributes of the stored resources. Mahasen has a metadata catalog, which is highly distributed and well scalable. The metadata layer ensures fault tolerance by keeping replicas of metadata.
The rest of the paper is organized as follows. The next section will discuss the related work in Metadata catalogs and Storage servers while comparing and contrasting them with Mahasen. The following section will discuss Mahasen architecture. The next section will present the performance evaluation of Mahasen. Finally the discussion section discusses limitations, other potential solutions and directions.

2 Related Work

2.1 Nirvana Storage

Nirvana SRB [7] is a middleware system that federates large heterogeneous data resources distributed across a network. The ability to access, manage, search and organize data across the entire SRB Federation is provided via a Global Namespace. MCAT is the centralized metadata repository which maintains two types of records – system- and user-metadata. Scalability of MCAT is achieved using database vendor's mechanisms [9], hence limited by Relational DB scalability Limits.

Storage/Replication. The stored resources are divided as Physical resources, Logical resources and Cluster resources. Replication of resources across multiple servers ensures the availability and recoverability of resources during failovers.

Retrieve. Data stream routing is handled by SRB and TCP/IP, making the data transfer process transparent to the users..

Search. Searching is done based on metadata attributes which are extracted and managed by the SRB.

Add/Update. Data can be added in two ways: Registration and Ingestion. Registration does not transfer any data but only creates a pointer to the data in MCAT. Ingestion is similar to registration but also transfers the data to an SRB storage resource.

Delete. If a file shadow object is used as a data object to ingest a file resource to SRB then file will be removed from MCAT but not from the physical location.

2.2 Apache OODT

OODT[10] is a middleware system for metadata that provides transparent access to the resources. It facilitates functionalities such as store, retrieve, search and analyze distributed data, objects and databases jointly. OODT provides a product service and profile service which manage data and metadata respectively.

Storage/Replication. OODT stores data product in a file-based storage in a distributed manner. They classify storage into three categories: on-line, near-line or off-line storage.

Retrieve. When OODT receives a request for retrieving a file, it issues a profile query to a product server that helps in resolving resources that could provide data. The response will include the target product server address in the form of a URI. The OODT issues a product query based on the profile query results to get the data, and it will actually retrieve data from the product server in a MIME-compliant format.

Search. OODT uses the profile server and the product server for searching the metadata and retrieve the products, and it has multiple of each type of server. OODT is based on client server architecture and it promotes REST-style architectural pattern for search and retrieve data. The profile or a subset of profile is returned for retrieval.

Add/Update. OODT provide data management including manage files and folders with the implementation of javax.sql.datasource interface.

Delete. The file management component of a Catalog and Archive Service support the delete of resource files and metadata through the implementation of javax.sql.datasource interface.

2.3 WSO2 Governance Registry

WSO2 Governance Registry [11] is a repository that allows users to store resources in a tree-structured manner, just like with a file system. However, unlike a file system, users may annotate resources using their custom properties, and also WSO2 Registry has built in metadata management features like tagging, associating resources.

However, WSO2 registry is backed by a Relational Database system, and it uses database features to store data, metadata, to manage them, and to search. Hence it has a centralized architecture. Mahasen extends that architecture to a distributed architecture.

Replication. There is no inbuilt mechanism to do the replication of resources in WSO2 registry.

Search. The WSO2 registry provides two types of searches. One is searching for a resource with their name, metadata etc., and it is implemented using underline relational database system. The second one is searching the content of resources, and implemented using Lucene [12]. The second search is only applicable to resources with textual content.

Add/Update. Adding of resources to registry can be done in two ways. First one is adding via the web interface provided by the registry. When adding a new resource, it is also possible to add additional metadata such as tags, properties of name value pairs, which later will be useful to search for that resource. The other way to add resources is by writing your own way by extending the registry API and exposing it as a web service.

The major limitation with registry, when storing resources, is the amount of memory available. Since it uses the java heap memory to buffer the resources before storing them, large files cannot be stored as the available memory is only limited to few hundred of megabytes.

2.4 Hadoop Distributed File System

Apache Hadoop Distributed File System is (HDFS)[13] is a file system designed to run on commodity hardware. HDFS has a master slave architecture that consists of a single NameNode as master and number of DataNodes. The NameNode is responsible of regulating access to files by client and managing the namespace of the file system. Generally DataNodes are deployed one per node in the cluster, and is responsible of managing storage attached to that node.

Storage / Replication. Hadoop supports hierarchical file organization where user can create directories and store files. It splits the file in to chunks with the default size of 64MB and stores them as sequence of blocks, and those blocks are stored in underlying file system of DataNodes. Those blocks are replicated for fault tolerance and the block size and the replication factor of data are configurable.

Retrieve. Applications that run on HDFS need streaming access to their data sets. Data nodes will be responsible for the read requests that issued from a user to retrieve data from the system.

Search. Hadoop Distributed File System does not provide a comprehensive search for users or applications, and it just fulfill the requirement of a distributed file system by supporting to locate the physical location of the file using the system specific metadata.

Add/Update. Writing to HDFS should be done by creating a new file and writing data to it. Hadoop addresses a single writer multiple readers' model. Once the data is written and file is closed, one cannot remove or alter data. Data can be added to the file by reopening the file and appending new data.

Delete. When a file is deleted by a user or from an application, the particular resource is not immediately removed from HDFS. The resource will be renamed and copied in to /trash directory giving the possibility to restore as long as it remains in the trash.

Mahasen's main differentiation from above systems comes from its scalability. It can scale significantly than Nirvana Storage that depends on relational databases to scale the system, since the Mahasen metadata layer is natively distributed using a DHT.WSO2 Registry provides the clustering as the scalability option, but it is not optimized for large file transfers and storing as it uses an ATOM based resource transfers. Furthermore, Mahasen provides users a comprehensive metadata model for managing the distributed resources they stored with user-defined metadata, unlike the HDFS, which only focuses on creating a Distributed file system. Further Mahasen's metadata layer is natively distributed and fault tolerant while HDFS has a single name node which can make fault tolerant only with an active passive failover configuration.

3 High Level Architecture

3.1 Mahasen High Level Architecture

As shown by Figure 1, Mahasen consists of several storage nodes which are connected as peers to a logical ring via FreePastry. Each node consists of a registry to store

metadata and a file system to store physical file parts. Once connected to the ring each node contributes to the metadata space as well as file storage capacity, scaling the system dynamically with new node additions. Nodes use underline DHT (FreePastry) routing protocol to communicate efficiently with each other.

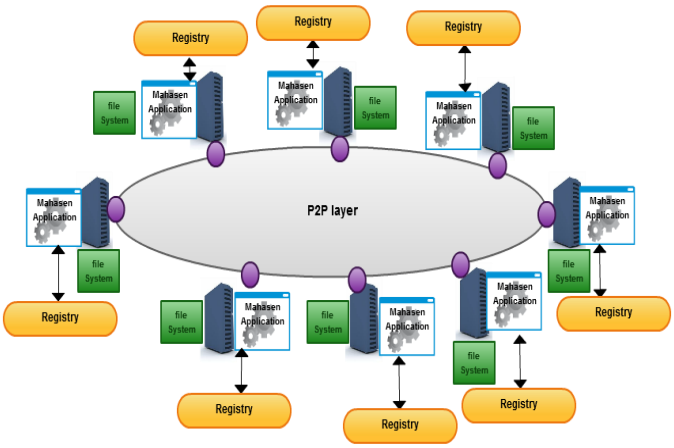

Fig. 1. Mahasen High Level Architecture

Mahasen uses a WSO2 registry and the file system in each node and DHT based architecture is used to connect the nodes to a one unit.

Mahasen has a distributed metadata layer that stores data about the distributed files in Mahasen peer to peer network. The metadata catalog is used to broker the stored resources in the network and to assist the user to locate the files in Mahasen distributed environment abstracting the metadata management from the user.

Mahasen stores two main types of metadata, which are system-defined metadata and user-defined (descriptive) metadata. System defined metadata is mainly used for server side resource handling. File name, file size, stored node IPs of file are examples of the system-defined metadata. User defined metadata is used to provide users the searching capability on those metadata. User can add tags and properties (name, value pairs) to the files that are uploaded.

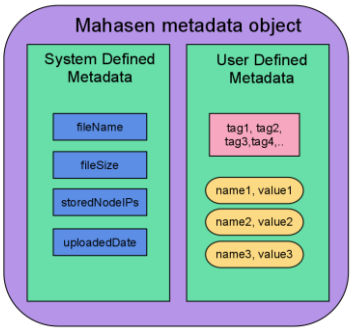

Fig. 2. Metadata Object Structure of Mahasen

When a file is uploaded connecting to a Mahasen node the file will be temporarily saved in that node. Then the node will act as the master node and split the file into pre-defined sized chunks and the split parts are stored in a selected set of the neighborhood nodes of master node through parallel transfer. Then the metadata object created by master node will be stored with replicas using PAST storage implementation of Free pastry. We have rewritten PAST node's persistent storage such that the data will be stored in the WSO registry in that node.

After storing the metadata, the nodes that received file parts act as worker nodes and replicate their file parts in parallel according to the replicate request issued by the master node. Each worker node will update the metadata object with stored locations of the file parts which were replicated after replicating their file parts using the capability of concurrent access to metadata objects, and Mahasen handles them using the locking system provided by the lock manager of DHT.

User can request to download a file from any Mahasen node and the node will first generate the resource ID for the requested and retrieve the metadata object. Then it extracts the locations of Mahasen nodes that contain the file parts from the metadata object and retrieve those parts to the local machine. The parts will be merged to create the original file after retrieving all the parts and the file will be streamed to the user.

Deletion can be performed with a single command across a heterogeneous storage system. When a delete request for a file is issued, by following the same method of retrieving the file, Mahasen finds nodes that store parts of the file and deletes them. Finally the metadata object will also be deleted with replicas

When user needs to update the user-defined metadata, the node that receives the update request retrieves the metadata object for the file from the DHT, updates it, and stores it back in the DHT.

. Using this model, Mahasen has built a complete decentralized metadata system that handles metadata management in a highly scalable and efficient manner.

Mahasen keeps replicas of both actual files and metadata objects. The main purpose of keeping replicas is for fault tolerance and failover recovery. We ensure the high availability of metadata while ensuring the scalability using free pastry's underlying DHT.

3.2 Mahasen Search

When the amount of data in the system grows, the complexity of the search increases. Mahasen builds a distributed data structure using the underlying DHT, which can improve the performance of different search options that Mahasen supports.

The resources in Mahasen are associated with metadata and for each tag or property in system, we maintain an index pointing to all resources which have that tag or property. This is implemented as a TreeMap [16] and the property trees are stored in the DHT which handles replicas of it.

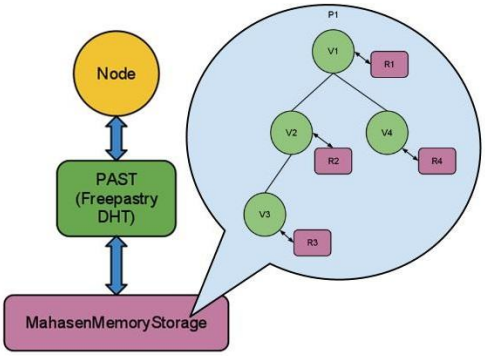

Fig. 3. A Property Tree Stored in Mahasen Memory Storage

When a user sends a search request, Mahasen extracts the requested search and initiate the execution of relevant search method. Then the resource IDs of the files which match with the given input are retrieved from the relevant property tree. Extracting the relevant resource IDs are done as follow.

Users can send search requests to any Mahasen node, and when a node receives a search request, Mahasen takes the property name given by the client and generates the property tree ID for that property. If the current node has the index for the property, it receives matching resource IDs for that property and sends them to the client. If not, the node acts as a master node and gets the node handles of the nodes which are having the specific property tree and routs Mahasen search messages with the required parameters to the node handles. Then those node handles will get the relevant resource IDs from the property trees in their memory storage and send back to the master node.

The property values in the property tree are sorted, so that if the search is a range based search, we can simply take the sub map between the initial and final property values and retrieve the set of resource IDs mapped to each of the node in the sub tree. Since these resource IDs represents the files having the given property values, Mahasen can look up for the metadata objects with those resource IDs and extract the file names to present to for the user. The operation of extracting the file names for the resource IDs has a high cost than extracting the matching resource IDs for the given search query.

Complete Data Structure built for Mahasen can support property based search, range based search, tag based search and Boolean operations for the properties such as AND operation and OR operation. The advanced search provided by Mahasen is capable of providing the search based on set of different properties and tags.

Mahasen Search utilizes the continuation model support by FreePastry in results retrieving and transferring. Therefore when a search request is issued, the application sends requests to look up node handles, which contain the particular TreeMap object to request results. Then the application will collect the first result incoming and resume action from the previous execution point.

3.3 File Handling

File Transfer. Mahasen is a network of storage nodes and users will be given a client which is the Mahasen Client to access and transfer files to the network. The Mahasen Client that is built using the Apache HttpClient [17] uses HTTP methods for transferring files to the network. First the client initiates a connection with one of the node in the network. An authenticated client is capable of uploading downloading, deleting, updating or searching for the files in the network. The File content will be added as an entity to the HTTP POST method and streamed to the target address. The receiving end will read the file stream and write it to the repository.

Replica Management. To achieve fault tolerance and failover recovery, the file will be split into a set of predefined chunks and each part will be replicated and stored in different nodes according to predefined replication factor. The placement of replicas is a critical part which affects the reliability and performance of the system. The purpose of having a policy for placement of replicas is for data reliability, availability, and network bandwidth utilization. The current policy of Mahasen is to store the replicated files in leaf nodes set to the initial node. The selection of nodes in the leaf set will be calculated using cost evaluation function which focus on the distance of the node.

After successfully transferring the file to the initial node, the client will be notified about the status of the file transfer and initial node will then replicate and transfer the file to other nodes. The number of copies kept for a file is called the replication factor of that file and will be decided by the Mahasen system.

File Splitting and Parallel transfer. Mahasen storage network is designed to store large files reliably across distributed nodes. When storing the file it will be split into blocks of fixed size and these blocks will be replicated across the network for fault tolerance. The transferring of replicated file blocks will be done in parallel to other nodes in order to utilize the bandwidth and to save time.

When focusing on the retrieval of a file by using the metadata object the system will then select a node which is closest to the reader node and download the blocks to the client. Downloading of file blocks will also be done in parallel and then the blocks will be merged to create the complete file.

3.4 Mahasen API

Mahasen provides a complete API to perform CRUD operations and search. Users can develop external clients apart from the default client Mahasen provides and integrate with existing systems to perform resource management and search operations.

3 Performance Analysis

The Mahasen System Scalability was tested by running a system with M nodes and N parallel clients. Here the value for M was 1, 6, 12, 18, 24 and N was 1, 5, 10, 15, 20. Each client carried out upload, download, delete and search operations for 10 times and the average was taken. The system configuration that was used in this test are, Two machines with Intel(R) Xeon(R) CPU E5-2403 1.80GHz 4 Core machines having 24GB RAM and One machine with Intel(R) Xeon(R) CPU E5-2470 2.30GHz 8 Core machines having 63GB RAM. Following Figures (from 4 to 7) depicts the results of this test. In the upload test, 500MB size files were used by each client. .

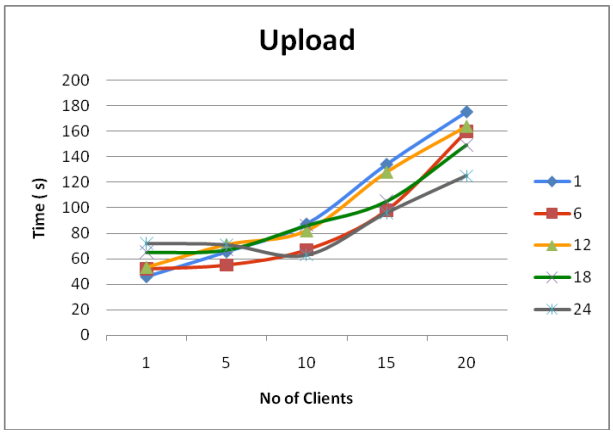

Fig. 4. Upload test results

In the results it is observed that when the number of client increases, the upload time is also increasing. We believe that this is due to the network congestion and background processes of data replication across nodes. When the number of nodes increased to 18 or 24, a reduction in upload time were observed. This was an expected behaviour, because the node which client selects to upload, distributes replica management task for other nodes in the p2p ring.

Fig. 5. Download test results

When download files using Mahasen client, it is observed that with the increase of number of client, the single node setup has a significant growth in the download time. In the performance test, a single node was chosen to send the client request while it coordinates the file transfer from other nodes in the setup. Therefore when there are multiple nodes in the system you can download file parts from other available nodes, which reduces the download time.

Fig. 6. Delete test results

When Mahasen performs a Delete on a resource, it involves 3 operations such as deleting metadata, deleting entries from search index, and deleting the physical file. When more nodes are in the system, each node can participate in deleting its own files in parallel, making the system more scalable and efficient.

Fig 7. Search test results

Search results illustrate that Mahasen can perform well even with more nodes added to the system. Usually single node should have the lowest possible time as it does not have to search across the p2p ring. But with multiple nodes, it has to aggregate results and present it to the client. This can be observed from the figure that, when more clients are in the system, results tend to converge into a lower value due to caching as we requested search operation through the same node.

3 Discussion and future work

Mahasen provides a highly scalable metadata structure with its peer-to-peer architecture in the metadata catalog. Unlike the existing metadata catalogs that use centralized architecture, Mahasen distributes metadata across the nodes in the system with the replication making the overall system scalable and fault tolerant.

Mahasen keeps replicas of both metadata objects and property trees as well. The DHT of FreePastry is used to store these objects in the system which provides easy access of them. Keeping replicas of metadata objects and property tree objects do not cost as much as keeping replicas of actual files which are very large in size compared to metadata and property tree objects. By having these objects with replicas in the system, Mahasen has been able to ensure the correct functioning of many of the Mahasen operations even in the conditions like node failures.

An important contribution of Mahasen is developing a distributed indexing structure on top of the DHT for searching data products using different properties associated with data products. Since Mahasen needed to support range based queries, we evaluated earlier effort to build such index structures. Skip Tree Graph [18] was one of the best candidates we selected for search assisting data structure, which can efficiently support range based queries over a DHT. Since we had different properties and data structure had to grow in two dimensions, one in number of properties and the other one in number of entries for one property we were forced to create different DHTs for different properties. Therefore we needed to evaluate a much less complex

solution since maintaining different DHTs could have been very expensive in terms of resources.

When the system scales up with the large number of nodes, it will be more costly to issue a search operation on the available raw metadata stored. Therefore Mahasen developed a combined data structure with DHT and TreeMap as explained earlier.

When a Mahasen node fails, and it is detected by the existing nodes in the network, Mahasen replicates all the metadata objects and the property tree objects which were in the failed node to the existing Mahasen node reading them from other replicas. Mahasen helps in preserving the availability of metadata objects and property tree objects by maintaining the replication factor of them a constant.

Current Mahasen design has several limitations, which we plan to handle as future works. Currently Mahasen stores each property indexes in one Mahasen node and assumes that it will fit within the memory of that node. This may not be major concern for simple cases, and even NoSQL storages like Cassandra makes similar assumptions. Dividing the property tree into parts and storing them in different nodes when it is larger than a given size can solve this problem. We can predefine the maximum size of a part that will be residing in one node.

Another challenge is that search based multiple properties where at least one is a common property would force Mahasen to join large data sets, and one potential solution is to negotiate the size of data sets before start the data merging.

To summarize, Mahasen project builds a scalable storage solution by making a group of existing open source registries work as a one unit. It provides a one logical global namespace, and users may talk to any node of the group and perform any operations.

Mahasen connects nodes (registries) using PAST, a storage overlay implemented on top of Pastry DHT algorithm. Furthermore, Mahasen builds a distributed indexing structure on top of DHT to support property-based search of data items.

A user can benefit from the Web Service API provided and effectively utilize for batch processing of file uploading task through a custom client or basic client provided by Mahasen.

References

1. Alexander, S., Szalay, Peter, Z., Kunszt, Ani Thakar, Jim Gray, Don Slutz, and Robert, J., Brunner.: Designing and Mining Multi-Terabyte Astronomy Archives.: The Sloan Digital Sky Survey. In: SIGMOD ’00 Proceedings of the 2000 ACM SIGMOD international conference on Management of data (2000)
2. Chaitanya Baru, Reagan Moore, Arcot Rajasekar, Michael Wan.:The SDSC Storage Resource Broker (1998)
3. Reagan, W., Moore.: Managing Large Distributed Data Sets using the Storage Resource Broker (2010)
4. G., DeCandia, D., Hastorun, and M., Jampani.: Dynamo.: Amazon’s Highly Available Key-value Store (2010)
5. Ghemawat, S.-T., Leun, and H., Gobioff.: The Google File System.
6. B., K., Nuno Santos.: Distributed Metadata with the AMGA Metadata Catalog.

7. Nirvana Storage - Home of the Storage Resource Broker (SRB®), http://www.nirvanastorage.com/index.php?module=htmlpages&func=display&pid=1 (2011)
8. XML Metadata Concept Catalog (XMC Cat), Data to Insight Center, Indiana University Pervasive Technology Institute, http://d2i.indiana.edu/xmccat.
9. Nirvana Performance, http://www.nirvanastorage.com/index.php?module=htmlpages&func=display&pid=54.
10. ApacheTM OODT, http://oodt.apache.org/ (2011)
11. WSO2 Governance Registry - lean.enterprise.middleware - open source SOA | WSO2, http://wso2.com/products/governance-registry/ (2011)
12. Apache Lucene - Overview, http://lucene.apache.org/java/docs/index.html.
13. HDFS Architecture Guide, http://hadoop.apache.org/docs/r1.0.4/hdfs_design.html (2011)
14. Pastry - A scalable, decentralized, self-organizing and fault-tolerant substrate for peer-to-peer applications, http://www.freepastry.org/.
15. P., Druschel and A., Rowstron.: PAST: A large-scale, persistent peer-to-peer storage utility. In: HotOS VIII, Schoss Elmau, Germany (2001)
16. TreeMap (Java 2 Platform SE 5.0), http://download.oracle.com/javase/1.5.0/docs/api/java/util/TreeMap.html (2011)
17. HttpClient - HttpComponents HttpClient Overview, http://hc.apache.org/httpcomponents-client-ga/ (2011)
18. Alejandra Gonz´alez Beltr´an, Paul Sage and Peter Milligan.: Skip Tree Graph: a Distributed and Balanced Search Tree for Peer-to-Peer Networks.