Literature Review on NoSQL Databases with Concentration on MongoDB and Application Level Optimization in Big-Data to enable fast Data Transfer

Dinesh Ramamoorthy and Nayana GK

Abstract — This is the age of Big Data where tremendous amount of data is generated due to the rise of internet and other applications. This has impacted many technologies and caused many innovations as well. This is a work which deals with two such impacts. The first one is the requirement of a database management system which can replace the relational database model, as it has many disadvantages when the data is huge. This has resulted in the evolution of Non-Relational Databases which are also called as NoSQL Databases. The other impact is the need to carry out the data transfers quicker than ever due to the increase in amount of data to be transferred. New technologies have used Parallelism, Concurrency, and Pipelining techniques to increase the speed of Large Data Transfers. This literature survey aims to throw light on these two wide topics.

Index Terms — NoSQL, MongoDB, Non-Relational databases, Pipelining, Parallelism, concurrency, parallel processing, data transfer optimization, cloud networks.

1. Introduction

This work has been intended to conduct a review on topics related to Big-Data. Two most important concepts that has tremendous relevance to Big-Data have been chosen and a wide survey is conducted. The first topic of interest is the changes in database management systems. The huge amount of increase in data has caused the requirements to be more and the relational database systems which already exists does not satisfy all the requirement that the huge amount of data demands. Hence a new database management system called the Non-Relational database systems or the NoSQL database systems have been introduced to improve the performance factors with respect to Big-Data. The first part of the survey addresses the reasons for opting a NoSQL database system than a relational database system.

The NoSQL system is compared with SQL databases and the highlights of NoSQL systems are discussed. There are many options available for a NoSQL system, but we have chosen MongoDB to explore more about this topic. There are many reasons for choosing MongoDB for this purpose and these reasons are discussed in the following chapters.

Comparative studies between SQL and NoSQL systems are carried out with respect to many factors like application development and execution of queries. There are many papers like [1], [2], [3], and [4] which have works based on this topic. We have taken a few and review them and compared the works and provided a clear understanding about the concepts. Then the comparisons are executed and tested respectively.

Relational databases store the data in the form of tables where each table represents an object, where in each table has rows representing a record and each column represents a field. These tables are linked with each other with the help of foreign keys or using common columns.

With recent development in technology traditional storage performed in SQL becomes ineffective. Cloud computing technology is an effective storage mechanism where the data is spread across multiple servers for storage. Using SQL database is not suitable for database operations since it requires joining of multiple tables which are large in size and it leads to a very complex problem. To overcome all these problems NoSQL database provides different means of storage mechanisms which will help in storing unstructured applications in a distributed manner. Papers [7], [8], [9], [10] and [11] are referred to support the topic.

Now that a new database system has been suggested, there must be a way to transfer the data which already exists in a relational database system to the non-relational database system. This review talks about the transition techniques that are used in real time implementation. A review is conducted on the papers [5] and [6] to address this topic.

Now the focus has been moved to the second part of the survey. We have said that the increase in the amount of data has caused many evolutions in the technology that is associated with the data handling and storing. Data Transfer is one of the major processes that is affected by the increase in the amount of data. As the size of data increases the requirement to transfer them in a faster way also raised its head. To address this issue, a concept called the application level optimization is implemented in Big-Data transfers.

Technologies like Parallelism, Pipelining, and Concurrency are used to customize the data transfer. There are many works like [12], [13], [14], and [15] which have experimented and proved that this technique has really impacted the data transfer rates. A review has been conducted with works like these and a comparison between the works is also given. The necessity of parallelism in transfers has been spoken about. There is a minimum and a maximum amount of parallelism that can be implemented in a transfer. There are many ways that has been proposed to predict the actual level. These methods are analyzed and presented in a vivid manner.

Thus, this review will speak about two major impacts on database technology due to the increase in the amount of data.

2. Choosing NoSQL over SQL

This review is started with a focus on the factors that has caused everyone to look for an alternative to relational databases to address the problems that emerged with handling big data. A brief description about the ways in which NoSQL database systems can address these issues and justifications for choosing a NoSQL database over a SQL database is given.

2.1 Problems faced with relational database systems or SQL systems

The primary problem is the inability of relational database system to handle unstructured data. As the data that is generated on a day to day basis due to the advent of internet has given rise to large amount of unstructured data like the images, videos, and others. The other problems consist of performance factors. The performance of relational database systems is comparatively lower than that of the SQL systems when it comes to large amount of data. These problems are spoken about in depth in the chapters to come.

3. Comparison of NoSQL databases with SQL databases

This part of the survey deals with the comparison of NoSQL with respect to the SQL databases. There are many works which have been carried out to compare the two. This review concentrates more on MongoDB to derive a clear picture and make it more focused. The reason for choosing MongoDB is that it has been launched in 2009 and a great amount of research is carried out on this application. MongoDB is suitable for both small scale and large scale applications, where the number of users can cross several thousands. We will review the works [2], [3], and [4]. The comparisons are carried out with the help of performance analysis based on the ability of the databases to handle the SQL queries from the user end. The NoSQL databases also come with high flexibility to add or delete the attributes corresponding to a database because they are not constrained to a schema of database which is fixed.

3.1 Comparative Study with MongoDB

In the work [2], A comparison is made between MongoDB and MySQL to analyze the differences between the relational database model and NoSQL database model. The non-relational databases follow four techniques to store the data, these techniques are described for better understanding of the working of NoSQL databases.

The four techniques are Key-value, Document, Column, and Graph-Oriented. The Key-Value deals with the dictionaries which are conceptually distributed and lack in a schema which is predefined. It has both self-generated and synthetic keys and multiple value types like that of a string or JSON. The Databases which are based on documents does not have a schema which is predefined and hence they are flexible with respect to the content type. They can work with different types of documents. The Bigtable type of databases like the HBase and Hypertable deal with the Column based Databases and they are provided with a schema which is predefined. The columns are a group of cells in which the data is stored. Then the families consist of groups of columns. The number of columns is not limited. The Graph-

Oriented type can assist the queries that are complex. And the execution time of the queries is also observed to be faster than that of the other types of systems that are mentioned above.

This description and study makes it easy for us to understand the basic functionality of the non-relational databases.

3.1.1 Comparison based on Application Development

Here in this part a comparative study has been carried out by the authors by choosing the technologies like MongoDB and MySQL and the comparison grounds have been chosen as the operational commands used in both applications representing the Relational databases and non-relational databases.

The study deals with developing a forum whose structure is dynamic and with an impact of preferences from the user. For a relational database like the MySQL the schema is fixed or static and the user should maintain the structure. For example, the user may contain sub-forums under forums and the method used for implementing the structure remains the same. The order is that the forums are created first followed by the sub-forums then the comments are added in to the discussion.

The MongoDB has collection sets, whose schemas are not predefined and their collection consists of tables and the data is stored in the form of BSON documents. The corresponding documents represents the fields which are like the rows that are present in the collections. The contents may be complex like the lists or they can also contain complete documents. A primary key is assigned to the id field of the document. The queries are applied to the documents.

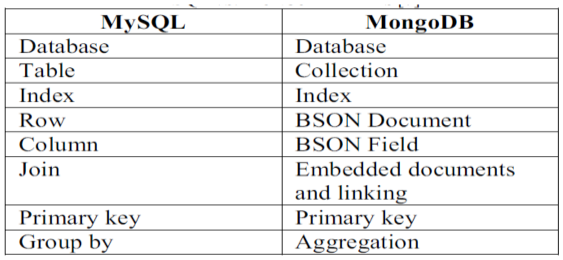
As the non-relational database, MongoDB is used here the users need not maintain a fixed structure like that of a forum followed by sub-forum, but the structure is flexible and can be customized by the user based on his requirements. This is a major advantage for the NoSQL databases when compared to the relational databases. In this paper [2] the integration of bundle is carried on with Symfony2 and the database is configured with the following lines of command.

```
# app/config/config.yml
doctrine_mongodb:
    connections:
     default:
         server: mongodb://localhost:27017
         options: {}
    default_database: my_database
    document_managers:
        default:
            auto_mapping: true
```

3.1.2 Comparison Study

The authors carry out a comparative study, starting with highlighting the differences in terms between the MySQL and MongoDB.

Table 1 highlights the differences in terms precisely and the basic outline of the two database systems can be understood with the help of this table.



MySQL	MongoDB
Database	Database
Table	Collection
Index	Index
Row	BSON Document
Column	BSON Field
Join	Embedded documents and linking
Primary key	Primary key
Group by	Aggregation

Table 1. Difference in Terms between MySQl and MongoDB. (Referred from Work [2])

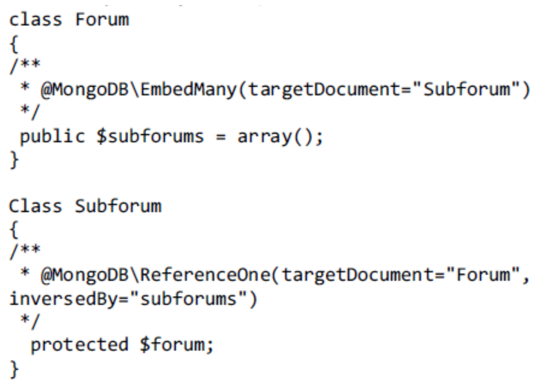
MongoDB withholds a graphical structure, where as the MySQL has a tabular form represented graphically.

```
{
    "_id": "d4acaf3a76e4378b853eb15fde216722",
    "username": "andra",
    "email": "andra@gmail.com",
}

{
    "_id": "d4rvgf3a76e4378b853tr15fde216722",
    "username": "ioana",
    "email": "ioana@gmail.com",
}
```

This example has been used to carry out the comparison study which depicts the user databases which consists of ids for the user which are unique and the email address and usernames are generated automatically. MongoDB consists of three types of users, the administrators, the moderators, and the normal users. The users are provided with the ability to create a forum or a sub forum and has been given the rights of the administrator, the moderators also can add and delete forums and sub forums. When it comes to the non-relational databases, the case is different. The normal users are deprived of the rights to create and edit the forums. The powers can be restrained within administrator and moderator in the MongoDB to reduce the space required for storage. The MySQL occupies more storage space than the NoSQL databases.

It is observed that, the MongoDB also has one to many relationships, like the ones existing in the relational database systems. But the only difference is that the foreign key is absent and is replaced by annotations. Thus, the annotations are used as specified below,



```
class Forum
{
/**
 * @MongoDB\EmbedMany(targetDocument="Subforum")
 */
 public $subforums = array();
}

Class Subforum
{
/**
 * @MongoDB\ReferenceOne(targetDocument="Forum",
inversedBy="subforums")
 */
  protected $forum;
}
```

Ids are used to create connections between forums and sub-forums and to establish a flexible and dynamic connection between them. The future insertion of field into the database is also made easy by inserting the name and the corresponding type without any impact in the data that had been stored already. There is no need to reframe the entire structure of the database.

3.1.3 Tests based on performance evaluations

In order to analyze the performance evaluations, the authors carryout real time performance tests on both the applications MySQL and MongoDB. The basic operations which are used are,

1. Insert
2. Select
3. Update
4. Delete.

Insertion: An empty database has been created in both MongoDB and MySQL. They are provided with the same structure and then

"- Table/document User with the columns: id, username,
password, email.
- Table/document Forum with the columns: id, title, author, info (short description).
- Table/document Subforum with the columns: id, title,
author, info, created, updated.
- Table/document Discussion with the columns: id, title,

author, created, updated, content.

- Table/document Comments with the columns: id, author,

created, content, approved. "

are added as tables to the databases. If it is going to be tables in MySQL and then it is going to be document in MongoDB. A general information dataset, like the user id for 10000 users has been inserted in both the databases.

The analysis has been carried out by measuring the time required for insertion of the same content in both the databases and the time is measured with "microtime" which is a function in PHP. And it has been observed that the time consumed in MySQL was 440 seconds and the time consumed in MongoDB was 0.29 seconds.

The results have been recorded as,

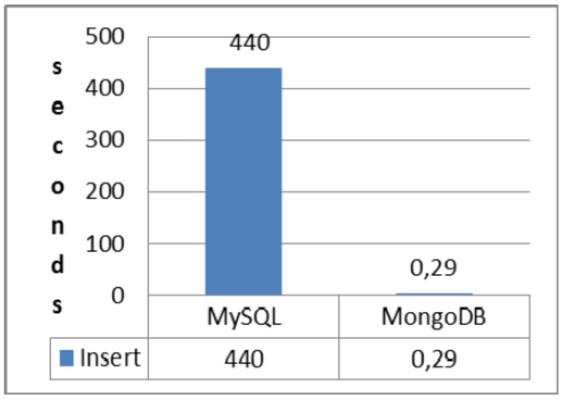


Figure 1. Results of performance evaluations.(Referred from work [2])

The id and other details are made to be autogenerated in both the databases. And the order in which the forums are created before the sub forums is maintained. And the Test has been carried out with 5000 rows to be inserted and 5000 of forums and sub-forums each with 5000 number discussions with comments numbered to 5000 as well.

And the time consumed by MySQL was observed to be 1010 seconds and then the time consumed by MongoDB is 3.3331 seconds respectively. And thus, the authors conclude that the performance of MongoDB is better than that of the MySQL.

Selection (Querying the database): The querying is one of the important aspects of the database and the authors in their work use two queries to carry out the evaluation. The first query is to select the discussions which are attended by a user and the date being a different one than the other. The other query is that to select the users involved in the discussions in a database and to count the number of discussions corresponding to each user. (The codes below are referred from work [2]).

The first query in MySQL is written as,

```
“ mysql_connect('localhost','root','');
mysql_select_db('routinie');
$query = "SELECT d.username, d.dtitle,
d.dcontent,
d.created, s.id, f.id
FROM discussion d
inner join subforum s on s.id =
d.subforumid
inner join forum f on f.id = s.forumid
where d.username = `andra`
and d.created <> '2014-11-27'";
$query_run = mysql_query($query);”
```

And the same query is written in MongoDB as,

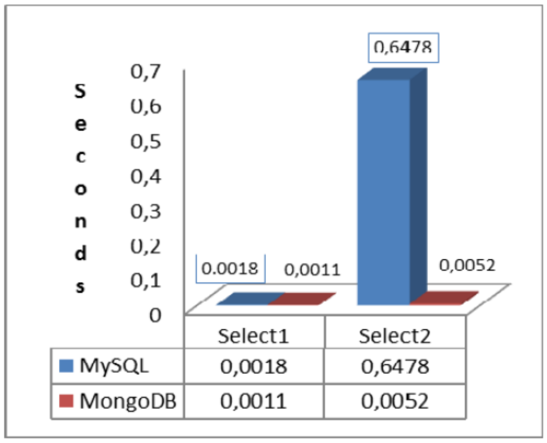
```
“ $m = new MongoClient();
$db = $m->selectDB('routinie');
$collection = new MongoCollection($db,
'categories');
$date = '1417091683';
$q = array(
'subcategories.topics.discussions.dauth
or' => 'andra',
'subcategories.topics.discussions.creat
ed' => array(
'$ne' => new MongoInt32($date))
);
$cursor = $collection->find($q);”
```

The second query can be written as,

In MySQL:

```
“ mysql_connect('localhost','root','');
mysql_select_db('routinie');
$query = "select u.id, u.username,
count(d.dtitle)
from users u
left outer join discussion d on d.userid
= u.id
group by u.id, u.username";
$query_run = mysql_query($query);
And in MongoDB:
$m = new MongoClient();
$db = $m->selectDB('routinie');
$collection = new MongoCollection($db,
'categories');
$user =
'subcategories.topics.discussions.dauth
or';
$find_users = $collection-
>distinct($user);
foreach($find_users as $find_user) {
$discussions = $collection-
>count(array('subcategories.topics.disc
ussions.dauthor'
=> $find_user ));”
```

The results are graphed as such that,



	Select1	Select2
🟦 MySQL	0,0018	0,6478
🟥 MongoDB	0,0011	0,0052

Figure 2. Results of performance evaluations. (Referred from work [2])

The observations show that the execution time for MySQL is 0.0018 seconds and the same in MongoDB is 0.0011 seconds. Then the second query took 0.6478 and 0.0053 seconds respectively. Thus the authors concluded that the performance of MongoDB was better than MySQL with respect to query operations.

Update operation: The update functions are checked by the authors in the two database systems with two update functions. One is to update the comment of a user specifically and the other is to update the user’s email address. (The codes below are referred from work [2]).

The first in MySQL is executed as,

```
“ mysql_connect('localhost','root','');
mysql_select_db('routinie');
$query = "update `comments`
set content = 'This is the new content.'
WHERE id = 10594
and username = `andra`;
$query_run = mysql_query($query);
and the following MongoDB syntax (Update
1- MongoDB):
$m = new MongoClient();
$db = $m->selectDB('routinie');
$collection = new MongoCollection($db,
'categories');
$criteria = array(
'subcategories.topics.discussions.comme
nts.cauthor'
=> 'andra',
'subcategories.topics.discussions.comme
nts.$id' => new
MongoId('5c936263f3428a40227908d5a3847c
0b');
);”
```

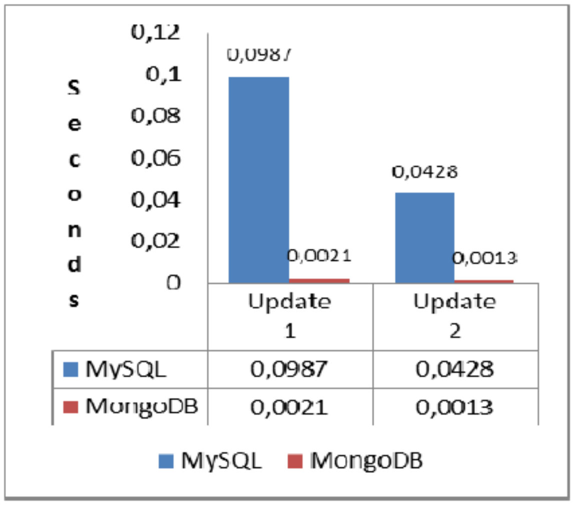
And the next query in SQL is proceeded as

```
“ mysql_connect('localhost','root','');
mysql_select_db('routinie');
$query = "update `users`
set email = 'andra.olah@gmail.com'
WHERE id = 1012";
$query_run = mysql_query($query);
```

And the query in MongoDB is written as,

```
$m = new MongoClient();
$db = $m->selectDB('routinie');
$collection = new MongoCollection($db,
'categories');
$criteria = array(
'users.$id' => new
MongoId('92cc227532d17e56e07902b254dfad
10');
);
$collection->update(
$criteria, array('users.email' =>
"andra.olah@gmail.com")
);”
```

And the output is recorded graphically as,



	Update 1	Update 2
🟦 MySQL	0,0987	0,0428
🟥 MongoDB	0,0021	0,0013

Figure 3. Results of performance evaluations. (Referred from work [2])

MySQL took 0.0987 seconds and then the mongoDB takes 0.0021 seconds and the second took 0.0428 seconds and 0.0013 seconds respectively. Thus the performance of MongoDB is better that MySQL even in update operation.

Delete operation: Then atlast comes the delete operation. Two delete queries are executed here , the first is to delete the comments by a specified user and the second is to delete the forum created by the specifiled user. (The codes below are referred from work [2]).

The query is written in MySQL as,

```
“ mysql_connect('localhost','root','');
mysql_select_db('routinie');
$query = "delete from comments where
username = `andra`";
$query_run = mysql_query($query);
And the same is wriiten in MongoDB as,
$m = new MongoClient();
$db = $m->selectDB('routinie');
$collection = new MongoCollection($db,
'categories');
$criteria =
array('subcategories.topics.discussions
.comments.cauthor'
=> 'andra');
$collection->update(
$criteria,
array('$unset' =>
array('subcategories.topics.discussions
.comments' =>
true)),
array('multiple' => true)
);”
```

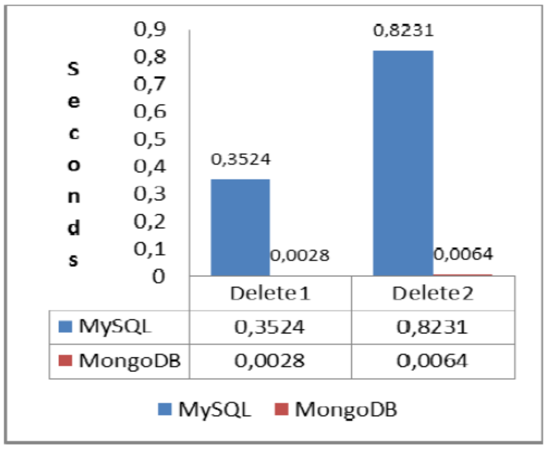
And the next query is written respectively as,

```
“ MySQL:
mysql_connect('localhost','root','');
mysql_select_db('routinie');
$query  =  "delete  FROM  `forum`  WHERE
username =`andra`";
$query_run = mysql_query($query);


and the following MongoDB syntax (Delete
2- MongoDB):

MongoDB:
$m = new MongoClient();
$db = $m->selectDB('routinie');
$collection  =  new  MongoCollection($db,
'categories');
$q    =    array('admins.username'    =>
'andra');
$cursor = $collection->remove($q);”
```

And the resultant graph is,



	Delete1	Delete2
🟦 MySQL	0,3524	0,8231
🟥 MongoDB	0,0028	0,0064

Figure 4. Results of performance evaluations.
The results show that the performance of MongoDB is better when compared to the MySQL with respect to Delete operation. (Referred from work [2])

3.2 Performance evaluations based on e-commerce data

In work [4], performance analysis is carried out using the comparison between PostgreSQL and MongoDB. The ERD of the relational data model is given below,

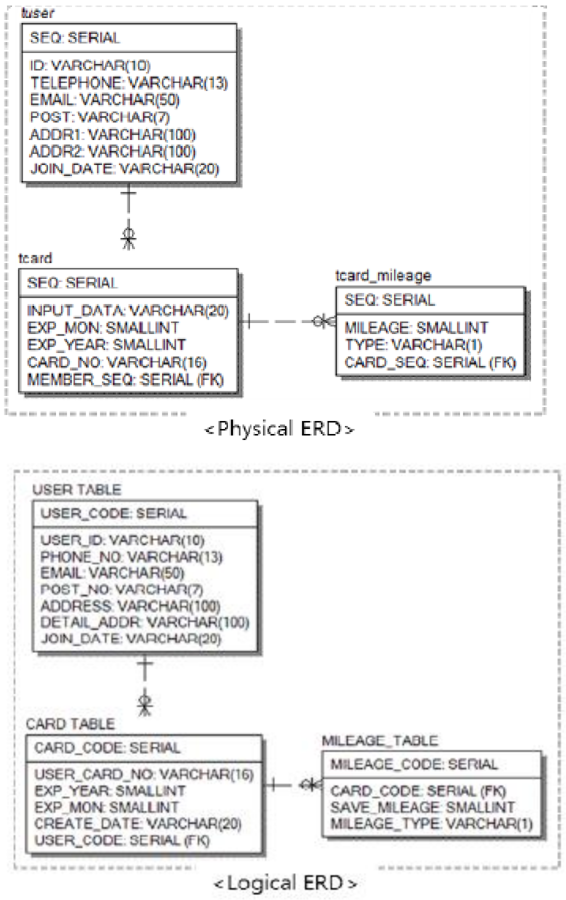


Figure 5. ERD relationship of data model. (Referred from work [4])

This diagram illustrates the relational database architecture. The MongoDB, which is an unstructured database has a model,

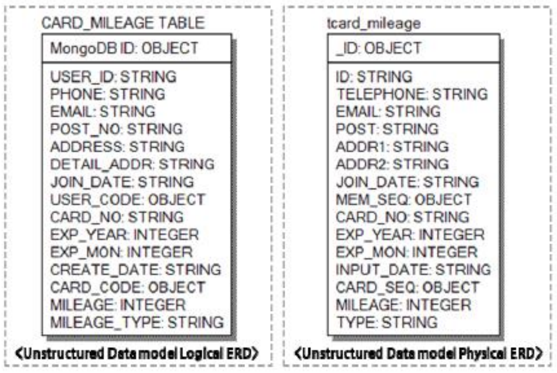
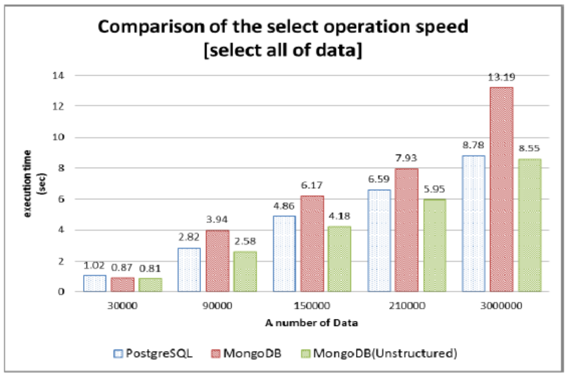


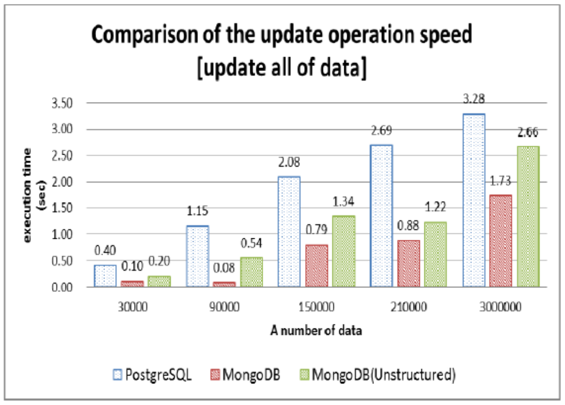
Figure 6. Model of MongoDB (Referred from work [4])

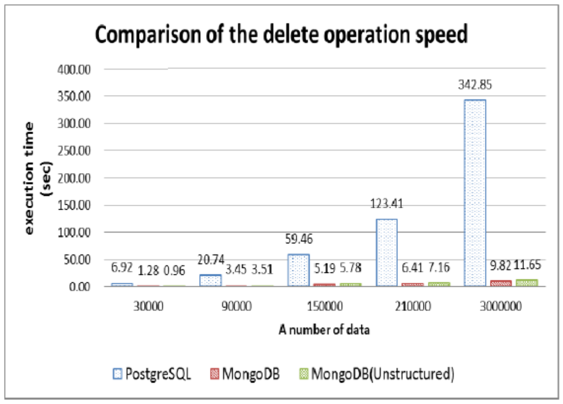
3.2.1 Comparisons based on performance

The authors in this paper have adopted the same factors for comparison as the previous. Insert, Select, Update, and delete operations have been performed and the results are observed.

Now the operations are performed individually in PostgreSQL and MongoDB and the results are marked in graphs as shown below.







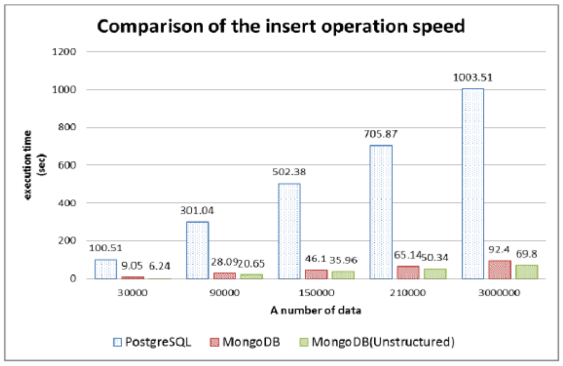


Figure 7. Performance results (Referred from work [4])

Thus, the observations prove that the performance of MongoDB is much better than PostgreSQL. It also proves that using an unstructured database helps in improving the performance of the database.

4. Migration of Data from Relational data bases to non-relational databases.

Now, we have reached the next part of the review which is the Data migration. Now that a new option is available to handle big-data which is the non-relational databases, there should be a way to transfer all the preexisting data in a relational database to the new non-relational databases. The works [5] and [6] , deal with this topic and many approaches are dealt with and experimented.

4.1 Transformation of model and migration of data from MySQL to MongoDB

In the work [5], the authors have proposed new methodologies and algorithms to carry out the migration of data from a preexisting relational database to a new non-relational database. They have used MySQL to represent the relational database and MongoDB to represent the non-relational database.

This work comes with the characteristics like, i) considering the characteristics of query and data related to the relational databases, ii) tags of descriptions and tags of action are used to create a transformation model with algorithms, iii) The results of the transformation model are used to carry out automatic migration of data form MySQL to MongoDB.

4.1.1 Challenges faced in transition of data

There is a difference between the relational databases and the non-relational databases based on the schema of both the systems. The schema of the relational database model is structural and its static where as that of the non-relational database model is flexible and can be customized by the user based on the requirement.

The other challenges that are faced are, Model related transformation challenge and migration of data related challenge. The challenges due to model arise due to the occurrence of redundancy or reduction in performance factors. The mongoDB has to customized based on the structure or model of the MySQL and thus with the fact that the Join operations are not supported in MongoDB it becomes difficult to establish the relationship between tables and if all the tables are connected with a relationship, then it may cause a redundancy.

The data model plays a major role is the structure of applications that carry out the data migration process. The code of the application must be changed for each database based on the model. At present, there is no application, which can perform this automatically. And the authors of this paper have proposed a method to resolve this shortcoming. The authors use the log files of the database to understand the characteristics of the databases and the tags of description are extracted. Then these tags are transformed into tags of action to help the transformation of the model of the system. These tags play a major role in understanding and adapting the model of the database.

4.2 Implementation

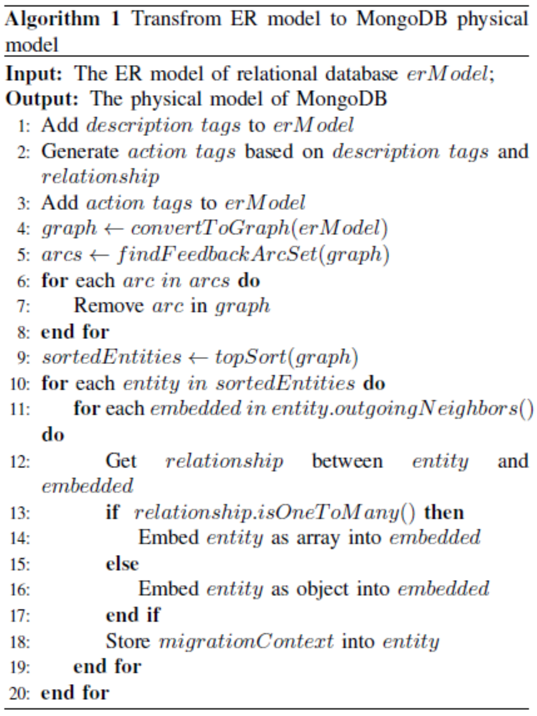
The absence of join operation in MongoDB has been compensated by aggregating different documents into a single document. But, the challenge arises as we saw earlier in determining the files or tables that ae to be aggregated. The authors have proposed four threshold values which the users can customize based on their requirement. These values determine which table or file to be aggregated. They can be classified as

i) “ Frequent Join” tag, which sets tag to the frequency values which cross the threshold.
ii) “Big size” tag: does the same with respect to the size of files.
iii) “Frequent Modify” tag: if many alterations are carried out in the same table.
iv) “Frequent insert” tag: relates the frequency of inserts used.

These tags cover the basic operations of the SQL. These characteristics are sufficient to derive a model with respect to the database. These description tags are transformed to tags of action. There is also a concept mapping that takes place.

MongoDB has the capacity to store the referred structured data. Thus, the relationships can be defined by using the references in MongoDB.

The problems discussed above are addressed with the help of an algorithm which has the capacity to convert the entity relationship model of a relational database to a non-relational database model. (The algorithm below is referred from work [5])



Algorithm 1 Transfrom ER model to MongoDB physical model

```
Input: The ER model of relational database $erModel$;
Output: The physical model of MongoDB
Add $description\ tags$ to $erModel$
Generate $action\ tags$ based on $description\ tags$ and $relationship$
Add $action\ tags$ to $erModel$
$graph \leftarrow convertToGraph(erModel)$
$arcs \leftarrow findFeedbackArcSet(graph)$
for each $arc\ in\ arcs$ do
    Remove $arc$ in $graph$
end for
$sortedEntities \leftarrow topSort(graph)$
for each $entity\ in\ sortedEntities$ do
    for each $embedded\ in\ entity.outgoingNeighbors()$ do
        Get $relationship$ between $entity$ and $embedded$
        if $relationship.isOneToMany()$ then
            Embed $entity$ as array into $embedded$
        else
            Embed $entity$ as object into $embedded$
        end if
        Store $migrationContext$ into $entity$
    end for
end for
```

This algorithm works by adding the model of ER with the appropriate tags of description. With this relationship, the tags of action are generated. The process by which the tags of action is generated is provided in algorithm 2. Then the ER model undergoes a conversion to form the diagraph G.

Then the entities are put together with the help of the order received and the model for MongoDB is generated. Based on the relationships, whether it is one to one or one to many, the entity is chosen to be an object or an array and then embedded. (The algorithm below is referred from work [5])

Algorithm 2 Generating action tags

```
Input: The ER model of relational database with description tags $erModel$;
Output: ER model with action tags
1:  function CAN_EMBED_TO_OTHER($entity$)
2:      if $entity$ has Big size tag or Frequent modify tag or Frequent insert tag then return false;
3:      end if
4:      return true;
5:  end function
6:  for each $relationship$ $in$ $erModel$ do
7:      switch Type of $relationship$ do
8:          case $OneToOne$
9:              if CAN_EMBED_TO_OTHER(child entity of $relationship$) then
10:                 Add $Embed$ $child$ $entity$ $tag$
11:             else if CAN_EMBED_TO_OTHER(parent entity of $relationship$) then
12:                 Add $Embed$ $parent$ $entity$ $tag$
13:             else
14:                 Add $Reference$ $tag$ to $relationship$
15:             end if
16:         case $OneToMany$
17:             if $relationship$ has $Frequent$ $join$ $tag$ and CAN_EMBED_TO_OTHER(child entity of $relationship$) then
18:                 Add $Embed$ $child$ $entity$ $tag$ to $relationship$
19:             else
20:                 Add $Reference$ $tag$ to $relationship$
21:             end if
22:         case $ManyToMany$
23:             Add $Reference$ $tag$ to $relationship$
24: end for
```

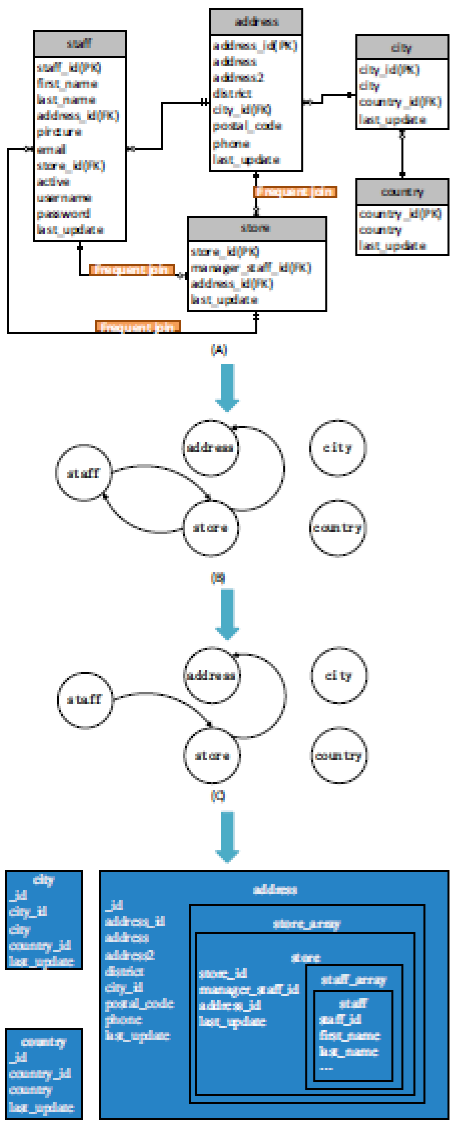


Figure 8. Transition structure (Referred from work [5])

4.3 Migration of data.

The new model of MongoDB that has been created after the transformation has a hierarchy data structure. There is resemblance to the physical model of the same. The collections in the MongoDB are represented by the child nodes. This relationship replaces the need for the join operation. This model is shown in the figure below.

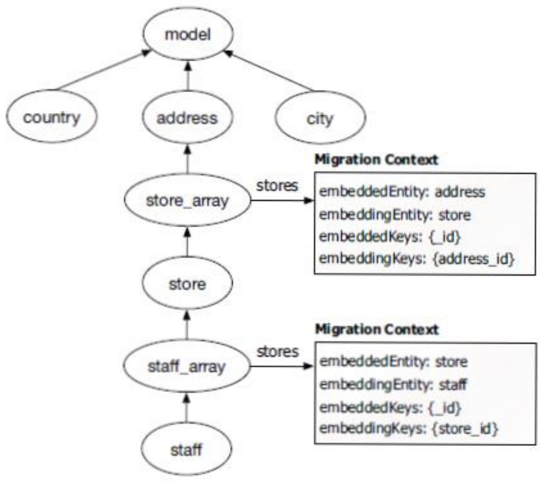
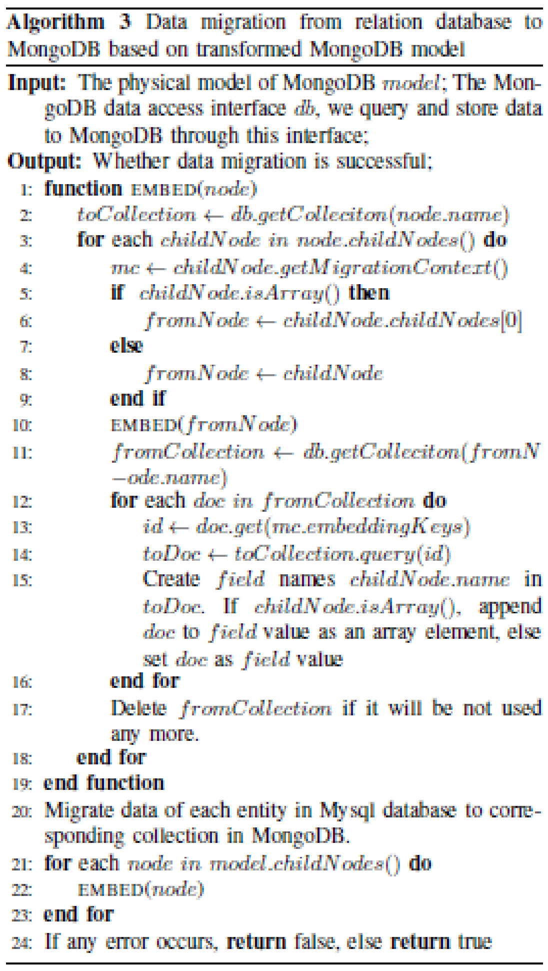


Figure 9. Migration model (Referred from work [5])

The algorithm used for the migration is specified in algorithm 3 (The algorithm below is referred from work [5])



Algorithm 3 Data migration from relation database to MongoDB based on transformed MongoDB model

```
Input: The physical model of MongoDB $model$; The MongoDB data access interface $db$, we query and store data
    to MongoDB through this interface;
Output: Whether data migration is successful;
function EMBED($node$)
    $toCollection \leftarrow db.getColleciton(node.name)$
    for each $childNode$ in $node.childNodes()$ do
        $mc \leftarrow childNode.getMigrationContext()$
        if $childNode.isArray()$ then
            $fromNode \leftarrow childNode.childNodes[0]$
        else
            $fromNode \leftarrow childNode$
        end if
        EMBED($fromNode$)
        $fromCollection \leftarrow db.getColleciton(fromNode.name)$
        for each $doc$ in $fromCollection$ do
            $id \leftarrow doc.get(mc.embeddingKeys)$
            $toDoc \leftarrow toCollection.query(id)$
            Create $field$ names $childNode.name$ in $toDoc$. If $childNode.isArray()$, append
            $doc$ to $field$ value as an array element, else set $doc$ as $field$ value
        end for
        Delete $fromCollection$ if it will be not used any more.
    end for
end function
Migrate data of each entity in Mysql database to corresponding collection in MongoDB.
for each $node$ in $model.childNodes()$ do
    EMBED($node$)
end for
If any error occurs, return false, else return true
```

5. Data Storage Mechanism in SQL and NoSQL Database.

5.1 SQL or Relational Database

Relational databases store the data in the form of tables where each table represents an object, where in each table has rows representing a record and each column represents a field. These tables are linked with each other with the help of foreign keys or using common columns. The most important design aspect of relational databases is the normalization of the schema. **1. First Normal Form:** It is to eliminate the groups which has repeating data by creating a new table for repeating data. **2. Second Normal Form:** If the set the values are repeating for multiple records then they are moved to new table and are linked with a foreign key. **3. Third Normal Form:** If fields are found to be independent of primary key then they are removed and moved to a different table. For a relational database, it is necessary that they satisfy the above 3 conditions. Another important aspect of Relational databases in ACID properties. **Atomicity (A):** In a transaction either all parts or none should be complete. **Consistency (C):** For a database, its integrity has to be preserved and it should not be left in an invalid state after transaction. **Isolation (I):** Every transaction must be isolated from each other since data involved in one transaction should not affect other transaction. **Durability (D):** The transactions made must be durable.

With recent development in technology traditional storage performed in SQL becomes ineffective. Cloud computing technology is an effective storage mechanism where the data is spread across multiple servers for storage. Using SQL database is not suitable for database operations since it requires joining of multiple tables which are large in size and it leads to a very complex problem. Storing unstructured data such as social media posts has grown rapidly. SQL is effective only when the data is stored in the form of structural format whereas it cannot do the same on unstructured data. It requires structure to be specified at the early stages. Any changes to the information schema will be a time-consuming process.

5.2 NoSQL Databases

The most important aspect of NoSQL database is that the will not need and predefined schema, each records can have different fields and hence they can be called as Dynamic Schema. Other important property of NoSQL database is it supports replication on multiple servers so that if one server goes done the replica server can become primary server. All the servers work in synchronized for transactions to eliminate errors. ACID properties followed by relational database is not fully guaranteed in NoSQL databases. This is done to achieve horizontal scalability and recovery in case of a failure.

Storage in NoSQL database is categorized as below:

5.2.1 Key-Value Storage Databases

Key-Value storage mechanism is a powerful model which provides Application Programming Interface (API). This type of data storage is schema less and data is some kind of programming language data type. Key-Value pair will have two parts, where key is a string and value holds the actual data. The stores are similar to hash tables where keys are used as indexes hence making it faster than RDBMS. This is very simple and it allows the user to request values based on key values. This method is highly scalable. High concurrency, fast lookups and options for mass storage is provided by key-value stores. The databases which use this storage technique are Amazon Dynamo DB, RIAK.

5.2.2 Column-Oriented Databases

Column-Oriented Database structure in NoSQL are hybrid row/column store and are different from pure relational column databases. Though both SQL and NoSQL databases have the same concept of column-by-column storage of columnar databases and columnar extensions to row-based databases, column stores do not store data in tables but store the data in massively distributed architectures. In column stores, each key is associated with one or more attributes (columns). This type of storage offers high scalability in data storage. The data which is stored in the database is based on the sort order of the column family. Column oriented databases are suitable for data mining and analytic applications, where the storage method is ideal for the common operations performed on the data. The most commonly used databases are Big Table, Cassandra.

5.2.3 Document Store Databases

Document Store Database stores data in the form of documents which is the minimum independent storage unit. Documents will be of standard format such as XML, PDF, and JSON. In document storage each of the document can have similar or dissimilar data. Every document in a database will have a unique key which is used to identify documents. Document storage is slightly complex compared to key-value pair storage. A group of documents will be split in to chunks where every chunk will have a share key and all the chunks will be uniformly distributed across the cluster of servers. This will lead to equal distribution of data among the servers hence cause workload balance. This model should not be used if the database has a lot of relations and normalization. The main application of this type of storage is seen in content management, blog software. Few databases which use this approach are MongoDB, CouchDB.

5.2.4 Graph Databases

In graph databases the storage of data is in graph from with nodes and edges. Nodes represent objects and edges represent relation between the objects. Index free adjacency technique is used where the node will have direct pointers which will be pointing to the adjacent nodes. It is easy to traverse through millions of records using this method. The main factor here is connection between data, and this is a very efficient storage mechanism with no schema. To process a query traversing is performed through graph and hence it is faster. This data storage is ACID compliant. This type of data store is widely used in social networking applications, bioinformatics, content management, security access control, recommendation software and cloud applications. The databases which follow this storage structure is Neo-4j.

5.2.5 Object Oriented Databases

Object oriented data base will have data stored in the form of object similar to object oriented programming language. It's a mixture of object oriented principles and database principles. It supports all the features of OOPS such as data encapsulation, polymorphism and inheritance. Table, tuples and columns can be related to class, object and attributes. Here access to the data is much faster since the data retrieving is done using pointers. Implementing this will make software support agile. These databases are being used in scientific research, CAD and telecommunication. It is advisable not to use this model when the relations are simple and small. Db4o database follows this model.

5.3 Normalization and Embedding in NoSQL Database.

Normalization on RDBMS is done using First, second and Third Normalization forms mentioned above. There are additional normalization forms as well such as Bouce-Codd Normal form, Fourth Normal form and Fifth normal form. These forms are not commercially applied. Many designers have experienced that tight application of these normalization forms will lead to poor system performance. Hence de-normalization is preferred over normalization to improve the performance. De-normalization is a process which reduces the number of tables that are accessed frequently to reduce the processing time.

Relational database is designed such that it minimizes data redundancy and also storage space. The main reason for normalization is to increase speed, data integrity and efficient usage of space. A comparison study is carried out to see the difference in performance between normalized database and demoralization in NoSQL MongoDB.

In MongoDB the document storage structure is used which is a collection key-value pair. Record of SQL databases is referred as documents in NoSQL database. Tables of RDBMS is referred as Collections in MongoDB. The joins in MongoDB is done by referencing and linking.

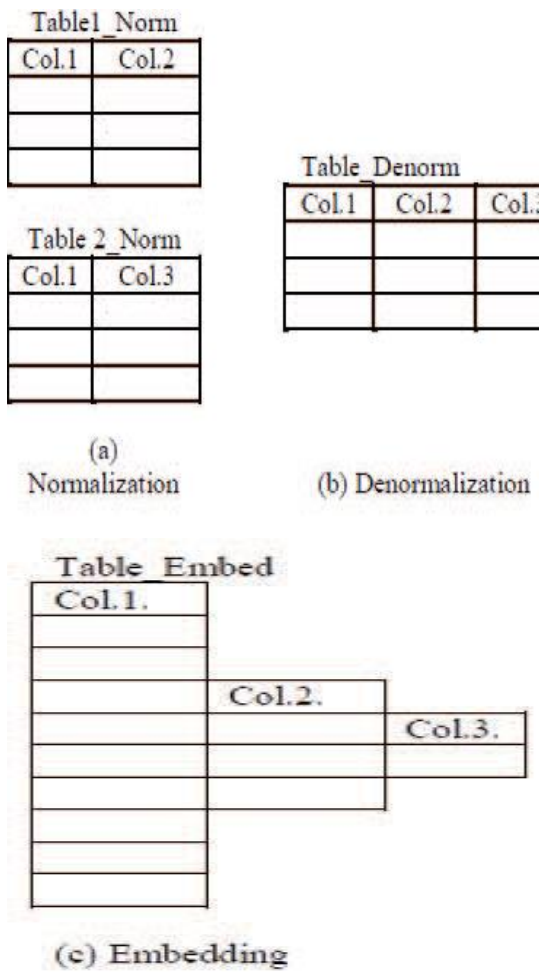


Figure 10. (a) to (c) Data Modeling in Database

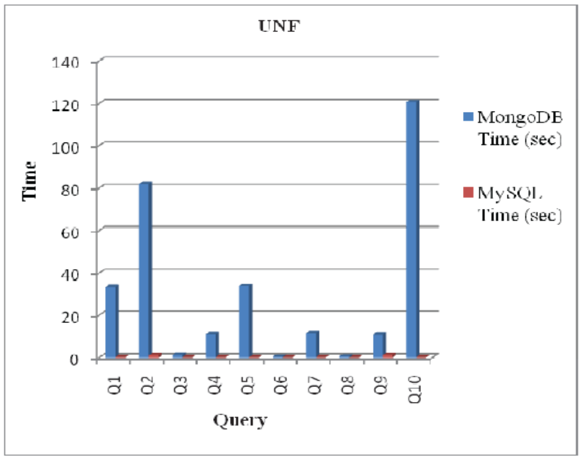
5.4 Experiment Result

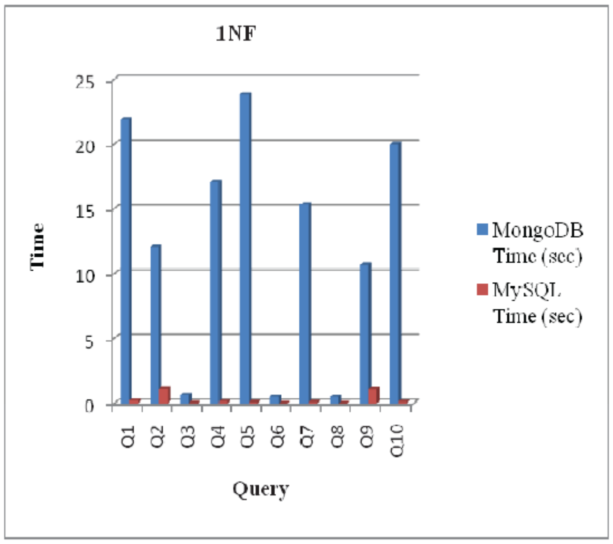
The experiment result is divided to two parts Part I and Part II.

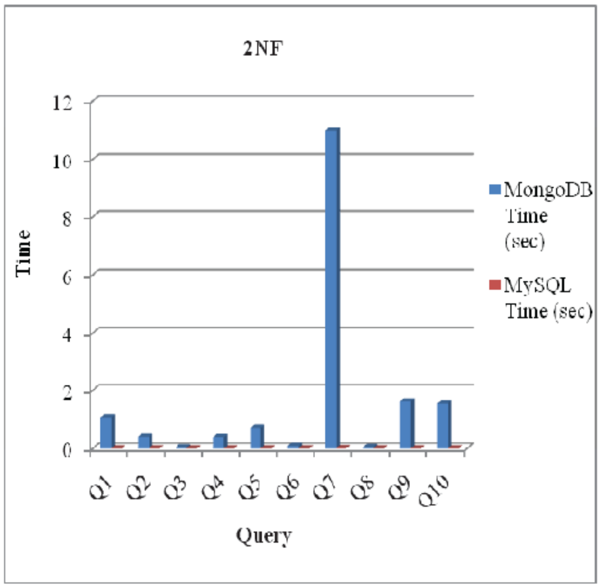
Part I consists of Open source MySQL database which incorporates normalized schema. Part II represents the results obtained by modelling highest level of embedding which is six in the present model. The results of both the schemas are compared. A set of queries are designed for both the schemas and query possessing time is measured.

A. Part I – Normalization

Set of common queries are executed for UNF, 1NF, 2NF, 3NF and embedded models for MySQL and RDBMS. (a), (b), (c), (d) show un-normalized query execution time. Each of the graph shows the difference in query execution for un-normalized and normalized 3 levels of NF. The time taken by SQL is represented in red and the time taken by MongoDB is represented in blue. Ten queries Q1 to Q10 are applied on databases. Y-axis represents query execution time in seconds. X-axis represents query applied on the databases.







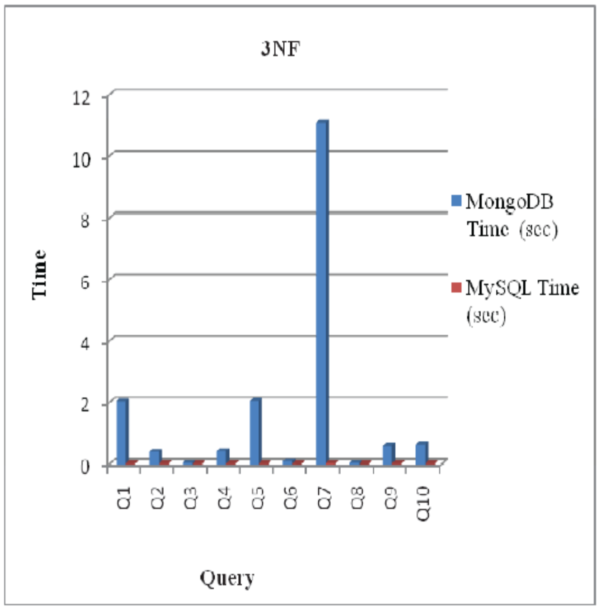


Figure 10. (a) to (d) Query Execution in Normalization Part

B. Part II – Embedding and Comparison with Normalization

Part II has data in the form of embedded documents to represent the relationship between the entities. The embedding level used here in this experiment is 6 which means that 6 documents are embedded in one another. The relation type is one-to-many.

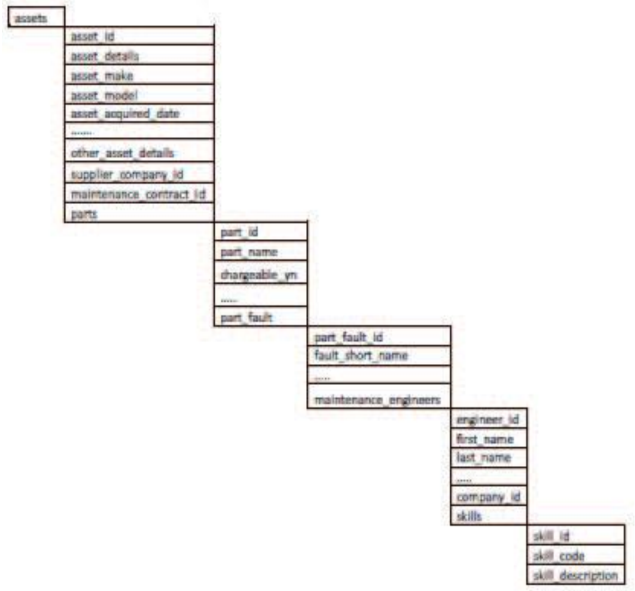


Figure 11. Embedded Document Structure

The graph below represents the execution time taken by embedded physical system and MongoDB in Normalized format. The execution time of each query for both modes is compared.

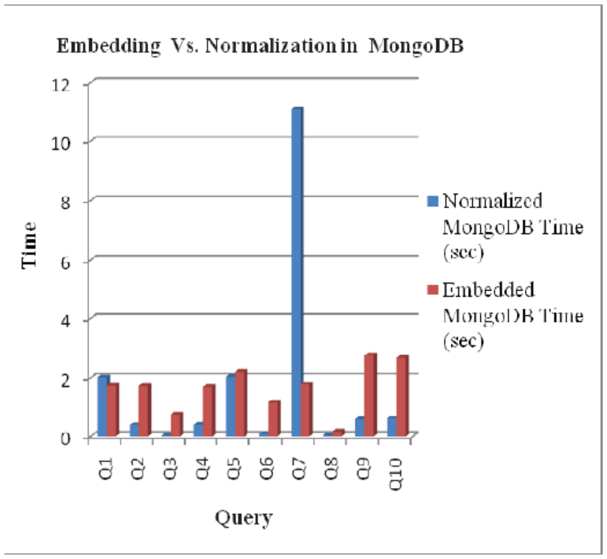


Figure 12. Query Execution in Embedding Vs. Normalization Part

It is observed that MongoDB gives better performance for all the queries. The embedded MongoDB performance was very high compared to normalized MongoDB data model and also the result is found to be much more consistent.

5.5 Dynamic Data Balancing in MongoDB.

In MongoDB the data is uniformly distributed across multiple servers so that workload balance is achieved and also processing of the data is faster which increases performance. However there are few drawbacks when there are hotspots in data since all the data access cannot be done in the same pattern. This study is mainly to analyses the dynamic work-load balance done in MongoDB to find the hotspots of data. Then dynamic data migration is performed to diffuse the hotspot across shared servers. This is followed by load balancing across servers which will lead to improvement in performance.

Application of RDBMS on big data is not a feasible approach. This is mainly due to the following reasons. First, accessing large amount data in RDBMS with millions of records is highly inefficient. Second, though the data can be assessed it is very weak in terms of data scalability due to its costly distributed storage solution which makes it not a suitable mechanism for big data applications.

This paper proposes a workload-driven approach to dynamic data balance in MongoDB. This approach is found to be more efficient when applied to big data applications.

5.6 DESIGN OF WORKLOAD-DRIVEN APPROACH

A. Architectural Design:

MongoDB mainly has 3 parts. They are Routers, Clients and interface of MongoDB and they are in-charge of accepting data, manuplating requests and locating target by querying Config Server and dispatching the requests to the Shard server. Shard server provides dynamic distribution of data, and supports dynamic migration of data chunks. Config servers maintain mapping one between chunks and their hosted server and the other between chunks and id ranges.

The figure below consists of original components of MongoDB except the one written in italic font. These new extended parts of MongoDB are used to perform workload-driven data balance.

These add-ons mainly have two parts Monitor and Analyzer. Monitor collets real-time status of Shard Server and Analyzer identifies the status of Shard server and based on data collected and this is sent to balancer. An upper bound is configured for each computing resource and when the value crosses upper bound analyzer will identify the local hotspots of data analyzing log file.

B. Status Identification of shared Server

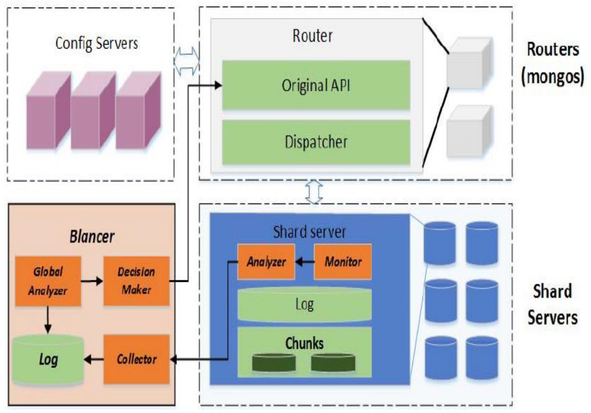


Figure 13: The architecture of workload-drvien approach to dynamic data balancing in MongoDB

In order to achieve workload-dynamic data balancing it is necessary to know the status of all the shard servers. For simplicity, two load types are defined normal and overloaded. The reason for overloading is mainly due to insufficient CPU resources. The status of shard server is calculated using average of recent historical values collected by Monitor.

The following function is used to determine the load type of shard server

L(S) = overload $U_s > \text{AC}P_u$

L(S) = normal otherwise

Here U_s is CPU utilization of shard server S, ACP_u is the acceptable predefined utilization of CPU.

C. Decision on Data Migration

Data migration is done to rebalance the workload across Shard Server. Before rebalancing a server has to be identified which can receive load from the overloaded server. Collector component of Balancer collects the status of all the Shard server and store it in Log Module. This is used by Global analyzer to choose the destination server.

The destination server is chosen by Global analyzer by Algorithm 1. Choosing destination is not only based on how light it is but it is based on how close to ideal level they will be after data migration. Algorithm 1 is used to analyze CPU utilization whereas Algorithm 2 will find the suitable destination. The result of these algorithms can have two different possibilities. If a feasible solution is found then all the chunk will be moved to destination server. If appropriate destination is not available then a suggestion will be given to optimize Shard key.

During data migration a few rules of MongoDB may be violated but this will not affect the running time since the balancing is performed on workload not the amount of data in MongoDB.

Algorithm1: Decision on Destination Shard Server
Input: $\{\{S_j\}, \{U_{Si}\}, \{S_j\}$ represents all shard servers
Output: pair (set, S_j) a pair contains destination shard and chosen chunks

```
1.  for each $S_j$ in {S}
2.  if $U_{Si}$ <= ACP$_u$
3.      add $U_{Si}$ into $\{U_S\}$;
4.  sort ($\{U_S\}$)
5.  for each $U_{Si}$ in $\{U_S\}$
6.      Set<C> set;
7.      if $U_{Si} + U_{So} \leq 2 * U_{ideal\_level}$
8.          $U_{except} = U_{so} - U_{ideal\_level}$
9.          $U_{threshold} = U_{ideal\_level} - U_{sj}$
10.         IfChooseChunk($\{C_{sj}\}$, $U_{except}, U_{threshold}$,Set)
11.             Return pair(set, $S_j$)
12.         end if
13.     end if
14. end if
15. return pair(null, null)
```

Algorithm 2: Find feasible solution.
Input:
$\{C_{sj}\}, Set < C > set,\ U_{except}, U_{threshold}$
Output: a set containing all chosen chunks.

```
1.  Sort($\{\{C_{sj}\}\}$);
2.  for each $C_i$ in $\{C_{sj}\}$
3.      if $U_{ci}$ < $U_{threshold}$
4.          add $C_i$ into set;
5.          $U_{except}$ = $U_{except}$ - $U_{ci}$
6.          $U_{threshold}$ = $U_{threshold}$ - $U_{ci}$
7.      end if
8.      if $U_{except}$ < 0
9.          return true;
10.     end if
11. end for
12. return false;
```

D. Experimental Results

1. Improvement in performance

From the results it was found that mean response time increased about 20 times and the utilization of CPU was beyond 95% with 5 times increase in workload. The result of comparison performed on MongoDB and Data Balancer is shown in the below figure below. The second figure shows the increase in mean response time with effective utilization of CPU.

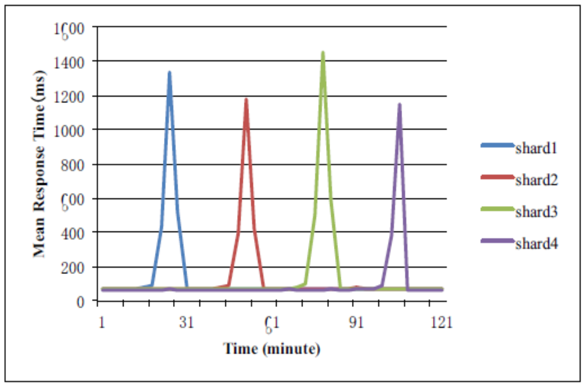


Figure 14: The mean response time on 4 shard servers

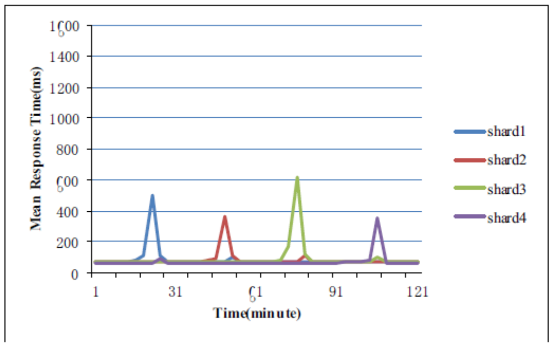


Figure 15: The mean response time on 4 shard servers with Data Balancer

Improvement of utilization of CPU

The utilization of CPU for MongoDB with Data balancer and without Data balancer is represented in the form of graph in which we see an increase in utilization of CPU on target Shard Server simultaneously occurred with the decrease of utilization of CPU on overloaded Shard Server. With Data Balancer, the utilization of CPU is improved by the shortened time spans of extremely high utilization of CPU on overloaded Shard Servers and increased utilization of CPU on spare Shard Servers.

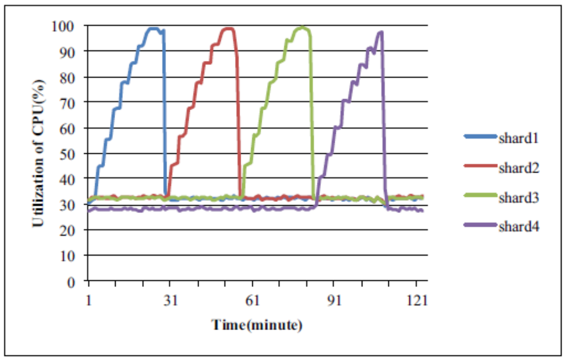


Figure 16: Utilization of CPU in MongoDB without Data Balancer

6. Application Level Optimization of Big-Data Transfers.

Now, we move on to the second art of the review. The topic in which the whole world has laid its eyes on is Big Data. The present world generates huge amount of data each day due to the advent of internet. It has become a necessity to process, store and handle this data efficiently. One of the major process involved in data management is data transfer. There are several factors that affect the data transfer. But, this concept deals with customizing the computer organization factors like pipelining, concurrency, and parallelism to enable fast and precise data transfer. The application level optimization of Big Data transfers can be achieved by this method.

The size of the files being transferred plays a major role. The heterogeneous nature of file sizes can cause loss of bandwidth during the transfer process. This can be avoided by optimizing the parallelism, pipelining and concurrency factors in the TCP.

The usage of pipelining in this transfer process gives a control which is single and the same data channel of data is used for all the files as they are sent in a continuous manner without waiting for the "transfer complete" command from the previous file transfers. When parallelism is adapted, it sends the different portions of a single file through channels of data which are parallel at a constant time. The concurrency is characterized by sending multiple files rather than the same file at the same time through different channels. Without using the benefit of all these three factors a file can only be sent after a "transfer complete" command is received.

Even though these factors can be used to enable fast data transfers the value or the amount of parallelism, pipelining and concurrency plays a major role. If their values go beyond or fall short of the required values, then the results may not be fruitful. Hence assigning the required values of pipelining, parallelism and concurrency plays a major role. These papers talks about the best ways and models used to assign the required values. They also address the requirement of pipelining, parallelism and concurrency and the factors like file sizes number of files and their impact on the techniques used. Other topics like the necessity of pipelining for all transfers, lower limit of data in a chunk with respect to pipelining of different levels, differences between performance of data channel caching and pipelining, when parallelism can be advantageous, setting levels of parallelism and concurrency after the optimization of pipelining, what is the upper limit of parallelism, advantages of concurrency over parallelism, will concurrency be sufficient without parallelism or pipelining, the effect of capacity of the network on the levels of optimization of parallelism and concurrency are also discussed. These topics highlight the necessity of each of the three factors and their importance in their own way.

Two algorithms are proposed to assign values to parallelism, pipelining and concurrency. The algorithm which was proposed first deals with the approach which is adaptive and gradually reaches the upper limit of bandwidth of network. The second adapts a more aggressive approach in using concurrency. The adaptive algorithm is called the recursive chunk division for optimal pipelining and the aggressive approach is called the optimal parallelism-concurrency-pipelining. The functioning of these two algorithms, their usage in different places, the advancements and improvisations that can be made in these algorithms to improve data transfers will also be discussed and analyzed.

An option to replace TCP with UDT for data transfers with the optimization techniques is also proposed. The advantages of using UDT with these application level optimization techniques than TCP is also discussed. At last the experimental verification of the optimization techniques and the results will be analyzed. The relation to these techniques and the concepts used in parallel processing of high level data flow in super computers will be highlighted and other aspects and improvements in these techniques will also be discussed.

Work [12] is chosen to be the primary paper and discussed in depth in the upcoming content. Chapter 7 deals with [12] in detail and the reviews of other papers [13], [14], [15], and [16] are written in chapters 8, 9, 10, 11, and 12 respectively. All these works are related to the topic which we were talking about and they differ in the methods of application and implementation. At the end of this survey, a clear understanding about this technology and the advancements made in this field is precisely gathered.

7. Importance of pipelining in all data transfers

We will start with the survey of work [12], When we talk about implementation of pipelining in data transfers, a natural question arises that whether the pipelining is required for all data transfers and whether it is being used profitably in all data transfers. The answer to this ambiguity is an obvious yes, because pipelining is useful when small files in large numbers are transferred. But there is a consideration which has its own importance, it is the bandwidth delay product (BDP). It acts as a threshold point and determines the level of pipelining used with respect to the dataset. High level of pipelining is not required when a dataset exceeds the bandwidth delay product point. In our case where we are using heterogeneous file sizes it becomes a necessity to classify the dataset into two part, one's file size below the BDP and the other's file size above BDP and we can concentrate on the part whose file size is less than the BDP. This is the part where the various pipelining levels has the chance to affect the throughput. The BDP can be calculated by taking the bulk TCP disk-to-disk throughput for a single TCP stream for bandwidth and average RTT for the delay.

7.1.1 Effect of file sizes on the optimal pipelining level

The file size plays a major role in determining the optimal pipelining level. This is more important in cases of long RTT networks. Figure 17 and figure 18 shows the experimental results. They interpret results of data sets consisting of 128 files of different sizes present in an environment of emulab setting with 1Gbps network bandwidth and 100 ms RTT and it is a disk-to disk GridFTP transfers. Tshark which is a network analyzer tool has been used to calculate the data about the number of bytes that has been sent or received in each RTT. The control channel conversation and the data channel conversations are isolated from each other and the data channel conversations are presented e the graphics. We can observer that irrespective of the levels of pipelining, all of them have a slow start irrespective of their file size. The highest number of bytes reached by a specific file size acts as the crucial point.

$$B = FS *(pp+1)$$

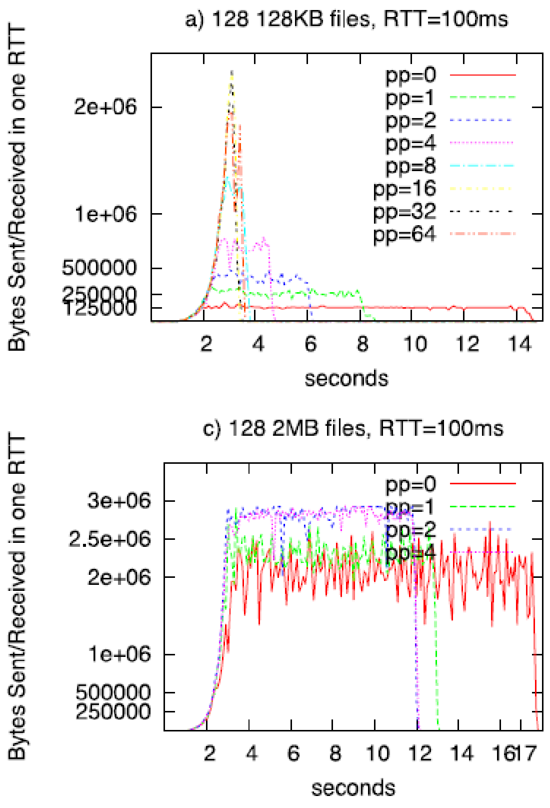


Figure 17. Influence of size of file and level of pipelining on sent and received bytes. (From Reference [1])

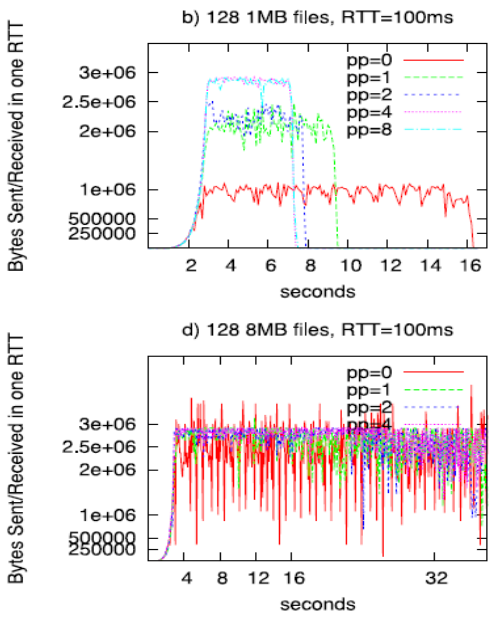


Figure 18. Influence of size of file and level of pipelining on sent and received bytes. (From Reference [12])

The number of bytes sent or received in one RTT can be referred as B, FS is the file size and pp is the pipelining level. The linearity between the number of bytes and the pipelining level exists only till the BDP point. Beyond the point there will be a logarithmic increase. Thus, we can calculate the optimal pipelining level as

$$PPopt \simeq \lceil BDP/FS \rceil - 1$$

7.1.2 Effect of number of files on optimal pipelining level

The number of files have direct effect only on the throughput but not the optimal pipelining level. As we said before the we can interpret from the figure 1 and figure 2 that the same characteristics are portrayed by the slow start curve corresponding to different pipelining levels. Thus, the increase in number of files only increases the length of the proportions in which transfer of data occurs at full capacity in the network. Figure 19 represents a comparison between the disk-to-disk transfers with different number of files with range [64-256] with different file sizes at the same testbed settings as before. We know that in almost all the cases the pipelining level will be the same even with different number of files and we can also observe that the increase in number of files increases the total throughput. But for the throughput which is instant at a time then they remain the same.

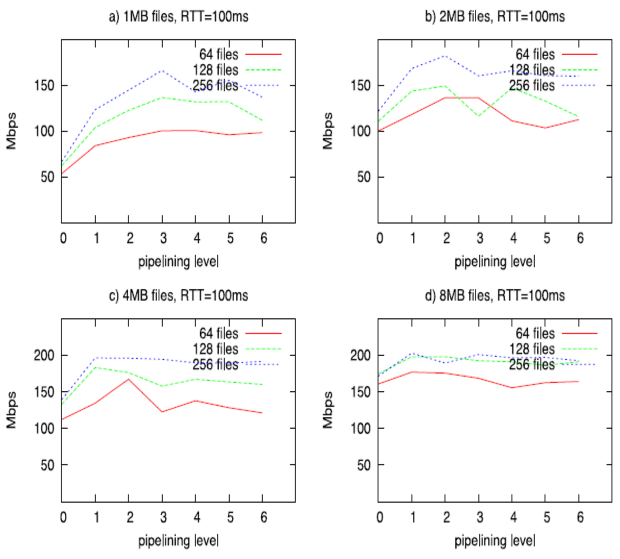


Figure 19. Influence of number of file on the optimal level of pipelining. (From Reference [12])

7.1.3 Minimal amount of data required in chunk for effective pipelining levels

The only difference between the files is the highest number of bytes that has been sent or received in a RTT at pipelining of different levels and we know that the file sizes consist of similar characteristics of slow start phase, thus the chunk should have a minimum file numbers to get the transfer through the slow start phase. Failing to be so will not bring improvement to the throughput with file sizes that are small. The minimum size of the chunk required can be calculated from a model proposed in [27], has been implemented in the proposed algorithm used for optimization in section 8.7. The concept of statistics and mining has been used in the model which gives the minimal size of the data that must be transferred and the throughput is predicted accurately.

7.1.4 Effect of optimal pipelining level setting between long and short RTT

We have seen before that the highest amount of bytes sent or received in a RTT in file sizes below the BDP gets affected by the level of pipelining and size of files. Thus, the RTT affects the process indirectly as the BDP calculation uses the RTT. There is considerable loss in the effect of pipelining for local area and metropolitan area network. The span of the document gets to be bigger than the BDP esteem and pipelining is just used to close the handling overhead and control channel discussion spaces between the exchanges. It shows that the throughput does not increase linearly with the levels of pipelining and the maximum throughput can be achieved by setting up the pipelining level to one or two. Figure 20 shows the results of throughput for disk to disk transfer with 128 number of files of different sizes with the predetermined settings. The optimal pipelining level is independent of the sizes of the file and pipelining levels of one or two can be used to gain maximum throughput.

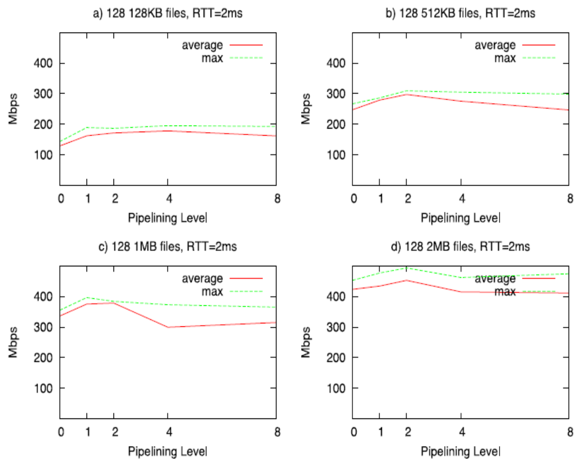


Figure 20. Influence of pipelining on short RTT networks. (From Reference [12])

7.1.5 Performance comparison between Data Channel Caching and Pipelining

We have seen before that the files whose size is larger than the BDP, pipelining of large numbers is not needed whereas the level of one is sufficient for maximum utilization of the BDP. There is an overlap of the data transfers and the control messages in pipelining. The transfer of files of small sizes in multiple numbers acts as a data transfer of large size. (Figure 17 and Figure 18). Data channel caching is the process by which a single channel can be used for transfer of files in multiple numbers, there can arise a question that whether the same effect can be achieved in data channel caching. Yes, it can be achieved but it has few limitations. It does not allow the overlap of data transfers with messages of control channel and it waits to start a new transfer until the existing one is finished**.** Figure 21 gives a comparison of both the techniques and a model without optimization. Here local area transfer of 128 1 MB files at 1 Gbps network with approximately 25 K BDP. Figure 21a, case without optimization is used, each 1 MB is carried through back to back transfer with less number of RTT on its path. The figure also shows the effect caused due to the window size growth for all the files. In (Figure 21b) where the application of data channel caching is shown, in which there exist the distance between each transfer but only the file which is first shows the effect of the growth of the size of the window. The rest of the file transfers begin with the BDP, which is possibly the large sized window.

Figure 21c shows the 1 level pipelining, there occurs a disappearance of the distance caused by the overlapping of transfer of data with overhead of processing and messages of control channel. There is continuous transfer occurring with the maximum size of the window at 25 K. Similarly, for other larger file sizes the distance remains the same between each transfer. It can be interpreted from the results that the best option would be pipelining rather than caching of data channel even with the small differences in performance. In the case of bandwidths which are large, RTT which are long networks the windows of large size occurring due to the caching of data channel will support the transfers which are consecutive and the drawback will be the distances which are small that remains. By using the pipelining, we can remove this small distance.

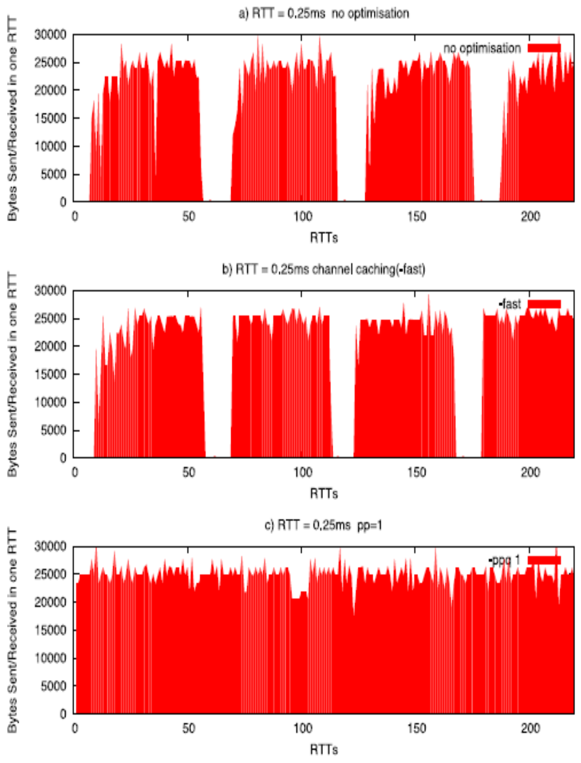


Figure 21. Result of caching of channel of data vs pipelining (From Reference [12])

7.1.6 Effect of setting levels of concurrency and parallelism after pipeline optimization

From Figure 17 and 18, we can say that a pipelined data set with transfer which is optimized is viewed as the transfer which is like that of big files which does not consist an optimization techniques. Thus, it is a good move to set the pipeline optimization level so that it is easier when it comes to finding things and setting parallelism optimization and levels of concurrency.

7.2 Advantages of Parallelism

The size of the buffer in the system must be set to a smaller value with comparison to the BDP in order to make the parallelism more advantageous. This can be predominantly noticed in the networks of RTT with bandwidths which are large. When there is need to transfer files, which are large it is always better to use the parallelism. Whereas with the files of small sizes there is no much effect due to the parallelism but it can be combined with pipelining with a value at which it does not affect the effect of pipelining. It can be executed when files and chunks are small in size. In Figure 23 the effect mentioned above can be clearly seen.

There is a negative effect when parallelism is applied with pipelining or small file sizes and the same changes to positive by gradual increase in the file size. Figure 24 shows similar such effect for the file sizes from 128 0 256 MB. The effects caused by parallelism is recorded while there is an increase in the file numbers for the files of 1 MB that are transferred with the process of pipelining. Thus it can be seen that there is a changeover form negativity to positivity due to parallelism. But, there occurs an increase in parallel stream numbers and the increase is constant and causes the size of the data to be divided further into many streams thus the case with data size which are big fails to show the positive effect. This can help us to make a conclusion that files of bigger size and many number can make us of parallelism better.

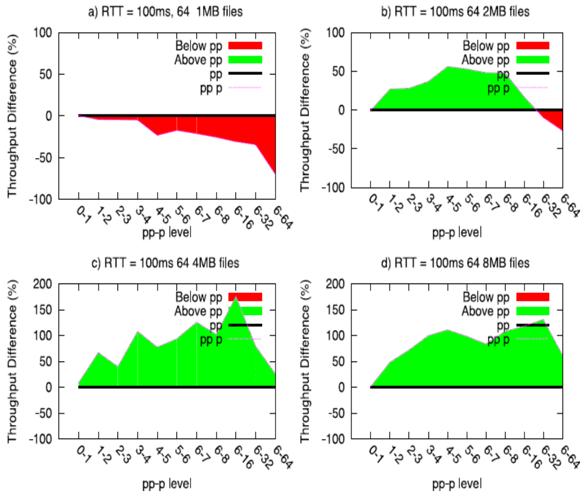


Figure 22. Influence on throughput by size of file and parallelism (From Reference [12])

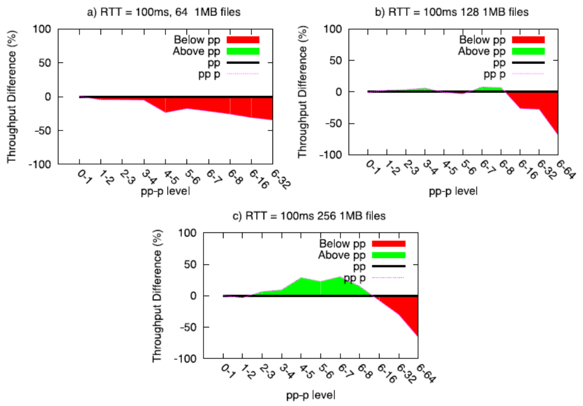


Figure 23. Influence on throughput by the number of files and parallelism. (From Reference [12])

7.3 Point of parallelism application up to which it is beneficial

We can predict the point up to which the parallelism is beneficial when we can predict the point at which the rate of loss of packet starts to increase exponentially. When it comes to characteristics of data sets there are two cases which must be taken into consideration. The first case is when a file of large size is transferred, there occurs a point at which the capacity of bandwidth of the disk is attained and the retransmission grows in numbers at that point where the level of parallelism is felt to be over limit.

In paper **[**15], level of optimization has been predicted by considering the measurement of throughput of the previous three transfers in which there is an exponential increase in the levels of parallelism. A point of knee also seems to occur in the curve of the throughput as the number of the parallel stream is increased. The other case is similar to the problem which we spoke before, the files of smaller sizes. There occurs negative effects if parallelism is applied to those small sized files, because when the division takes place, the file sizes tend to become even more smaller and becomes many streams and resulting in a failure of the size of windows of the stream to reach the maximum value as the minimum required data to be sent is missing. This problem can be rectified by using the pipelining**.** Figure 24, It can be seen that the parallel streams of multiple numbers are used on transfers which are pipelined and of files which are 64 in number and of 1MB size, the parallel streams are spotted to be at the phase of slow start for a long time. The overhead is directly proportional to the parallelism and hence the highest size of the window is not possible to be reached Fig 25. It is unfortunate that it is not possible to tell the breaking point at which the parallel streams which are individual are ought to use their time at the slow start phase.

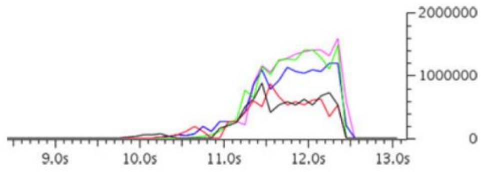


Figure 24. Effect of parallelism on datasets of small size. (From Reference [12])

7.4 Effects of concurrency without pipelining or parallelism

It is possible to use concurrency without applying pipelining or parallelism, it can be done so be using it in situations where there is complete utilization of the available bandwidth added with transfers which are concurrent and less in numbers and the file numbers are large, concurrency can be implemented with the caching of data channel and the performances derived are similar to that of the case where the parallelism, pipelining and concurrency are used together. But the drawback is that the level of concurrency optimization will be much higher while using it individually without pipelining or parallelism. It means that there are many processes and the performance will be not up to the expectation. Thus, it is always advisable to use the concurrency combined with parallelism and pipelining.

7.4.1 Advantages of concurrency

Concurrency can be used at places where parallelism proves to be deteriorating the performance of the system with pipelining. Figure 25 shows that combining concurrency with pipelining performs better than the combined use of the concurrency, parallelism, and pipelining. This occurs due to the fact that there is a negative effect on pipelining caused by parallelism. Whereas when the size of files increases then the combination of all three, pipelining, parallelism and concurrency prove to be equally productive as the concurrency and pipelining combination. In the case of smaller RTT networks, the effect of pipelining is little effect and thus a faster rise to the maximum throughput can be achieved and the performance seems to be better with the combination of parallelism, concurrency, and pipelining as there is no negative influence on the pipelining caused by parallelism. When there is increase in the size of the files and their number then additional parallelism causes a fail in its effect since the same came be provided by concurrency.

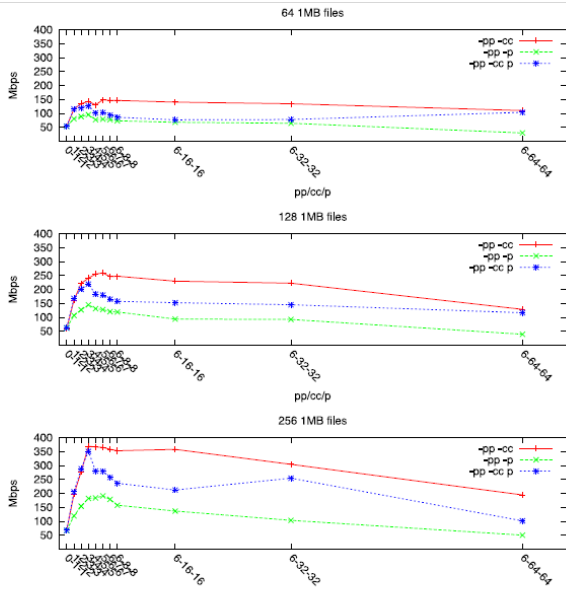


Figure 25. comparing parallelism and concurrency (From Reference [12])

7.4.2 Effect of capacity of network on optimal parallelism and levels of concurrency

In order to take a measure of the capacity of network's effect we have to eradicate the bottlenecks which were seen earlier, that are associated at system end and characteristics of dataset. This can be experimented by a collection made up of instances with AWS at four categories of performance bases networks which are different. The categories are low, medium, High at 10 Gbps. With 32 files of size 128 MB which are transferred with a RTT which is artificial at 100 ms among the instances. The files are chosen as such that they are large in size, to avoid the effect of datasets and their characteristics. Figure 26 depicts the capacity of network's effect on various methods like the parallelism, concurrency and caching of data channel. It is best to observe the benefits of performance of concurrency and parallelism through transfer of data occurring in wide area. The throughput displays an increase which is linear at first due to the increase of number of parallel streams which is linear and the concurrency. At a state when it keeps becoming higher it comes close to the capacity of the network and then there is exponential increase followed by a decrease or attainment of the steady state transfer form. Thus it can be obviously noted from these instances that ‘the optimal parallelism and concurrency levels increase as the network capacity increases’.

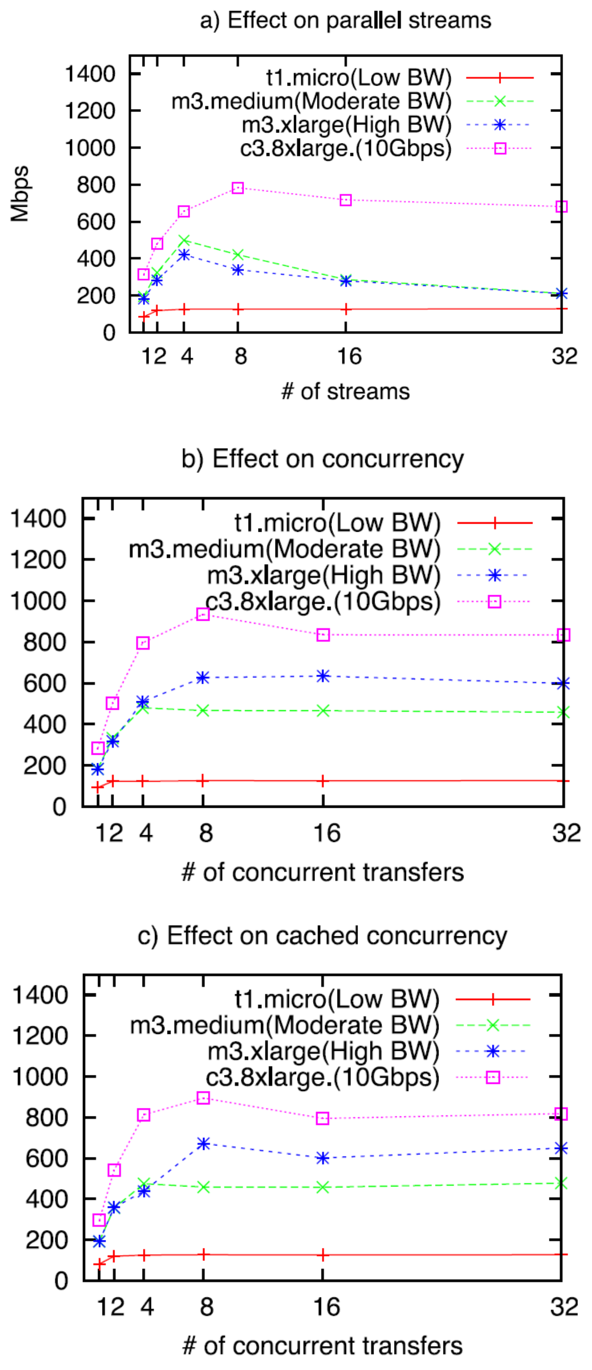


Figure 26. influence of capacity of network on concurrency and parallelism. (From Reference [12])

7.5 Effect of usage of UDT rather than TCP

There are previous studies about UDT and its comparison with TCP, this topic is spoken about in works like ([28] and [29]). But there exists a common ground among these studies, it is that they are carried out at 1 Gbps In networks of this speed. Which uses transfers of wide area. So, these observations show that the TCP which is parallel has better performance that the UDT but in reality UDT proves to be better when compared to the TCP of single stream. And UDT also has good performance characteristics when used in RTTs which are long. But, no works discussed above has spoken about the same at 10 Gbps networks.

Now, We shall see the comparison of UDP with TCP with 10 gbps networks and with long RTT. It remains difficult to attain the maximum capacity of the network in spite of the networks ideal nature. The other types of problems faced may be, capacity of the CPU, Capacity of the NIC and bandwidth of the disk**.** Figure 27 shows the concurrency at UDT with a 10 Gbps network combined by an RTT with 2 ms. The stream which is single for a UDP performed at 3 Gbps. At the point when there is an increase in the levels of concurrency from 2 to 4, there is no considerable change in the throughput. The 3 Gbps networks are shared within themselves by the streams. In Figure 28 shows both TCP and UDT together. A overhead was observed additionally while using UDT with GridFTP. But the performance of the TCP in parallel has been observed to be better than all. Whereas in XSEDE it is UDT which has performed better than everything else. Figure 29, shows the output of transferring 1 MB files 1000 in numbers with an XSEDE network along with short as well as long RTT. And the results depict that the UDT has bad performance with short and can perform better than TCP at a stream which is single and RTT which is long.

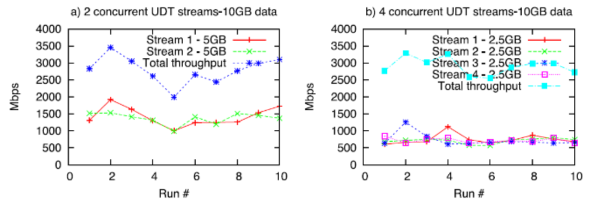


Figure 27 UDT transfer reactions due to effect of concurrency. (From Reference [12])

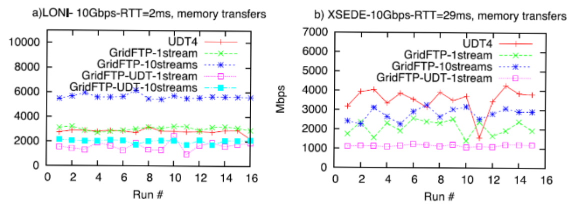


Figure 28. GridFTP vs UDT memory transfers (From Reference [12])

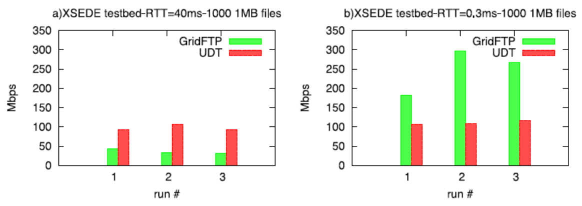


Figure 29. GridFTP vs UDT transferring small files in large numbers (From Reference [12])

Thus, we can come to a conclusion that for RTT that are long it is better to use UDT but its performance is not commendable in LONI. TCP can perform equally at both circumstances but care has to be taken to set the value of parallelism.

7.6 Rules involved in optimization

There are a set of rules which must be followed to work with the algorithm involved in transferring datasets which are large in size combined with GridFTP concurrency, parallelism and pipelining.

It is advisable to use pipelining always irrespective of its effect with the throughput because it has other advantages which are very important like that it makes use of a single channel of data thus enabling many files to be sent in a continuous manner. Which would result in the total amount of bytes that are sent and received by a single RTT. Other than this it enables the overlapping of overheads of processing with the messages of control channel along with the transfers of data channel which ultimately removes the time which is idle between the transfers.

The dataset must be divided into two parts and we have to set various pipelining levels for the part which has lesser file size than the BDP. The amount of data sent or received through a single RTT could not exceed the size of the file which is average and has to be multiplied along with the levels of pipelining. The effect of pipelining will dominate in the region where the size of the file is less than that of the BDP.

It is always important to keep the data size in a chunk big because this plays a major role in pipelining levels when they go through transfers.

It is better to use concurrency and pipelining together for files of small size and less in numbers. If we use parallelism the throughput will be affected drastically.

When the size of the file is big use parallelism with pipelining and concurrency as there will not be any negative effects due to parallelism on pipelining.

When there is a shortage in the file numbers to use concurrency, parallelism can be used.

In transfers involving wide areas, UDT can be employed with a preference that a stream which is single is used. If there is an option of transfer of stream which is parallel, then the TCP combined with the number of streams which is optimal is equally good as the UDT and at cases it is better than it.

7.7 ALGORITHM

This part deals with the execution of all the theoretical parts which we have discussed earlier. These algorithms which we are going to see, plays a major role in determining the correct values for optimizing the concurrency, pipelining and parallelism.

There are two algorithms which can be used, and they both use two different approaches, they are adaptive and aggressive. In the adaptive one there is a gradual movement to attain the peak bandwidth of the network, whereas in the aggressive one there is a quick approach in doing the same.

7.7.1 Adaptive approach Algorithm

As we saw earlier, the BDP plays a major role in this algorithm. The first step involved in this algorithm is that the files are divided into two parts based on their sizes, where the first part contains the files whose sizes are below the BDP and the other part consists of the files whose size is above the BDP. The logic is that if we identify the files with sizes less than the BDP then it is easier to assign the levels of pipelining as their effect is commendable on files sizes which are less. A recursive chunk division algorithm is used in the first part where as the second is set with a static level 2 of pipelining. (The algorithm shown below is referred from work [12])

Algorithm 1. Recursive_Chunk_Division(RCD)

```
Require: $list\_of\_files \vee start\_index \vee end\_index \vee$
    $total\_number\_of\_files \vee min\_chunk\_size \vee$
    $parent\_pp \vee max\_pp$
  Calculate $mean\_file\_size$
  Calculate $current\_opt\_pp$
  Calculate $mean\_file\_size\_index$
  if $current\_opt\_pp! = 1$ &
  $current\_opt\_pp \neq parent\_pp$ &
  $current\_opt\_pp <= max\_pp$ &
  $start\_index < end\_index$ &
  $mean\_file\_size\_index > start\_index$ &
  $mean\_file\_size\_index < end\_index$ &
  $current\_chunk\_size > 2 * min\_chunk\_size$ then
    call $RCD\_dividing\_the\_chunk\_by\_mean\_index$
    $(start\_index -> mean\_index)$
    call $RCD\_dividing\_the\_chunk\_by\_mean\_index$
    $(mean\_index + 1 -> start\_index)$
  else
    $opt\_pp = parent\_pp$
  end if
```

7.7.2 Recursive Chunk Division

Different files clusters are constructed with this algorithm which is based on mean, and the clusters deal with pipeline values which are different. The value of the pipelining is calculated based on the divided BDP on to mean size of the file, then the dataset is divided recursively with the mean size of the file with many conditions being satisfied. Division can be applied only to a file chunk whose optimal value of pipelining is not higher than that of the maximum level of pipelining of its parent. Algorithm 1 presents this idea.

Figure 30 depicts an example where the dataset is clustered as chunks based on levels of pipelining. Here there are randomly generated datasets with values as such 1.5 and 7.5 MB. The X axis deals with the size of the files in KB and the Y axis shows the level of optimal pipelining.

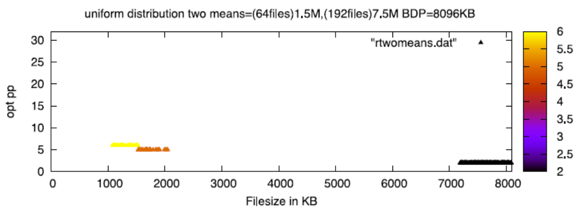


Figure 30. performance of the recursive chunk size division algorithm (From Reference [12])

7.7.3 Levels of Concurrency and adaptive parallelism

This algorithm uses different techniques for the two parts mentioned above, it uses concurrency and pipelining for the files which are less in size than BDP and uses the parallelism along with this for the part whose size is greater than the BDP. Thus, it follows the rule 3 and rule 4 mentioned before. There are various details given as the input, they are list of the file, file numbers, least size of the chunk, BDP and least number of chunks. For the first part, a phase revision is applied on the chunks that are divided, when the chunks of small size are combined and the chunks of large size can be divided again to apply concurrency which is adaptive. The first level of concurrency is started with and then the chunks are made to transfer in accordance with levels which increase, this is carried out till the drop down of the throughput and gradually the level of optimization is found out. The left-over chunks of data undergo transfer in accordance with the level of optimization. In mean time, the chunks which have the same levels of pipelining are combined and transfers are made with those concurrency and pipelining values.

In [22], the advantages of exponential increase of the parallelism level is discussed. The curve of throughput behaves in a logarithmic way along with the increase in levels of parallelism along the streams of parallelism. As there is small number the changes in throughput are a lot significant, and when there is large number then its significance is less. The increase which is exponential helps in faster ascend to the highest throughput. But the increase which is linear causes more data divisions which further results in overhead increase. The aim of this algorithm is to swiftly find the breaking point at which there is drop down in the throughput. The highest level of throughput can be reached greedily by dropping down the throughput by 5 percent and stopping the increase in level of parallelism which will eliminate the fluctuation within the network and also the load of the end system.

For the part which is higher than the BDP, the static level of parallelism is 2 and the level of optimization of parallelism is derived first then the level of concurrency which is optimal is derived. There are few conditions which must be considered to form the chunk with the files, they are the size of the chunk should be greater than the lease size of the chunk with multiplication with levels of concurrency and then the other condition is that the file number in a chunk should be higher than the level of concurrency. The level of parallelism is exponentially increased along the transfer of chunks from single level of parallelism, and this is done until there is a drop in throughput. Once the level of parallelism is set the same method can be adopted and the level of concurrency can be set for the transfers. By finding out the levels of parallelism, concurrency and pipelining the same can be added to a single chunk and the optimized values can be set.

7.7.4 Multi-Chunk Algorithm

This algorithm is discussed briefly in the work [30], It works to improve the throughput transfer with datasets of different file sizes. The algorithm works by dividing the files based on their sized into ‘small, medium, large and huge. And it uses them to find the optimal value of all three concepts for each of the divided parts. Followed by transferring the chunks along with their optimized parameters.

The multi chunk algorithm is focused on reducing the influence of throughput transfers which are low corresponding to the files that are small in size and with datasets that are mixed. The throughput is dependent on the ration of quantity of small files over the total amount of dataset, thus it can be less than that of the overall value. This algorithm works by transferring large and small files simultaneous rather than doing them separately, hence the period of time is reduced.

In contrast to the previous algorithm the Multi chunk algorithm evaluates the values of the parallelism and pipelining from the start point associated with the transfer. In-order to find the value of concurrency related to chunks, all the chunks are equally treated and the balance channels available are distributed in even to the chunks. The round robin method is used to here to carry out the distribution. The chunks can be ordered to improve the results and it offers good distribution of chunks when the channels available is less than that out the chunks. The algorithm works to determine the way in which the channels will be allocated to the chunks which are in need. When the distribution is completed, the algorithm schedules the file chunks with the level of concurrency calculated by each of them respectively. After completing all the transfers the channels are scheduled in accordance to the chunks with respect to their time of completion. (The algorithm mentioned below is referred form reference [12]).

Algorithm 2. Optimal Parallelism-Concurrency-Pipelining (PCP)

```
Require: $list\_of\_files \vee total\_number\_of\_files \vee min\_chunk\_size \vee$
    $BDP \vee min\_no\_of\_chunks$
  Sort the files
  Divide the files into $Set1$ and $Set2$
  FOR SET1
  Create chunks by applying RCD algorithm
  while Number of chunks is less than min_chunk_no do
    Divide largest chunk
  end while
  $curr\_cc \leftarrow 1$
  $prev\_thr \leftarrow 0$
  perform transfer for the first chunk
  while current throughtput $> 0.95\times$ previous throughput
  do
    perform transfer for the consequent chunk
    $curr\_cc = curr\_cc * 2$
  end while
  opt cc = prev cc
  Combine chunks with same pp values
  Perform transfer for the rest of the chunks with optimal
  pp and cc
  FOR SET2
  Set optimal pp as 2
  $curr\_p \leftarrow 1$
  $prev\_thr \leftarrow 0$
  Create a chunk with the minimum chunk size
  perform transfer for the consequent chunk
  while Current throughtput $> 0.95\times$ Previous throughput
  do
    perform transfer for the consequent chunk
    $curr\_p = curr\_p * 2$
  end while
  opt p = prev p
  Repeat the same steps for finding Optimal CC
  Transfer the rest of the data with Optimal PP, P and
  CC values
```

The difference between the two algorithms discussed can be written as,

- The Multiple chunk method assigns a level of concurrency which is static to all the dataset as a whole and its corresponding transfers and the streams which are parallel also has a static value. They remain the same throughout the transfers. Whereas the first algorithm which we saw begins with a least value for concurrency and parallel stream and the values increase till they attain the optimum value.
- The multiple chunk method as the name says can transfer many chunks at the

given time whereas the PCP method follows one after another transfer.

- The multiple chunk method adopts the aggressive method and thus has a value of parallelism and concurrency which is aggressive, whereas the PCP uses adaptive method and does the same slowly.

7.8 Experimentation and Results

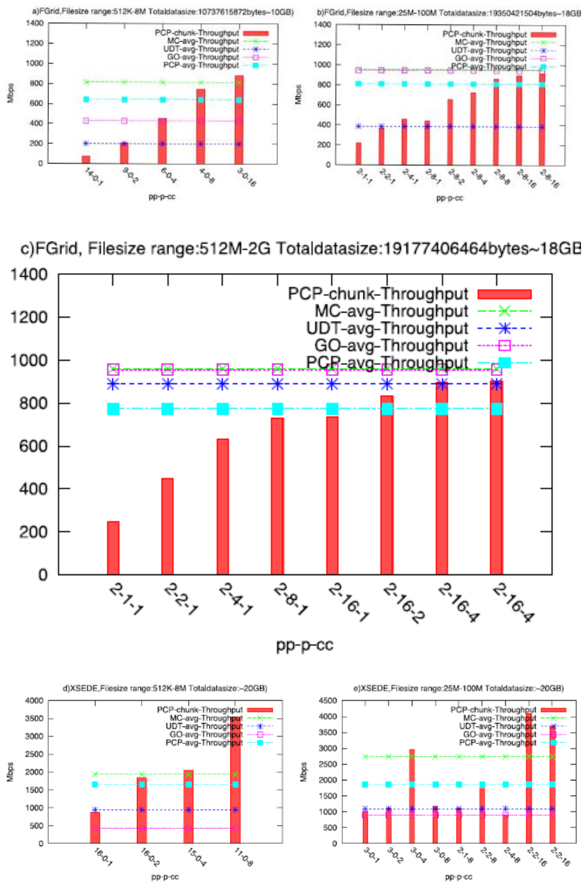
The experimental verification of the algorithms discussed starts here. The test background Is chosen to be a Xsede and FutureGrid testbed with highspeed network and cloud testing can be done by cloud networks hosted by Amazon Web services Ec2 instance. For the testing purpose datasets of different sizes say small, medium and large are used. The dataset weighs about 10 to 20 GB in total. FutureGrid is a testbed with cloud computing which is wide area and has a speed of !0 Gbps. But the support is offered only for 1 Gbps interface networks and the throughput is dependent upon the interface of the network. Xsede has the similar characteristics but with performance of CPU and disk dependencies and nodes of data transfer dependencies.

7.9 Results

Figure 31 depicts the results of the experiment with FutureGrid. It can be seen that for the datasets which are small in size the performance of the UDT is poor. From this we can understand that the tools are incapable of handling datasets with files that are small in size and large in numbers. The Multiple chunks and PCP algorithms perform based on their principles by dividing the datasets and increasing and keeping constant the levels of the three factors respectively.

For the datasets, whose size is medium Globus Online satisfies the throughput generated by the multiple chunks algorithm with a static setting of levels of pipelining, parallelism, and concurrency values. The performance of UDT is poor and the throughput is average for the PCP algorithm. The final parts made by the PCP algorithm weighs about 10 GB and then transferred with the speed that is same as that of the multi chunk algorithm and Globus Online. Then as per the principles there is gradual increase in the value of parallelism followed by the concurrency by the PCP algorithm till the point beyond which the throughput cannot be increased.

For the datasets whose size is large, the combination with multiple chunk algorithm gives a throughput which is average and it begins to saturate the bandwidth of the network, whereas the performance of the UDT is better. The output chunks of PCP algorithm were able to attain the maximum bandwidth limit of the network. The largest throughput possible is within the interface of the network instead of the throughput of the disk. From these results it is proved that the algorithms that were discussed has the capacity to reach the largest limit without the influence of the characteristics of the dataset. While the UDT and Globus Online are best used with files that are large.



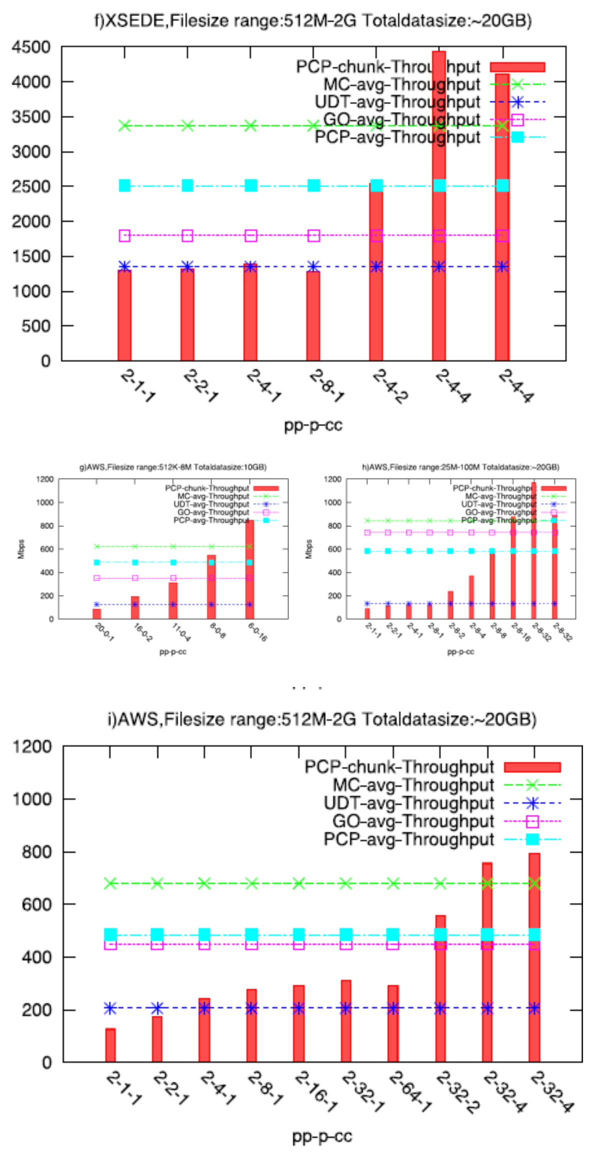


Figure 31. Results of effect of algorithm on FutureGrid (From Reference [12])

Figure 31e, 31f, 31g, depicts the experimental verification of the algorithms with the Xsede network. Datasets of the same characteristics have been used here. For the dataset with the size which is small the multiple chunk and the PCP algorithms are the ones which perform best. The UDT and Globus Online shows the performance which is worst. The final chunk generated by the PCP, attains the throughput of 3500 Mbps. The dataset whose size is in the middle is split into two parts and the first part deal with the increase in level of concurrency while the later deal with that of the parallelism. Even here the multiple chunks algorithm outperforms the PCP and others while the PCP identifies the best level of concurrency. The UDT and Globus Online shows the performance which is worst. The final chunk generated by the PCP, attains the throughput of 4000 Mbps. For the datasets of large size

8. Introduction and review on Pipelining in GridFTP

This chapter deals with the review of work [13]. The above paper speaks about many issues, now we will see in details about one concept which was used in the previous paper. The topic of our interest now is the role of pipelining in data transfer and the "lot of small files" problem. In the work [12], the pipelining concept has been discussed in depth and the first real time application for rectifying the " lot of small files" problem has been stated. We will go across the " lot of small files " problem, the methods that were available to rectify it and the effect of pipelining in solving this problem with the experimental versification and results.

GridFTP acts as a protocol to transfer data in huge volumes and it works in high speed. But the major difficulty faced by the protocol is the transfer of files in small size and large numbers. The throughput of the transfer is lowered due to the existence of latencies in the transfer path. One more problem is that, at times the transfer path has to remain idle due to lack of data to be transferred, at this situation the window of the TCP remains shut and to start again it should initialize through the slow start algorithm, which takes more time. This problem involves penalties in performance which are severe. This phenomenon is called as the "lot of small files problem". Pipelining has been suggested as a solution to address this issue in this work. As we saw earlier the pipelining has the capacity to execute multiple transfers at the same time.

There were other solutions available to address this problem earlier, which include tarring the multiple files into a single file before transferring and transferring it as a single file, this requires excess space In the disk and time in CPU. Other similar solution is tarring the files while on the fly, this requires extra overhead in the network and time of processing. These solutions become complicated as not all files can belong to a single directory. Hence these cannot be the perfect solutions which can be used readily to address the problem.

8.1 The problem of "lots of small files"

The GridFTP has two channels as semantics of protocol. One deals with the messages of control, as such the request for transfer of files and the next one to stream the payload of the data. These protocols play a major role on the "lot of small files" problem.

Transferring a file requires establishing a channel for data. The procedure requires sending commands through the channel for commands with a description of the required channel of data. The descriptions include the protocol that would be used, ASCII or binary format of the data, connection used to be active or passive, and attributes of protocol that are specific. After executing and completing these commands the transfer for a file request can be successfully made. Then the requested channel of data is formed and the requested file can be sent through the channel.

We have been seeing about a concept called the "data channel caching", it the advantage which GridFTP has over the standard FTP. It is the enhancement, by which multiple files can be transferred by the same channel of data.

8.2 Transfer of files

In order to initialize the transfer the send and receive commands must be made and then the client responds to this command and then the data can be transferred between client and the server. "226 transfer complete" is the message which acts as an acknowledgement to indicate the completion of the transfer. The next transfer can only be initiated upon receiving this command. This command marks the completion of one trip through the network and gives the signal that the transfer path Is idle and can be used for the next transfer. This wait delays the completion of transfer thereby affecting the throughput. There is one more problem related to the window size of the TCP which affects the throughput.

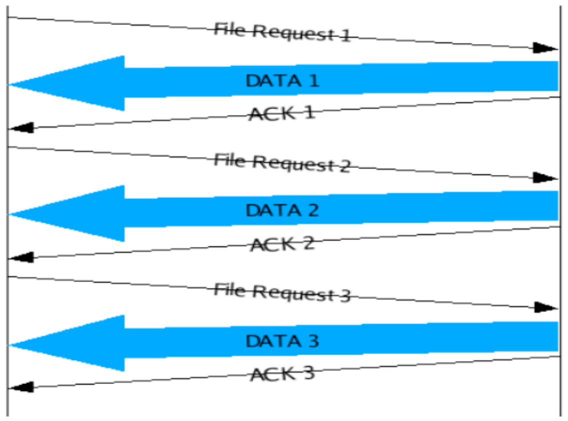


Figure 32 Transfers without pipelining. (From Reference [13])

8.3 Pipelining

Pipelining works towards reducing the time involved in transfers for rectifying the "lots of small files" problem. Pipelining enable the execution of multiple transfers at a time. It doesn't wait for the message "226 transfer successful". The command required for transfer can be sent without any time constraints. The acknowledgements follow the order in which they are sent**.** Figure 33 depicts the given process.

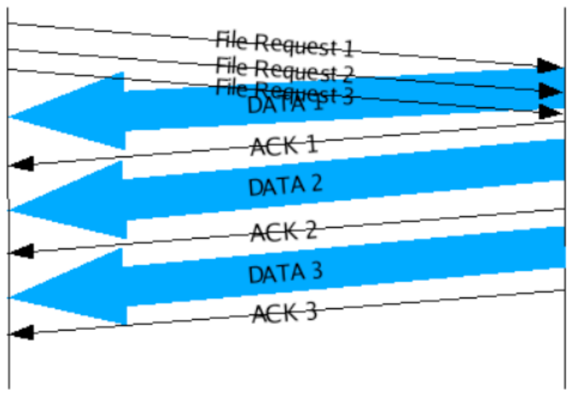


Figure 33 Transfers with pipelining. (From Reference [13])

The pipelining uses the overlapping technique with the request of transfers and the actual transfers of data. Thus, multiple transfer requests can be made at the same time. Empty queues are immediately allocated with next transfer instructions. But, the buffer corresponding to the server will have a closed window of TCP, upon the buffer getting full.

8.4 Implementation and Result

The concept of pipelining is implemented in a Globus Toolkit GridFTP and the results are recorded. The results are as expected and the transfer rates are quick and the throughput is as expected. Thus the advancements derived due to the changes have been proved.

9. Application of pipelining in distributed systems for jobs that involve intensive data execution

This chapter is the review of work [14]. Which deals with the usage of pipelining concepts which we have been seeing, to improve the results of data intensive jobs in distributed systems. The importance of data transfer technologies to manage the increasing size of data is highlighted. The challenges faced in analysis from a remote location and disseminating huge data sets in real time are also discussed. The concept of executing jobs in parallel is quite common and "heterogeneous earliest finish algorithm" is one such technique but had few disadvantages. There were few suggestions to rectify the disadvantages with clustering of tasks, but this didn't sound good when it comes to movement of data becoming a factor which is trivial.

This work concentrates more on proposing models of pipelining with execution in parallel to carryout transfer of data. It also incorporates the bandwidth of network and the resources which are reserved at the site of execution. These models are analyzed formally and the ones which turned out to be the best are projected with the optimal level of parallelism. These models are implemented in GridFTP. The realtime execution of this model in actual distributes systems shows an improvement of 50 % in performance.

9.1 parallel computing and distributed parallel computing

There is good amount of research on the topic parallel computing within a large and single machine. The computer architecture associated with these machines can be classified as Temporal, data, functional and instruction level parallelism. The instruction level parallelism is the one which we saw in the first paper and we have discussed more in detail about this process. These techniques can be used in distributed systems with other constraints as such that communication in wide area networks within the distributed systems.

Distributed parallel computing were developed to manage the scientific workflows profitably. Condor DAGMAN, Pegasus are famous workflows. There are many limitations constrained to the static approach such as ignoring the properties of the job. Clustering the task dynamically has been referred as a solution to remove the limitation.

The recent algorithms that deal with workflows has interests on movement of data efficiently and allocation of resources in computers. There are views that state that the scheduling of a job in parallel distributed systems has not shown any considerations on data transfer optimization. A few models which enable parallel execution are analyzed and the overlapping in data transfer is considered.

9.2 Scheduling of jobs in parallel that are massive and distributed

The challenges involved in this section are briefly discussed and the models of parallel execution are analyzed formally. The authors have formalized the problem involving scheduling in accordance with the HPC single site, within which the allocations provided to the user are in limited numbers so as the jobs of parallel which are in multiple numbers. Again, as we saw earlier, the optimization of parallelism is depicted as the solution for this problem by the authors. The number of jobs to be executed in parallel is estimated with respect to the available resources of networks.

9.2.1 Analyzing the time of execution theoretically

In this part the authors have evaluated the execution time theoretically from various derived equations. The equations which were used to determine the performance are used here. The methods like the simple pipeline execution, superscalar style execution, and coscheduling with networks are considered and evaluated analytically

9.3 Experimental evaluations

Figure shows the experimental setup that is required for a verification. the discussed theoretical evaluations and topics have been experimented and the result are recorded. Thus, the proposed parallel methods have been proved to be efficient.

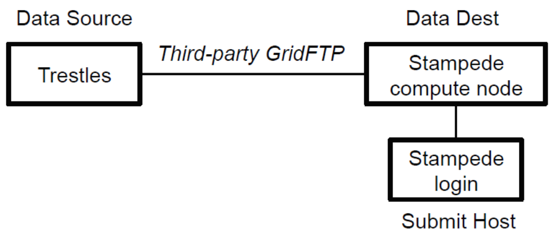


Figure 34 Experimental setup. (From Reference [15])

10. Level of optimization prediction of parallelism in data transfers corresponding to wide area

This chapter deals with the review of work [15]. In the work [22], more details about the necessity of predicting the optimal level required for parallelism is stressed. The drawback of increased number of streams in a network due to increased level of parallelism is spoken about. The prediction of the level of optimization is used as a solution for this problem. The authors speak about many models which can be used for this prediction and suggest the best out of it. The models suggested does not rely on the historic information and the prediction overhead is also low. The authors have also provided an algorithm which can be used to predict the levels that would be required. Then at last the feasibility of this model is measured with comparisons to a GridFTP data transfer model.

11.1 The model which exist and their advantages and disadvantages

The “Hacker et al model” is the one whose throughput is dependent on the factors like the time for the round trip, rate of the loss of packets, and the size of the segment which is maximum. The size of the segment which is maximum corresponds to the “Maximum Transmission Unit (MTU)”. The time for a round trip is the measure of the time consumed each complete transaction of data between the sender and receiver. The ratio of packet loss deals with the ratio of packets which are missing to the number of packets in total. Thus the formula for this model shows the relation between these factors and the throughput.

The disadvantage of this model is that it does not worry about the increase in the value of packet loss. There is a point at which when the number of streams increase the value of packet loss also increases. This makes it necessary to point to the point of the knee corresponding to the rate of loss of packet.

The “Dinda and Colleagues Model”, the precious model deals only with the networks which are uncongested and fails to provide us with the relation between the rate of loss of the packet and the number of streams. This model rectifies the drawback by considering them and providing the relationship between the two.

The “Altman et al. model” depends on the theory that when too many connections are opened it causes the overhead in processing and makes only small amount of the bandwidth to be available for few other flows. A formula is used to use this model, and two issues must be addressed first to use this. The first is to know the capacity of the bottleneck link and the other is that the formula used gives back the streams in total numbers that are required to survive to get the throughput aggregate. This causes a problem if we need additional streams for improving the functionality.

The ”kola and Vernon Model” is regarded as implementing a protocol, which will be used with regard to utilizing the links of high bottlenecks and the backlog average is low. This goal can be achieved with computing a targeted backlog among the bottlenecks that are measured currently. This model is executed in multiple steps. The disadvantage of this model is that the calculations require information has to be extracted from a layer which is at low level.

11.2 The proposed models

The authors propose a model which has similarities to the “Hacker et al. Model” and the “Dinda et al.models”. The basic concept is extract from these two models and their limitations are rectified and a new model is developed. The rate of packet loss data is modelled as such that to rectify the disadvantage of the model. Here in the new proposed model the congestion point’s location is enabled due to the occurrence of multiple streams. The next is to increase the fitting of curve with more amount of data. The characteristic of the curve of the throughput which shows a sharp increase and then a stable point till reaching the link’s capacity. This model is customized to predict the throughput correctly and the behavior in order to achieve the required output. Then comes the curve of throughput modelling in a logarithmic manner. Here the relationships which were missing in the previous algorithms are rectified and the relationships are defined. The last change is to predict the model equation of the order through the Newton’s iteration. There are transition problems that were related to the previous models. This step rectifies the transition problem which occurs when there is increase in the number of streams associated.

11.3 The final algorithm

Then using this model an algorithm is developed to find the actual level of optimization. The authors introduce the algorithm to find the actual combination of pairs. The algorithm is customizable and can be adjusted easily to the models that are used. (The algorithm written below is referred from [15]).

```
Input: $CollectiveData[m][2] = \begin{pmatrix} n_0 & Th_{n_0} \\ n_1 & Th_{n_1} \\ \cdots & \cdots \\ n_l & Th_{n_l} \\ \cdots & \cdots \\ n_{m-1} & Th_{n_{m-1}} \end{pmatrix}$
Output: $(n_r, Th_{n_r})(n_s, Th_{n_s})(n_t, Th_{n_t})(a', b', c')err_{min}$
Begin
  for $i \leftarrow 0$ to $m-2$ do
    for $j \leftarrow i+1$ to $m-1$ do
      for $k \leftarrow j+1$ to $m$ do
        $a, b, c \leftarrow$ GETCOEFFICIENT($CollectiveData[i]$,
          $CollectiveData[j], CollectiveData[k]$)
        $err \leftarrow$ ERREVALUATION($a, b, c, CollectiveData$)
        if $minerr$ is not initialized then
          $minerr \leftarrow err$
          $r, s, t \leftarrow i, j, k$
          $a', b', c' \leftarrow a, b, c$
        else if $err < minerr$ then
          $minerr \leftarrow err$
          $r, s, t \leftarrow i, j, k$
          $a', b', c' \leftarrow a, b, c$
        end if
        $k \leftarrow k+1$
      end for
      $j \leftarrow j+1$
    end for
    $i \leftarrow i+1$
  end for
  $n_r \leftarrow CollectiveData[r][0]$
  $n_s \leftarrow CollectiveData[s][0]$
  $n_t \leftarrow CollectiveData[t][0]$
  $Th_{n_r} \leftarrow CollectiveData[r][1]$
  $Th_{n_s} \leftarrow CollectiveData[s][1]$
  $Th_{n_t} \leftarrow CollectiveData[t][1]$
End
```

11.4 Experimental Analysis

The proposed algorithm is analyzed experimentally and the results are verified. The results are favorable to the benefits, which we were discussing throughout. All these experiments prove that there is good amount of improvement in performance that is possible due to the addition of these factors in the regular process.

12. Tuning the parallelism levels dynamically in the data transfers in wide area.

This chapter deals with the review of work [16], deals with the same parameters that we saw in the previous paper. But the approach is to dynamically alter the levels of parallelism in the wide area data transfers.

A brief introduction about the previous models which we saw in the previous work is used in this paper. An additional model named "Crowcroft & Oechslin Model" has also been discussed. This model is almost similar to that of the "Hacker et al Model". But the additional feature is the MulTCP which is implemented in this model. This is similar to the behavior of that of the TCP connection. It also gives a method to maintain the fairness to limit the size of the buffer in the connections. A formula is also used to apply this model.

This paper deals with almost the same ideas and concepts as that of the previous paper which we have discussed. A few additional changes like that of the instant measurement collection is highlighted in this paper.

12.1 Extending the Multi-parameter optimization

The previous papers and chapters dealt with finding the value of optimization required for parallelism. There are many other factors other than parallelism which can be optimized to get a better output. But it becomes a complex task to optimize everything at the same time.

In applications where the throughput should be improved for many users carrying out transfers, it becomes difficult to optimize one of the many factors and this causes a negative effect on other transfers initiated by other users.

The requirement to solve this problem is a component which can keep track of all the transfers that are involved. In this work the authors use a data scheduler named "Stork" to satisfy this requirement. The function of this component is to collect all the information required for the transfer from many sources along with their links at the same time. This will be used to carry out optimization of parameters for the multiple numbered transfers. Without this component, it is impossible to obtain the maximum efficiency with the system that is involved.

Thus, the data schedulers have a major role in these transfers. The requirement of the system is that it should have a control over tuning, scheduling and executing the transfer of data and the operations involved. When two or more transfers tend to share the same resources like the links and online sources a virtual link is created by the system and the usage track record is maintained. The virtual communication system assists the scheduler to use the prediction of networks as it has enough global knowledge about the system. Figure 35 provides the structure of the scheduler that has been used in the system. It can be seen to use the virtual channels in predicting the parameters of the network with concentrations for individual transfers at a place where there are multiple transfers taking place at the same time.

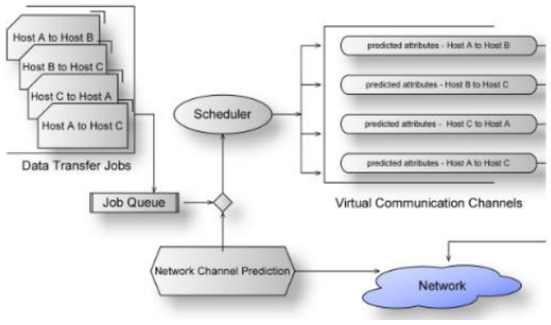

Host A to Host B
Host B to Host C
Host C to Host A
Host A to Host C
Data Transfer Jobs
Scheduler
predicted attributes - Host A to Host B
predicted attributes - Host B to Host C
predicted attributes - Host C to Host A
predicted attributes - Host A to Host C
Virtual Communication Channels
Job Queue
Network Channel Prediction
Network

Figure 35. Overall Structure of the scheduler. (Referred from work [5]).

Thus, this paper deals with a concept which is important to improve the overall efficiency of the system. But this topic has not been discussed in detail in previous works.

13. Conclusion

Thus, this review talks about two issues related to technologies that has been impacted by big-data. There will further increase in the amount of data generated and which will further lead to development in the above technologies. The future of NoSQL databases is very strong.

Reference

[1] Donald J. Berndt, Ricardo Lasa, *Sitewit Corporation: SQL or NoSQL that is the Question*.

[2] Cornerlia Gyorodi, Robert Gyorodi, George Pecherle, Andrara Olah,, *A comparative study:MongoDB vs. MySQL*. Engineering of Modern Electric Systems (EMES), 2015 13th International Conference on Engineering of Modern Electric Systems (EMES), June 2015.

[3] Seyyed Hamid aboutoraabi, Mehdi Rezapour, Milad Moradi, Nasser Ghadiri, *performance evaluation of SQL and MongoDB databases for big e-commerce data*. International Symposium on Computer Science and software Engineering (CSSE), 2015.

[4] Min-Gyue, Jung, Seon-A Youn, Jayon Bae, Young-Lak Choi, *A study on data input and output performance comparison of MongoDB and PostgreSQL in the Bigdata Environment*. 8th International conference on Database Theory and Application (DTA). 2015.

[5] Tianyu Jia,Xiaomeng Zhao, Zheng Wang, Dahan Gong, and Guiguang Ding, *Model Transformation and Data Migration from Relational Database to MongoDB*. IEEE International Congress on Big Data, 2016.

[6] Ramon Lawrence, *Integration and Virtualization of Relational SQL and NoSQL Systems including MySQL and MongoDB*. International conference on Computational Science and Computational Intelligence. 2014.

[7] Type of NOSQL Databases and its Comparison with Relational Databases. International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868 Foundation of Computer Science FCS, New York, USA Volume 5– No.4, March 2013 – www.ijais.org

[8] A Workload-Driven Approach to Dynamic Data Balancing in MongoDB. 2015 IEEE International Conference on Smart City/SocialCom/SustainCom together with DataCom 2015 and SC2 2015

[9] Analysis of Data Storage Mechanism in NoSQL Database in MongoDB. 2015 International Conference on Consumer Electronics-Taiwan (ICCE-TW)

[10] Application of NoSQL Database MongoDB Application of NoSQL Database MongoDB

[11] A Study of Normalization and Embedding in MongoDB

[12] Esma Yildirium, Engin Arslan, Jangyoung Kim, and Tevfik Kosar, *Application-Level Optimization of Big Data Transfers through Pipelining, Parallelism, and Concurrency*. IEEE Transactions on Cloud Computing, Vol. 4, Issue: 1, Jan-March 1 2016.

[13] J. Bresnahan, M. Link, R. Kettimuthu, D. Fraser, and I. Foster, *Gridftp pipelining*, presented at the TeraGrid Conf, Madison, WI, USA, 2007.

[14] Eun-Sung Jung, Ketan Maheswari, Rajkumar Kettimuthu, *Pipelining/Overlapping Data Transfer for Distributed Data-Intensive Job ExecutionI*, presented at the 42nd International Conference on Parallel Processing, Lyon, France, 2013

[15] E. Yildirim, D. Yin, and T. Kosar, "Prediction of optimal parallelism level in wide area data transfers," IEEE Trans. Parallel Distributed Systems, vol. 22, no. 12, 2011.

[16] E.Yildrium, Mehmet Balman, Tevfik Kosar, *Dynamically Tuning Level of Parallelism in Wide Area Data Transfers*, International conference on Complex, Intelligent and Software Intensive Systems, 2009.

[17] B. Allen, J. Bresnahan, L. Childers, I. Foster, G. Kandaswamy, R. Kettimuthu, J. Kordas, M. Link, S. Martin, K. Pickett, and S.Tuecke, *Software as a service for data scientists*, CommunACM, vol. 55, no.2,pp.81-88,2012.

[18] R. S. Prasad, M. Jain and and C. Davrolis, *Socket buffer auto-sizing for high-performance data transfers* J. Grid Comput., vol. 1, no. 4,pp. 361–376, Aug. 2004.

[19] G. Hasegawa, T. Terai, T. Okamoto, and M. Murata, *Scalable socket buffer tuning for high-performance web servers*, in Proc. Int. Conf. Netw. Protocols, 2001, pp. 281–289.

[20] Hildebrand, D., Eshel, M., Haskin, R., Kovatch, P., & Andr, P. (2008). *Deploying pNFS across the WAN: First Steps in HPC Grid Computing*. in Proceedings of the 9th LCI International Conference

[21] E. Yildirim, M. Balman, and T. Kosar, *Data-aware distributed computing*, in Data-Intensive Distributed Computing: Challenges and Solutions for Large-Scale Information Management. Hershey, PA, USA: IGI Global,2012.

[22] E. Yildirim and T. Kosar, *End-to-end data-flow parallelism for throughput optimization in high-speed networks*, J. Grid Comput.,vol. 10, no. 3, pp. 395–418, 2012.

[23] E. Arslan, B. Ross, and T. Kosar, *Dynamic protocol tuning algorithms for high performance data transfers*, in Proc. 19th Int. Conf. Parallel Process., 2013, pp. 725–736.

[24] T. J. Hacker, B. D. Noble, and B. D. Atley, *The end-to-end performance effects of parallel TCP sockets on a lossy wide area network*, in Proc. IEEE Int. Symp. Parallel Distrib. Process., 2002, pp. 434–443.

[25] D. Lu, Y. Qiao, P. A. Dinda, and F. E. Bustamante, *Modeling and taming parallel TCP on the wide area network*, in Proc. IEEE Int. Symp. Parallel Distrib. Process., Apr. 2005, p. 68b.

[26] T. Kosar, M. Balman, E. Yildirim, S. Kulasekaran, and B. Ross, *Stork data scheduler: Mitigating the data bottleneck in e-science*, Philosoph. Trans. Royal Soc. A, vol. 369, pp. 3254–3267, 2011.

[27] E. Yilidirim, J. Kim and T. Kosar, "Optimizing the sample size for a cloud-hosted data scheduling service" in proc. 2nd International workshop on Cloud Comput. Sci. Appl., 2012.

[28] E. Yildrium, j. Kim, and T. Kosar, "Optimizing the sample size for a cloud-hosted data scheduling service," in proc. 2nd International workshop cloud computing.Sci. Appl,2012