# Mahasen: Distributed Storage Resource Broker

K.D.A.K.S.Perera, T.Kishanthan, H.A.S.Perera, D.T.H.V.Madola
Department of Computer Science and Engineering
University of Moratuwa
Moratuwa, Sri Lanka.

*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 come from 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 and reliable 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 searching built on top of the Distributed Hashtable (DHT).**

***Keywords-component; Metadata, Distributed Computing, Broker, DHT, P2P, Replication, Datta Intensive***

## I. 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 and 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 ADAS 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 processes 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 shall discuss in the related work section, most of the metadata catalog implementations uses centralized architectures and therefore have limited scalability. For example, Nirvana Storage[7] has a centralized metadata catalog which only support scalability through vendor's mechanism such as Oracle Real Application clusters. Also, 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 in data intensive computing through aggregating low cost hardware. 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.

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 other directions.

## II. Related Work

### A. *Nirvana Storage*

Nirvana SRB [7] is a middleware system that federates heterogeneous data resources. It provides a uniform interface to discover, access, share and manage a large volume of resources such as file systems, archives, databases, which are distributed across a network. Storage Resource Broker (SRB) offers a Global Name-space to access, manage, search, and organize data across the entire SRB Federation.

SRB's architecture is based on relational database technology and distributed servers. It has a centralized metadata catalog which is named as MCAT. Scalability of the metadata catalog is achieved using database vendor's mechanisms [9], hence limited by Relational DB scalability Limits.

*1) Storage /Replication:* Nirvana supports all major storage systems, file systems, operating systems, relational databases, etc. It keeps data replicas to maintain availability and recoverability. High availability and disaster recovery is maintained with definable number of data replicas in distributed locations. The storage resources are divided into three categories. They are Physical resources, Logical resources and Cluster resources.

*2) Retrieve:* Data is retrieved as the way it is loaded. User is transparent from where the data is coming since SRB and TCP/IP handle the routing of data.

*3) Search:* Searching is done based on metadata attributes which are extracted and managed by the SRB and this system which handles metadata is centralized so that the scalability of the network can be challenged

*4) Add/Update:* There are two ways when adding data into SRB. They are Registration and Ingestion. Registration does not touch the data; it only registers a pointer to the data in MCAT without transferring any of the data contents. Ingestion is similar to registration but also transfers the data to an SRB storage resource. This SRB uses parallel I/O to efficiently transfers data over TCP/IP.

*5) 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.

## *B. Apache OODT*

OODT[10]is a middleware system for metadata that provides transparent access to the resources, originally designed by NASA. This maintains a distributed resource system ensuring the scalability and allowing distributed computing. OODT can be used for hardware file management, linking databases, information integration, etc. OODT facilitates functionalities such as store, retrieve, search and analyze distributed data, objects and databases jointly. OODT provides a product service and profile service and they manage data and metadata respectively.

*1) 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.

*2) 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.

*3) 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.

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

*5) 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.

## *C. 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. Users may store resources can of almost any kind: Image files, PDF documents, spreadsheets, Java archive (jar) files or anything else that we need to store and manage. Like other file systems, there is inbuilt facility to create folder structure and store resources within it. This feature allows us to easily search for resources with their metadata in hand. The Registry provides a secure authentication mechanism to secure the resources stored within it. System defined metadata such as date created, last modified date, are updated automatically for resources while the users are allowed to add additional metadata as tags or properties. It also has built in facility to search for resources based on their name, metadata, etc.

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.

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

*2) 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.

*3) 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.

### *D. 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 name space of the file system. Generally DataNodes are deployed one per node in the cluster and is responsible of managing storage attached to that node. There is a file system name space exposed to the user and allow them to store data in files. The NameNode handles the functionalities like open, close and rename files and directories and resolve the mapping between blocks and DataNodes. DataNodes stores internally split file blocks and responsible of reading and writing to those file blocks regarding the client's read/write requests. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

*1) Storage /Replication:* HDFS is designed to store very large data sets reliably and to stream those data sets at high bandwidth to user applications. Hadoop supports hierarchical file organization where user can create directories and store files inside those directories. Hadoop 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.

*2) 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.

*3) 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.

*4) 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.

*5) 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. Default policy in HDFS to keep the deleted files for a 6 hours period, which can be override by user with well defined interfaces.

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.

## III. MAHASEN HIGH LEVEL ARCHITECTURE

As shown by Figure 1, Mahasen system consists of many storage nodes, and they are connected as peers to a logical ring via FreePastry. Each node consists of a registry to keep metadata and a file system to store physical file parts. Once connected to the ring each node contribute 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.

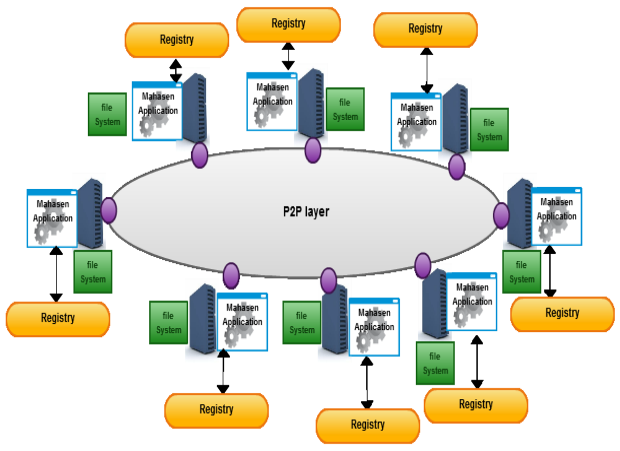

**Figure 1: Mahasen High Level Architecture**

In Mahasen, each storage node is in fact a WSO2 registry, and we used DHT based architecture to make all WSO2 Registry Nodes to work as a one unit.

Mahasen has a distributed metadata layer that stores data about the distributed files in Mahasen peer to peer network. Mahasen 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, and the user defined metadata is used to provide users the searching capability on those metadata. User has been given the opportunity to add tags and properties (name, value pairs) to the files that are uploaded. File name, file size, stored node IPs of file are examples of the system-defined metadata.

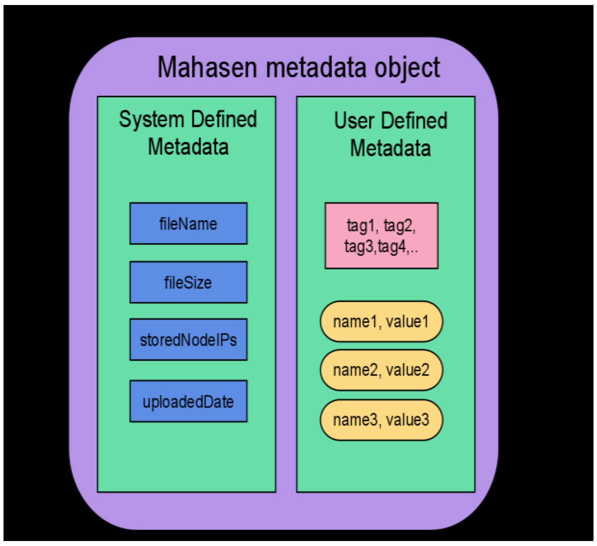

**Figure 2: Metadata Object Structure of Mahasen**

When the user uploads a file to a node in to Mahasen network via a client, the node will get the file and store it in a temporary location. This node will then act as the master node and will split the files in to a set of pre-defined sized chunks. Then those split parts are stored in selected nodes of the neighborhood nodes of master node and this file transfer is done in parallel. Then the metadata object created by the master node is stored in the DHT using PAST storage implementation of Free pastry with replicas. We have rewritten the individual PAST node's persistent storage using WSO2 registry and when PAST stores data in a specific node, it is stored data inside the WSO2 registry that runs with that node.

After storing the metadata, the nodes that received file parts act as worker nodes and replicate their file parts according to the replicate request issued by the master node. Each worker node will update the metadata object with stored locations after replicating their file parts. File replication is also done in parallel providing an efficient file uploading and transferring capability with concurrent access to metadata objects, and Mahasen handles them using the locking system provided by the lock manager of DHT.

User may request to download a file from any Mahasen node, and the node, first generates the resource ID for the requested file using the file path, and retrieve the metadata object related to that file. Then node extracts the addresses of Mahasen nodes that contain the file parts from the metadata object and retrieve those parts to the local machine. After getting all parts of the file, the parts will be merged to create the original file 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 data product (file) from the DHT, updates it,and stores it back in the DHT.

To summarize, when user adds a new data product to Mahasen, it generates the metadata for that data product and stores them in a DHT ring. Actual data product is stored also within some of the Mahasen nodes, and Metadata object for that data product points to nodes where the actual data product is stored. Here the DHT is used to distribute and locate metadata object, and it is very much scalable. 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 assuring the scalability using free pastry's underlying DHT.

## *A. 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 underline DHT, which can improve the performance of different searches that Mahasen supports.

Each resource in Mahasen may have associated name-value pairs and tags. For each tag or property in system, we maintain index pointing to all resources have that tag or property. We use the tag or property value as the key, and store the index (which is implemented as a TreeMap [16] object) in the DHT. DHT also handles the replicas of the Index

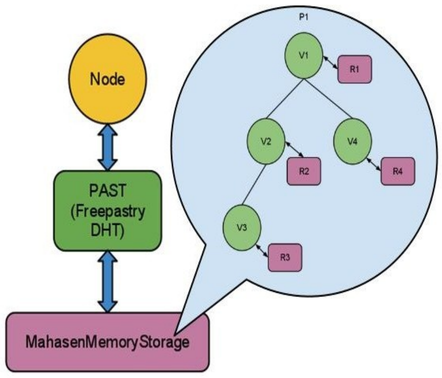

**Figure 3: A property Tree Stored in Mahasen Memory Storage**

When a user sends a search request Mahasen extract the associated search type with the given inputs 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. For an instance, if the user request for the files of size between 200MB and 500MB those values should be given as initial and final values for the search and the request is directed to rangeBasedSearch method along with the user inputs. 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 the index for the current property in a different node, the node who receives the request 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 given by the user, iterate through the sub map and retrieve the set of resource IDs mapped to each of the node in the sub tree. These resource IDs represents the files having the given property values to the specified property. Having these resource IDs in hand, Mahasen can look up for the metadata objects with those resource IDs and extract the file names to present to 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.

### B. *File Handling*

*1) 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. After the client gets authenticated, he is capable to upload download, delete, update or search 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.

*a) Replica Management:* To achieve fault tolerance and fail over recovery, the file will be split into a set of predefined chinks and each of the split parts will be replicated to a number of times and stored in different nodes according to predefined replication factor. The placement of replicates is 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 evaluating 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. The initial node, which the client made the connection and transferred the file, will then replicate and transfer the file to other nodes. The number of copies of a file is replicated called the replication factor of that file. This number will be decided by the Mahasen system.

*b) File Splitting and Parallel transfer:* Mahasen storage network is designed to store large files reliably across distributed nodes. As mentioned earlier, when storing the file as replicas first the file will be split into blocks of fixed size and these blocks only will be replicated across the network for fault tolerance. After storing each block of a particular file, the metadata object of that file will be updated with the stored information for later retrieval. 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, first the system will locate the metadata object of the file and retrieve the information about the stored location of the blocks of that file. The system will then select a node which is closest to the reader node and download the blocks to the client. In here also the downloading of file blocks will be done in parallel and then the blocks will be merged to create the complete file.

### C. *Mahasen API*

*a) Put:* An external Mahasen client can upload files to Mahasen network by connecting to one of the Mahasen nodes. Client is given the capability of adding user defined metadata and tags to the uploading files so that he can later search the files based on those metadata. Put method will store actual file as well as the metadata with replicas.

*b) Get:* When a user request for a file from a contact node, get method will retrieve the file parts by name, merge them and stream to the user.

*c) Delete:* A client can delete a file by giving the required file name. Then the Mahasen system will delete the actual file including all its' replicas and delete the replicated metadata as well.

*d) Search:* Search is the most important functionality provided by Mahasen. Mahasen provides four types of searches, simple property search, range based search, tag based search and advanced search. Searching can be done based on system defined metadata as well as user defined metadata

*e) Update:* Update method will give the client the capability of updating the user defined metadata. If a user needs to update the tags or the properties of an already existing file, those metadata can be updated by specifying the file name that he needs to update and sending the new properties and tags that should be added to the file with given file name. Other than updating the existing properties and tags of a file, Update method allows the user to add new properties and tags to a file in the network.

## IV. Performance Analysis

We tested Mahasen Scalability by running a Mahasen system with M nodes and loading it with N parallel clients. Here M was {1, 5, 10} and N was also {1, 5, 10}. Each client performed operations upload, download, update, delete and search on the Mahasen system 10 times. Following Figure 4-7 depicts the results. In the upload test we used 11.4MB files in each operation to upload using the client.

In our configuration we used 10 machines with Intel Core 2 Duo 2.0 GHz CPUs and 5 machines having 2GB RAM and other 5 with 3GB RAM configurations. We connected all the nodes with 100 Mbps Ethernet cables and created a one Mahasen Ring to establish the test setup.

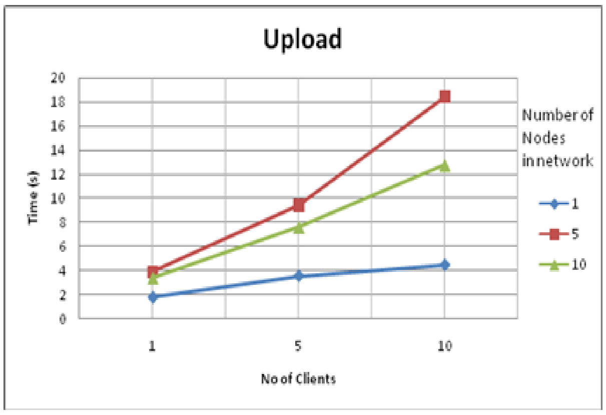

**Figure 4. Upload test results**

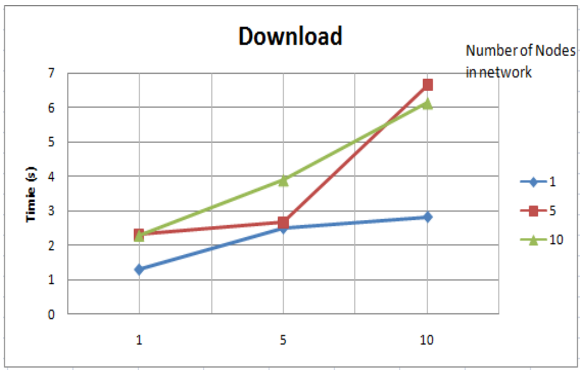

**Figure 5. Download test results**

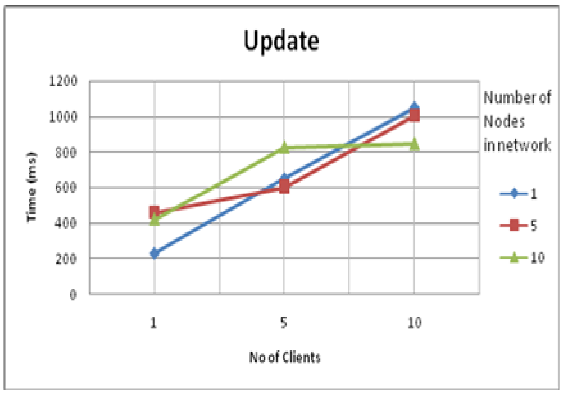

**Figure 6. Metadata update test results.**

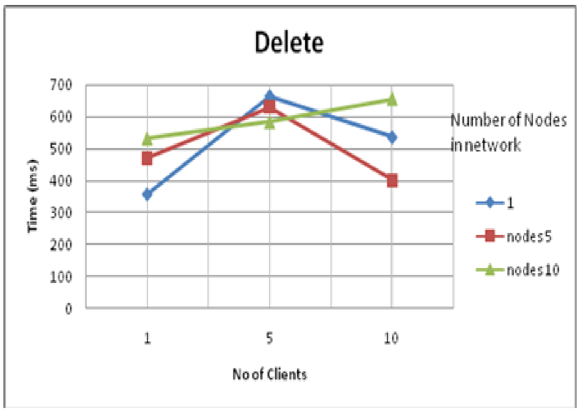

**Figure 7. Delete test results.**

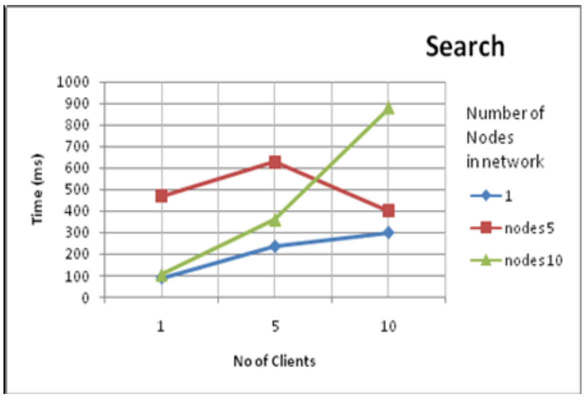

**Figure 8. Mahasen property search results.**

In the results we observed that when we have larger number of clients the upload time grows, and we believe this is due to network congestion. In Mahasen, we are trying to keep replicas only if we have ample nodes to store them. When we have one node we are not keeping replicas since everything stores in one node. Therefore the amount of data related in that operation is reduced. But we can observe in 10 and 5 nodes setup the resulted time increase may be possibly due to replica transfers across the network.

When we download files, Mahasen node that receives the download request from the user needs to collects the file parts and joins them prior to transfer back to client. When we have a larger number of nodes the file distribution is fairly sparse. This will reduce nodes being exhausted by large requests thus performs better. In a single node setup the file parts are stored in local node reducing the latency of file parts transfer.

Therefore we can observe a drastic reduction in time to complete the operation.

During the update operation we were only updating the metadata of the system. When we scale up the system to 10 nodes, there is a performance improvement. When we update the metadata, Mahasen has to look up the actual metadata object as well as updating the indexing structure built for search. For example, when we update 10 properties we need to lookup 10 property trees and then update which resulting increase number of messaging between nodes. In a dense network with less number of nodes latency can be high compared to sparse system due to message queuing in node handles of FreePastry.

When Mahasen performs a Delete on a resource it involves 3 operations; deleting Metadata, deleting entries from search index, and deleting the physical file. Results illustrate that Mahasen can perform better with more nodes when the no of clients increases minimizing the loads of a node.

## V. Discussion

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.

In Mahasen, the metadata catalog is implemented by using the DHT functionality of FreePastry together with the storage functionality of WSO2 Governance Registry. Within each Mahasen node, a registry is used to store the metadata objects persistently in the system.

Mahasen keeps replicas of files uploaded to the system to preserve high availability and fault tolerance of the system.

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. When a user adds a data item to Mahasen, it assigns a node to hold its metadata based on DHT routing through PAST, and store the data item as parts across the nodes. Then any queries or changes are supported by looking up the metadata object for the resource using DHT lookup, and then using the information in the metadata object.

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 trough a custom client or basic client provided by Mahasen. A client does not have to connect to storage network but can utilize the service through web services remotely.

## 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.

[2] Chaitanya Baru, Reagan Moore, Arcot Rajasekar, Michael Wan, “The SDSC Storage Resource Broker.”

[3] [3] Reagan W. Moore, “Managing Large Distributed Data Sets using the Storage Resource Broker.”

[4] G. DeCandia, D. Hastorun, and M. Jampani, “Dynamo: Amazon’s Highly Available Key-value Store.”

[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®),” 06-May-2011. [Online]. Available: http://www.nirvanastorage.com/index.php?module=htmlpages&func=display&pid=1. [Accessed: 06-May-2011].

[8] “XML Metadata Concept Catalog (XMC Cat) | Data to Insight Center | Indiana University Pervasive Technology Institute.” [Online]. Available: http://pti.iu.edu/d2i/xmccat. [Accessed: 26-Sep-2011].

[9] “Performance.” [Online]. Available: http://www.nirvanastorage.com/index.php?module=htmlpages&func=display&pid=54. [Accessed: 26-Sep-2011].

[10] “ApacheTM OODT,” 25-Sep-2011. [Online]. Available: http://oodt.apache.org/. [Accessed: 25-Sep-2011].

[11] “WSO2 Governance Registry - lean.enterprise.middleware - open source SOA | WSO2,” 04-Sep-2011. [Online]. Available: http://wso2.com/products/governance-registry/. [Accessed: 04-Sep-2011].

[12] “Apache Lucene - Overview.” [Online]. Available: http://lucene.apache.org/java/docs/index.html. [Accessed: 26-Sep-2011].

[13] “HDFS Architecture Guide,” 06-May-2011. [Online]. Available: http://hadoop.apache.org/hdfs/docs/current/hdfs_design.html. [Accessed: 06-May-2011].

[14] “Pastry - A scalable, decentralized, self-organizing and fault-tolerant substrate for peer-to-peer applications.” [Online]. Available: http://www.freepastry.org/. [Accessed: 06-May-2011].

[15] P. Druschel and A. Rowstron, “PAST: A large-scale, persistent peer-to-peer storage utility,” presented at the HotOS VIII, Schoss Elmau, Germany, 2001.

[16] “TreeMap (Java 2 Platform SE 5.0),” 04-Sep-2011. [Online]. Available: http://download.oracle.com/javase/1.5.0/docs/api/java/util/TreeMap.html. [Accessed: 04-Sep-2011].

[17] “HttpClient - HttpComponents HttpClient Overview,” 04-Sep-2011. [Online]. Available: http://hc.apache.org/httpcomponents-client-ga/. [Accessed: 04-Sep-2011].

[18] Alejandra Gonz´alez Beltr´an, Paul Sage and Peter Milligan, “Skip Tree Graph: a Distributed and BalancedSearch Tree for Peer-to-Peer Networks.”