San José State UNIVERSITY

Team Project 1

File Edge

CMPE 275

Fall 2013

Submitted To

Prof. John Gash

Submitted By
Amrita Mohanty < amritamohanty2004@gmail.com>
Shobha M Gowda < shobha.gm@gmail.com>
Shrikanth Krishnamachari <krish1631988@gmail.com>
Veenu Agrawal < agarwal.veenu18@gmail.com>

Table of Contents

1. Project Statement:

The focus in this project was to build a distributed storage based application, which could satisfy the requests for storing any document (of any type) while replicating it within the cluster, finding the document in the cluster with an added functionality of querying that document. The project also provides the functionality for removing the document from the cluster. The base technologies to be used for the project was that of JBoss Netty and Google Protobuf which enabled us to communicate in distributed environment asynchronously and also understanding the messages between the nodes respectively.

However, the project had an urge to work collaboratively not only within the team but also with other teams to help finalize the global standards. This collaborative effort helped the team to understand various concepts related to designing effective architecture.

2. Scope of our project:

In this section, we would like to discuss about the scope of our project and would like to mention about what functionalities we have developed in the project. This document would be entitled to provide required explanation for all those implementations and their approaches. Below are the functionalities that we have covered in our project.

a. Servicing the requests for document storage and replicating it within the cluster.
b. Servicing the requests for document retrieval or finding from the cluster.
c. Servicing the requests for querying for the document within the cluster and from external cluster.
d. Servicing the request for removing the document from the (internal) cluster.
e. Implementation for external cluster communication whenever our internal cluster could not satisfy the DOCQUERY requests.

In the below sections we would be discussing about the implementations details for the above functionalities. Thanks to professor Gash that we were required to try out different approaches while implementing any functionality and learnt about advantages and disadvantages of those.

Event Driven communication and Asynchronous Message handling

This project FileEdge uses Netty’s asynchronous, event driven application framework for message handling. When a client sends a request to the server, a channel is created and a permanet pipeline gets associated with it. This pipeline has a list of channel handlers which handles or intercepts channel events i.e. I/O events. Some of the channel handlers inside the pipeline are listed below.

Protocol Decoder - translates binary data into a Java object.
Protocol Encoder - translates a Java object into binary data.
Business Logic Handler - performs the actual business logic (e.g. database access).

A server handler(which is our business logic handler) instance is created for each socket connection this in turn allocates a new PerChannelQueue, which is a server queue for processing the incoming requests. Every ChannelQueue is associated with inbound and outbound queues monitored by inbound worker thread group and outbound worker thread group.

Every request is thus enqueued to the inbound queue of the PerChannleQuueue and processed by inbound worker threads. The response obtained after processing the request is enqueued to the outbound and outbound worker threads picks the message and writes in the main pipeline and a downstream event is notified to the client side handlers.

3. Collaboration effort:

As mentioned earlier, for the successful completion of the project, it was necessary for our team to collaborate the efforts internally as well as with external teams too. As a team we made it a point to try out different possible solutions for certain functionalities and discuss about the fallacies (if any) they presented. The team members were also asked to discuss with other teams as well about how they solved the problem and vice-versa. The team also actively participated in finalizing the global standards to be able to communicate within the large clusters.

4. System Architecture:

We have built a distributed file system that focuses to provide distributed file storage, search capabilities in the network of servers or nodes and the capabilities for querying to removing of the document. Our system can handle multiple connections asynchronously. Network discovery has also been implemented to be able to communicate across the internal cluster. Therefore, our system could interoperate between internal and external servers or nodes in order to form an overlay network.

The team cluster consists of four nodes that act as servers and they are connected to each other following a ring topology. Every server in the cluster has a backend database support with the likes of PostgreSQL. There is two pronged communication between the nodes of the clusters differentiated as internal for health check message exchanges through heartbeats and public for application oriented message exchange (sending and retrieving documents or querying and replicating the documents). The network paradigm used in the project is that of JBoss Netty and hence understanding the internal working of the framework was important too. The system is built with considering the fact to avoid single point of failure.

For message exchanges at the public messaging level, Google Protobuf has been used which has the required message structure for each type of message that would be traveling across the system including internal and public messages. The Proto file underwent several changes after having discussed global standards for the project.

Every server in the cluster would accept and service the request for file storage and would replicate it across to it two adjacent notes. For replication, we considered replicating both the metadata and the actual file within the cluster. For file storage we are storing the file into physical disk and metadata has been stored about the file in the DB. The folder path for the file that is stored in the DB is relevant to actual path of the file in that particular server. For servicing the requests related to DOCFIND, DOCQUERY or DOCREMOVE, we first try to service it at the first receiving server and if not satisfied the request is propagated within the cluster following the ring topology. Additionally, in case of DOCQUERY, we also forward the request to the external cluster if the request could not be satisfied after exploiting all the nodes of our cluster. We also use correlation ids to be able to distinguish between different requests and map them with corresponding responses.

Every server at the start up picks up the required configuration detail from JSON file and starts building public and management boot using the public and management ports respectively. At the client side whenever a request is being sent to the server request type is notified in the request header. Client side implementation also has some listeners implemented to notify particular client when particular request is satisfied and has response. Even the responses have reply status codes.

5. Design Consideration for System Cluster:

After several discussions within our team and with other teams, we came up with two possible approaches to design and build the File Edge system. We were able to figure out the pros and cons of each approach in order to finalize our design.

5.1 Approach-1: Can we have a Single Master node:

In this approach, we decided that one of the nodes in a cluster would serve to forward request to next nearest node i.e serve as the master node. Therefore, any request, such as finding a particular document in internal network, downloading a document from a node, saving the document on a node and then replicating it to two other nodes in the cluster, is sent to the master first and then it forwards the request to the adjacent nodes. In addition, the master would keep track of all the information across the cluster and would be able to guide the cluster nodes accordingly. Although this approach had its advantages but there were some disadvantages too. After brief discussion with professor, we were able to figure those disadvantages for building the system using this approach.

Advantages:
Since, we have a master node, load balancing could be achieved between the cluster nodes as master would have the required information about the data being stored in the cluster nodes and the size available with the cluster nodes as well. Even the master node would be able to analyze how much request would be handled by which cluster node.

Disadvantages:
The following were the disadvantages that were evident in this approach:

1. **Inefficient usage of resources in a cluster:** As one of the serve only holds responsibility to forward request to other nodes, so the resources of this node is utilized to its full capacity where as the other nodes may be under-utilized.

2. **Single Point of Failure:** If the master node goes down then this would lead to single point of failure in our case as we did not have anything called as backup master node to take over the responsibility of the primary master node.

After evaluating the possible disadvantages, the most significant one being the single point of failure, we came up with another approach that handles the single of point of failure scenario for us.

5.2 Approach -2: Can we consider each node as master:

In this approach, each node in the cluster holds similar responsibility and acts in a similar way while accepting a request, processing the request and sending the response to a client, i.e. each node serves as a master node. Each node is capable to process all kinds of incoming requests such as saving, replicating and finding a document even if an external node initiates the document-find request. This approach helped us to overcome the disadvantages of the first-approach. We were able to eliminate single point of failure and inefficient usage of resources in this approach. Our system is implemented using this approach. However, we did face some disadvantages with this approach as well and they are as below:

Disadvantages:
1. **Overwhelming utilization of resources:** Since every node in the cluster is alike in behavior, every node would try to save a file and replicate it across to adjacent nodes. This will cause overwhelming utilization of resources as at some point of time the cluster would fall short of storage space to be able to satisfy most of the requests.

6. Strategies Used to Implement the System Functionalities:

In this section, we would like to discuss about the strategies we thought of and tried while implementing our system functionalities. The strategies that would be discusses here are for functionalities such as saving the file and replicating it, searching or querying for the file and retrieving it and removing the file. In addition there are strategies discussed for external cluster communication.

6.1 Document Upload and Replication:

For replication of the document within the cluster, we at first thought of having only one node that would accept the write document save request and would replicate it across to the adjacent node. So whenever a new request came to any other node other than Node 0, the request would be forwarded to the Node 0 and from there the replication would take place. In this case we have had to face the challenge of distinguishing between the different types of save requests as in this case there would be simple save request which means just save the document and save+replicate request to save the document and replicate it to the adjacent nodes. This approach that we thought had some disadvantages even though it avoided SPOF. These are as below:

Disadvantages:

a. Since there is only one node Node 0 that would be able to satisfy the Save+Replicate request, if a request comes at Node 2 which is two hops away from Node 0, the request must have to travel to Node 0 and then get replicated. This would cause delay in actual replication of the document.

b. Since, Node 0 would be the node to initiate the replication task, only nodes that would get exploited in this approach would be Node 0, 1 and 3. Hence, Node 2 will never be utilized for storing the file or in other words, we are not utilizing the storage of Node 2.

c. If by any chance, Node 0 goes down then our system would have to operate on degraded mode where it would only be able to satisfy the read requests and no more write requests would be accepted or serviced. But even in this case there is no SPOF.

The approach that we discussed in Approach 2 of Section 5, solved the above issues for us. We made every node in the cluster as write node, which can accept the both simple save and save+replicate requests very well. In this case even if any single node goes down, the other nodes can server well as write nodes. This is the approach that is currently implemented in the system. However, we did specify the disadvantages of this approach. Below are the implementation details for this one. The Figure 6.1 gives diagrammatic representation for the approach followed.

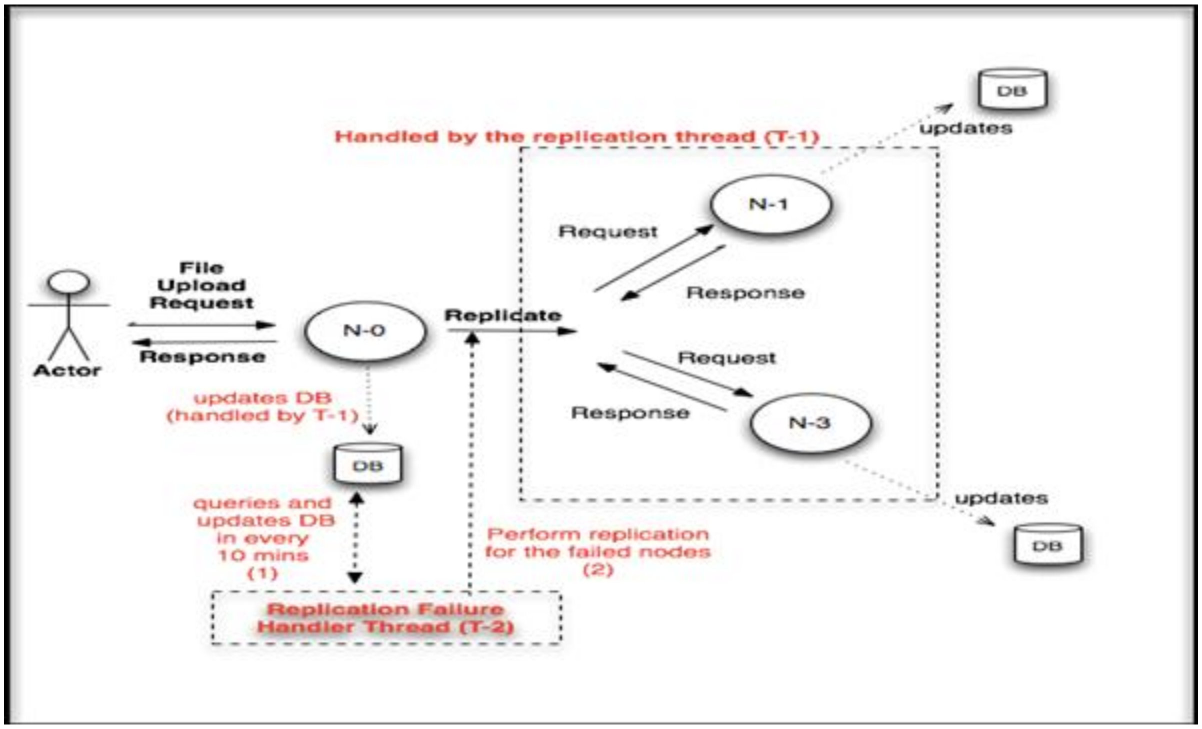
Handled by the replication thread (T-1)
updates
DB
N-1
Request
Response
File Upload Request
Response
Actor
N-0
Replicate
Request
Response
N-3
updates DB (handled by T-1)
DB
updates
DB
queries and updates DB in every 10 mins (1)
Perform replication for the failed nodes (2)
Replication Failure Handler Thread (T-2)

Figure 6.1: Approach followed for replication

Design:

This design that we followed has various steps or stages as explained below.

Step-1:

According to our design, the user can upload the file in any of the four nodes in the ring topology. In order to upload the file in the server, user needs to provide the following information:

1. Server IP:
   If server the user then the default server-IP does not provide IP is localhost.
2. Port No.:
   User needs to select a port number between 5570 to 5571. Incase the user enters any other number, then the default port 5570 is taken into consideration.
3. File name to be uploaded along with the file path.
4. Namespace:
   If namespace is provided then the file is uploaded in the server in the given namespace inside the default folder else the file is uploaded in the default folder of the server.

```
DEFAULT-FOLDER
        |
        |-Namespace (optional)
                |
                |- File
```

Incase the sub-folders given in the namespace are unavailable; they are created at the time of uploaded the files in the server.

Step-2:
Once the user provides the information asked for in Step-1, the files is uploaded in the server and port entered by the user. Also the file is stored in the namespace provided by the user or else in the default folder. A reply (response) is sent to the client informing him the status of the file upload in the server i.e. success or failure.

Step-3:
After a file is saved in the server a thread (i.e. T-1 in Fig-4) is spawned which takes care of storing the metadata about the file in the "localfiles" table of the database and isOwner column is set to 1 for the original node.
The "localfiles" table has the following columns:

fileId (PrimaryKey)	**filename**	**filepath**	**namespace**	**isOwner**	**isReplicated**

1. **fileid :** This column is the primary key and is the time (in nanoseconds) when the files was stored in the server.
2. **filename :** Name of the file stored.
3. **filepath :** This column contains the path of the location where the file is stored in the server along with the filename (i.e. filepath + filename).
4. **Namespace :** Contains information about the namespace provided by the user for storing the file in the server.
5. **isOwner :** this column is used for determining whether this server is one where the file was originally uploaded or whether it is just a replicating node. If yes then isOwner = 1 else 0 for replicating nodes.
6. **isReplicated:** this column holds the information about whether the files has been successfully replicated in both the adjacent nodes or node. Only if the file replication is successful in both the adjacent nodes the value of isReplicated = 1 else it is 0.

Then the thread replicates the files in the adjacent nodes. If the reply status from the replicating node is success, then the fileId and replicating node information is stored in the "replication" table of the server where the file was originally uploaded. Also, the localfiles table of the replicating node is updated to store information of the file added to the node and isOwner column is set to 0.

The "replication" table has the following columns:

fileId (ForeignKey)	**replicatedNode**

1. **fileId:** It is a foreign key and maps to the fileId column of the localfiles table.

2. **replicatedNode:** contains information about the server and the port# where the files is replicated. replicatedNode = serverIP:port#

Handling the replication failure scenario:

In order to handle the replication failure scenario, a thread (i.e. T-2 in Fig-4) is running in the server which wakes up in every 10 minutes and queries the localfiles and replication tables to look for files with isOwner=1 and isReplicated=0 and determine the adjacent nodes where the replication had failed and try to resend the file to those nodes. Once the replication is successful the node information is updated in the replication table. Also, if the replication is successful in all the adjacent nodes then the isReplicated column is set to 1.

Initially, the replication thread was actually sending the request for replication to the adjacent nodes sequentially and we were blocking the response to be sent to the client after the first save. We handled this by having one more threaded implementation at the replication logic at the server side. Now, the moment the file is saved in first server, the client will get the response for the save, and then thread would take the responsibility for replicating it to the adjacent nodes and updating the DB Tables.

Advantage of this approach:

The main advantage of this approach was that even if an adjacent node is down during replication, the files that were unable to be replicated in that node can still be replicated once the node is up and running again.

Disadvantage of this approach:

Consider the replicating factor of the network is N. If one of the adjacent nodes is down then the replication factor of the file will be N-1 as long as the node is down. Also, over a period of time there would be no storage left in the cluster and there would be need to increase the storage limit as there would overwhelming resource utilization since all of the nodes are acting write nodes or master nodes.

6.2 : DOCQUERY and DOCFIND implementation:

The client can request to find a document on any node in the cluster and receive the valid response. The client just needs to provide the document name it is searching for. The entire process of sending the request and receiving the response is asynchronous. Request should contain the filename. The database of each server stores the absolute file path of the documents stored in it. If the file is found, the server retrieves the file path from database and searches for the file in the server's disk. The file contents are sent in byte format as response to the client. We need that client or any other external node do provide us with the filename which is a mandate and optionally a namespace. If the client is not sure about the namespace, the client must mention namespace = "" and our server would find the document in the default namespace folder i.e. "default". Below are the cases that we tried in the project.

Case 1: Query or Find successful in the internal network

When the request to find a document is fulfilled by the node where the client had sent the request to, then the response is sent back to the client by the first traversed node. For example, the client sends a request to find a document on Node3 and the document was successfully found in Node3, then the response is immediately sent to the client. But incase, the node does not have the requested document, the request is forwarded to the other nodes in the ring network using the following two strategies:

Strategy 1: Forwarding the request in a ring and sending the response back in same path.

When the request is sent from Node1 to Node2 and the document is found under Node2. Then, in this condition the response is sent to client from Node2 to Node1 and then finally to the client. In this approach, since at the time of forwarding the request to other nodes, the previous nodes would have to act as client, we added the same type of implementation for Handler and Decoder that existed between client and the Node 0. But we did not have the need to implement listeners in this case. The classes WhenServerBecomesClientHandler.java and WhenServerBecomesClientDecoderPipeline.java serves this purpose. Also, we maintain a cache for the connection between the Nodes, so as whenever there is a need for forwarding the request the connection is fetched from the cache based on adjacent nodes nodeId. Now, to be able to trace the request and it matching response, we added correlationId in the comm.proto file which would be randomly generated and would be unique between request and response. This helped us solve the problem of tracing the request and response in the asynchronous environment. So whenever the request was forwarded from one node to the other node, we also cached the PerChannelQueue instance with correlationId as key. Hence, when there was the need to send the request back to the client, the required perChannelQueue instance was retrieved and the response was enqueued on it. The below figure 6.2 represents this approach.

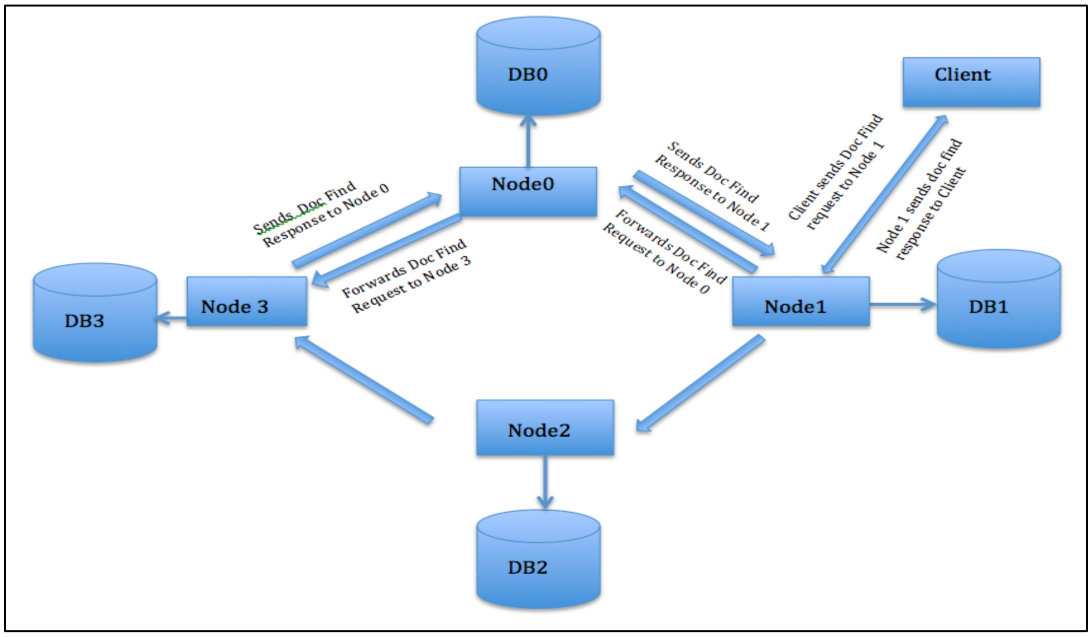
DB0
Client
Node0
Sends Doc Find Response to Node 1
Client sends Doc Find request to Node 1
Sends Doc Find Response to Node 0
Forwards Doc Find Request to Node 0
Node 1 sends doc find response to Client
Forwards Doc Find Request to Node 3
DB3
Node 3
Node1
DB1
Node2
DB2

Figure 6.2: Approach1 followed for both DOCQUERY and DOCFIND.

Advantage of this approach was that we did not have to traverse rest of the nodes to provide response even if we found the document at Node 1 or Node 2. We were able to trace back the same path the request initially travelled. But there was also a possible disadvantage too. If the request was satisfied at Node 3 then we still had to trace back the same path as Node 3-Node2 – Node 1- Node 0, thereby delaying the response. We thought we could try one more approach for this which is described below.

Strategy 2: Forward the response as request to the next node:

Client sends a request to find a document to Node0 but the document was not found there. Then Node0 forwards the requests to Node1, which in turn forwards the request to Node2 if it could not find the document. If Node2 could has the document, it fulfills the request and sends the response from Node2 to Node3, and then Node3 forwards the response to Node0. Finally, Node0 sends the response to client. In this approach we did not need the use of WhenServerBecomesClientHandler.java and WhenServerBecomesClientDecoderPipeline.java as it was only request that was getting traveled to the other even though it was actually a response.

But we had some disadvantage in this too. What if request got satisfied at Node 1, in this case still the response would have to travel as request till it reaches the initial Node 0. Also we had some challenge in matching the request to its response (which is also a request), but we managed it by having the correlationId untampered during the response to request conversion. (this approach can be found in the alternate zip file in submission.). The Figure 6.3 shows this approach.

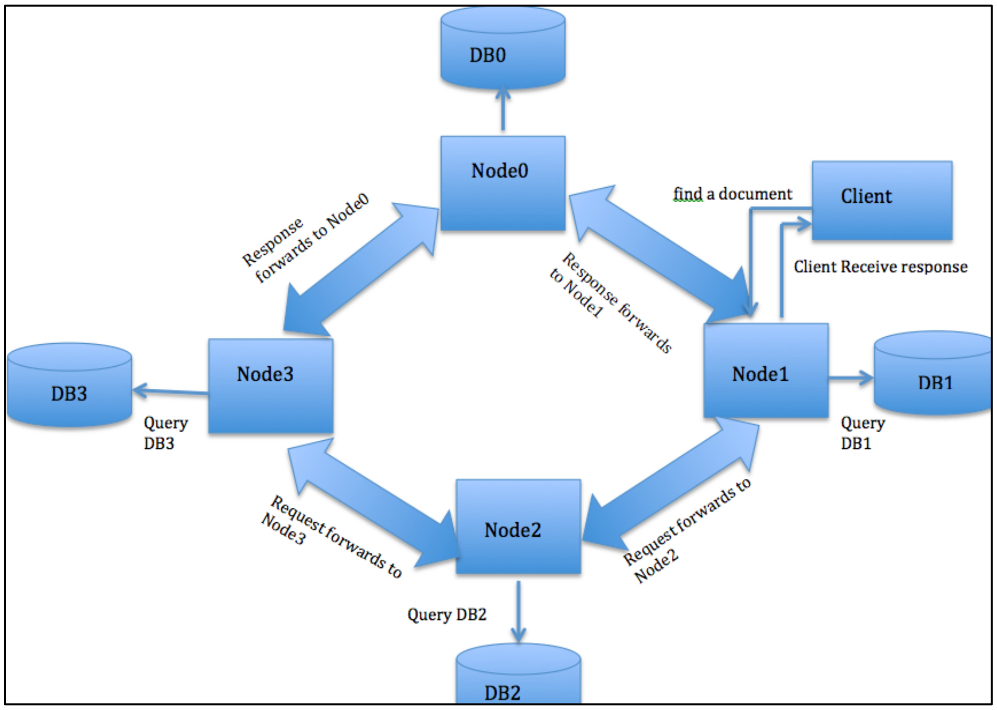

Figure 6.3: Approach 2 follwed for DOCFIND and DOCQUERY

However, in both the approaches, we did not implement failure scenarios in the ring. Although the system would still be able to perform without SPOF. Initially, we thought of using remainingHopCount internally when we forwarded the request to the internal nodes when one particular node was not able to satisfy the request. Then after meeting with the other teams and having a discussion with professor, we reverted the code to use remainingHopCounts for external nodes only. Now the remainingHopCounts would be decremented only when all the internal nodes are exploited.

The difference between DOCFIND and DOCQUERY implementation in out project is the response for DOCQUERY would have only success message but for DOCFIND we retrieve the document and sent it across to the client by following the approaches. In that case we read the document from the physical drive and convert it to bytes and attach it to the response. At the client side we read the response and convert bytes to actual file. For both we need namespace and document where namespace can be “” if the client is not sure about it.

6.3 : DOCREMOVE implementation:

In this implementation, we only considered about the removing the file from only our cluster. So there is a possible disadvantage that a file may exist in other cluster, but we do not forward the request to other cluster to remove that file which is not correct. But this implementation was done as teams decided so in the global standards meeting. In this case two we require namespace and document name and the request keeps travelling to the nodes until all the nodes have removed the document from their DB and physical drive and the client is notified of the success message.

6.4 : External Node communication implementation:

To be able to forward the client request to the external cluster if by any chance the internal cluster is not able to satisfy the request. In this case, we add a flag isExternal and decrement the rwmainingHopCount by 1 and then forward the request to the external cluster. So our approach is to first exploit the internal cluster nodes to make sure there is no request leakage taking place and if the internal cluster is not able to provide the response then it must be forwarded to the external node.

In this approach, we first detect request loop and if we find that then make the isExternal flag in the request header TRUE and decrement the remainingHopCount by 1. Then again the request is made to travel our cluster. But this time, since every node id would be available in the RoutiingPath and isExternal = TRUE, the request would not be processes by any of the node, instead each node would check the JSON file with itself to find if it has any external node. If it does contain any external node IP and Port then that node will first revert the isExternal to FALSE and then forward the request to the external cluster.

When the response comes back, then also, it can trace the perChannelQueue based on correlationId to send the response to the client.

Even here at first we tried the approach of sending HeartBeat to the external node but that was not required as discussed with professor. In this case the external node would be treated as internal node which is not correct. This approach is implemented in the second zip file. We reverted this logic, as now we do not send heart beats to that node. Instead we create the connection with socket connection approach and cache it for further use. The figure below explains this approach. We are forwarding the request to external node only in case of DOCQUERY.

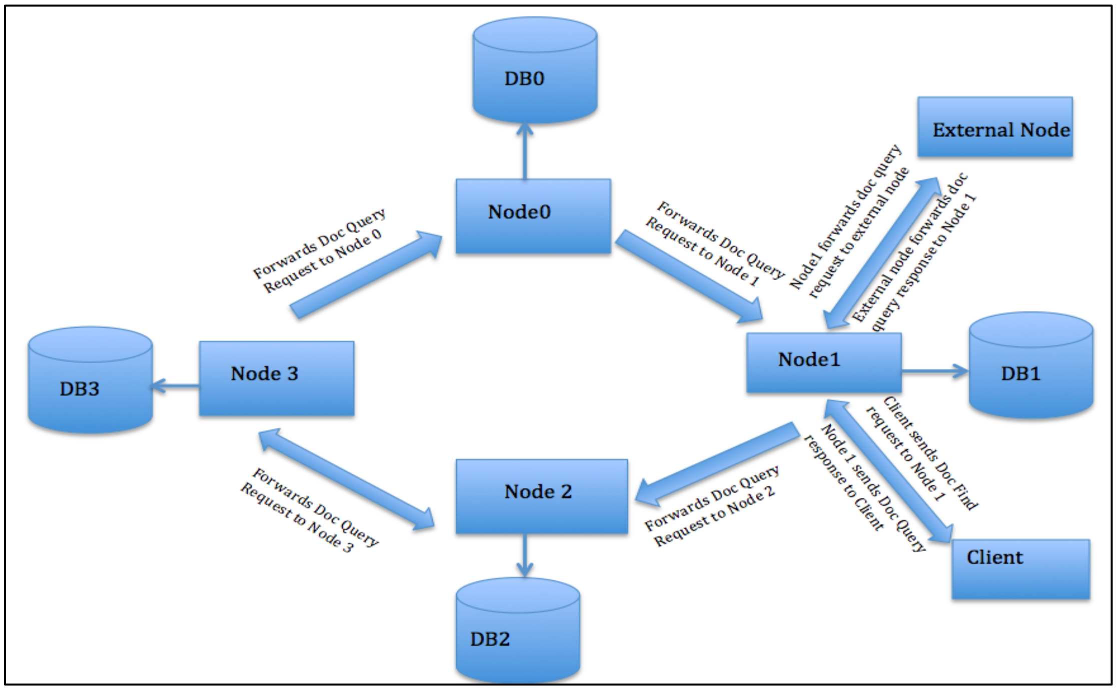
DB0
External Node
Node0
Forwards Doc Query Request to Node 0
Forwards Doc Query Request to Node 1
Node1 forwards doc query request to external node
External node forwards doc query response to Node 1
DB3
Node 3
Node1
DB1
Client sends Doc Find request to Node 1
Node 1 sends Doc Query response to Client
Forwards Doc Query Request to Node 3
Node 2
Forwards Doc Query Request to Node 2
Client
DB2

Fig-6.4: Sending doc-query request to external network

7. Conclusion:

In the end, we would like to say that, although this project seemed daunting to us. But we are able to accomplish several milestones we set. The important milestone was to come up with a good approach to store and find data in a decentralized network of servers. We are able to implement replication as a core feature of our distributed file storage system. Another challenge, which we faced, was setting up global standards and to follow a similar protocol in order to communicate with other teams. Also, the project was thoroughly a great learning experience. We really appreciate the contribution from each of the team members and the guidance that professor gave by directing us to have enough collaborative discussion with other team.