CHAPTER 5
CONSISTENCY MAINTENANCE ON THE MOBILE GRID ENVIRONMENT

5.1 INTRODUCTION

In a replicated environment, same data are often distributed to several sites in order to improve the data availability, fault tolerance and faster access. However, when a replica is modified by a user, the other replicas become stale. In such situation a mechanism is needed to keep them consistent, which means all the replicated data have the same content. This can be achieved using a synchronization mechanism and it provides the same view of all the replicas by propagating the modifications done on the first replica to others. This process is usually called as update propagation which is the main part of the consistency mechanism. When replicas are read-only or when no consistency guarantees are required by the users of the replicated system, a synchronization system is not needed. In the present chapter, different replica synchronization techniques are studied. Also, based on the agent based framework proposed in the previous chapter an update propagation mechanism is proposed to provide consistency in the mobile grid environment.

5.1.1 Replica Synchronization Techniques

In a replication system, there are two basic methods of ensuring consistency among all the sites which has a copy of the same data as given below,
- Pessimistic technique

- Optimistic technique

The pessimistic technique uses locking or restricting the operations on the data. Due to this, during the update of a replica, the access to all the other replicas is blocked, until the update has been correctly applied to the respective replica and propagated to the others. The process of the pessimistic approach is shown in the Figure 5.1. This approach requires permanent availability of participating sites and transmission bandwidth. If one of the participating sites holding a replica is not ready and is not confirming its acknowledgement, then the update transaction may not proceed or complete.

![Figure 5.1 Lock Based Update Propagation](image-url)
This ensures consistency as only one copy of the data can be changed at a time. However, the performance of pessimistic technique decreases when the number of replicas is high and the updates are frequent (Salman Abdul Moiz et al 2010).

On the other hand, the Optimistic approach provides more flexibility. It allows multiple replicas to be concurrently updatable based on the assumption that update conflicts are rare. In this, the access of the replica is not blocked when one of them is being updated by a site. It allows all the changes to be made locally and later on the updated data will be propagated to other replicas in a relaxed way as shown in the Figure 5.2. This approach causes stale reads for a limited amount of time.

Figure 5.2 Relaxed Update Propagation
Using optimistic replication mechanisms in the mobile environment have been studied already and the suitability to provide a scalable replication solution for the mobile users analyzed (Monterio et al 2007, Ratner D et al 2004 and Archana Sharma 2011). The authors concluded that, among these two techniques, the pessimistic approaches are not feasible when replicas are connected through unreliable links or when replicas are subject to failures. In the optimistic approach, delaying the propagation phase helps to speed up the write access even in slow and unreliable links. Optimistic algorithms work well over intermittent or incomplete network links by allowing the updates to be exchanged between any pair of nodes (Ratner D et al 2004). One of such replication model proposed by Tolia et al 2007 which uses a client server design is adopted as reference model in the present research. In this approach, a central server holds the master copy of the data object. The replicas can execute the access and update transactions locally. In frequent intervals the replica is refreshed from the master copy. Generally, the optimistic replication algorithms consist of the following elements,

- **Operation submission**
  - Operation edits local versions of the files.

- **Update propagation**
  - Each site push modified version of the data item to the primary server.

- **Conflict resolution**
  - If there are any conflicts among the operations that a site has scheduled, it must modify.

- **Commit**
  - The primary server resolves the conflict and modifications are made permanent and propagated.
Since, the present research considers replica synchronization in a mobile grid, where the mobile devices are included in the replication schema, the proposed research focuses on optimistic replication, where all replicas are kept consistent in a relaxed way as shown in the Figure 5.3.

![Figure 5.3 Update Propagation using Gateway Server](image)

**5.1.2 Update Propagation Techniques**

Another important aspect of implementing replication and consistency protocols is to decide about what to propagate as update message to the other replicas. In general, three different approaches have been followed for message construction to propagate update information as listed below,

- Send the data
- Send the update operation
- Send an invalidation message
In the first approach, the update data objects are simply sent to the other replica. In the second approach, the operation that caused an update to the data is sent to all the replicas. This operation is then performed by the remote replicas, updating their local stores. The final approach, sending invalidation involves notifying the replicas that the copy of data item that holds is no longer valid. It is then up to each replica to contact the sender of the invalidation to retrieve the new state of the data item.

From all the above approaches, sending the update operations is useful when the bandwidth is limited. Moreover, since replicas can create and store a log file for all the update operations, it is easy to propagate update log alone. It will be an efficient mechanism. However each update message is assigned with a time stamp specifying the creation time. The present research adapts this method for constructing the update message.

Besides deciding what to send in an update message, it is also important to decide who transfers an update, which determines the replica that is responsible for transferring the updates. There are two different strategies available to decide about this in a replicated environment. They are,

- **Push based strategy**
  - In push-based approach, the replica that has been updated is responsible for triggering and performing the update propagation.

- **Pull based strategy**
  - In pull-based approach, it is up to each replica to retrieve the updates and synchronize its content with the one of the latest replica.
The choices of using one or the other method depends on several factors, like the frequency of read and write operations and the availability of the secondary replicas. The pull model cannot perform well for the mobile grid environment. All the replicated mobile nodes have to scan the environment for updates periodically which may drains the battery and degrades the performance. Moreover, due to the unavailability of mobile nodes owing to link failure, the latest copy of the data object cannot always be retrieved. Besides, the push model is a useful approach for a dynamic environment. In this approach all the mobile nodes which has replica need not worry about getting an updated copy of the replicated data object. The mobile node which does the update transaction has to propagate the update to all the mobile nodes. In the present research, in order to reduce the communication overhead of the mobile node, the update message will be forwarded only to the gateway server, which will carry the update information to all the other replica nodes.

At this point, data transfer is a critical component to propagate update operations in a fast and reliable manner. The Globus Toolkit provides a protocol called GridFTP to meet these requirements. GridFTP includes the features of FTP protocol which is commonly used for data transfer over TCP. GridFTP uses two different channels such as control channel and data channel for data transfer as shown in the Figure 5.4. The control channel used to make a request to get the port number for file transfer to GridFTP server. The GridFTP client is implemented with a command line tool which can transfer the file using this protocol called as globus-url-copy. Since the proposed agent based framework is implemented with the Globus toolkit, GridFTP is used for propagating the update operations to the gateway server and from the gateway server to other replicated mobile hosts.
5.1.3 Types of Data Resources

A data resource is defined as an entity which acts as a source of data. Data resources are characteristically heterogeneous based on their semantics and implementation. Different data resources are used in the grid environment such as flat files, tables in relational databases and objects in object based database systems. In the initial stages of grid implementation only flat files are used as data resources and all are provisioned with read-only access. In order to handle the data processing challenges, current grid implementation uses relational databases such as ORACLE, DB2 and MySQL (Domenici et al 2006).

The consistency maintenance problem for unstructured data like flat file organization is harder to maintain. However, file system consistency can be achieved by replacing the replica with the modified file or based on the binary difference between the files. ADBMS has full control of all the data that it manages and provides straightforward programming interfaces to access and update data in a database. The proposed consistency approach considers a MySQL database transaction which is integrated with the Globus toolkit.
In conclusion, it is proposed that in order to maintain consistency in the mobile grid environment in a flexible way, optimistic techniques can be used. Also, sending the update operations to the middleware gateway server will be a good choice in the dynamic environment. Moreover, incorporating agents in push based strategy should be adopted to improve energy efficiency in the wireless environment.

The present chapter explains the proposed mechanism and working principle of update propagation used for maintaining consistency in the mobile grid environment.

5.2 AGENT BASED UPDATE PROPAGATION PROCEDURE

The agent based update propagation provides global consistency by propagating the update information to the remote sites. The consistency manager of the agent based framework proposed in the previous chapter is responsible for managing the propagation. The Figure 5.5 shows the architecture of the proposed consistency maintenance service over the agent based framework.

Each mobile host has N-agent, and is responsible for propagating recent updates that are performed on the Mobile host to the base station which manages the mobile hosts. The BS-agent acts as a broker for propagating recent updates between the fixed and mobile nodes in the environment. Also, it is responsible for providing a total ordering of messages that are received from the environment based on FIFO model. The consistency manager is responsible for resolving conflicts based on the timestamp assigned with each message and creating the instance of U-Agent which has the update list that is to be propagated.
The replica manager gives the details about the nodes which has the corresponding replica. From the localization manager, the reachability information about the list of replica is retrieved to propagate the update message.

5.2.1 State Transition Diagram for Update Propagation

Different states for U-agent have been defined in the proposed research. They are as discussed below

- Creating instance
  - The U-Agent creates an instance of it and stores the recent updates on this instance.
- Migration
  - The U-Agent instance migrates from the base station to other mobile nodes which contain replica that is inconsistent.

- Insertion
  - In this state, the U-agent instance inserts its stored recent updates in the data set of the mobile node.

- Removing
  - The migrated instance of U-Agnt removes itself after completion of the insertion process.

The Figure 5.6 shows the state transition diagram for update propagation:

Figure 5.6 State Transition Diagram
5.3 EXPERIMENTAL SETUP

The proposed agent based framework is implemented with small scale mobile grid environment in a test bed which is shown in the Figure 5.7.

Figure 5.7 Agent Based Framework in the Experimental Setup
Test bed includes 16 mobile nodes and all the mobile nodes run with the Red Hat Linux 7.3 operating system with each node having the same hardware configuration which includes core 2 duo processor with 500 GB storage and 3 GB RAM. The N-Agent routine is implemented with JVM (Java Virtual Machine) on each machine. These mobile hosts are connected with the grid environment by the Broadcom Dual-Band 802.11 WLAN access points. All the mobile and fixed resources are installed with the MySQL version 5.0 data base which is used as a data storage unit in the test bed environment.

To coordinate the above grid resources Globus Grid Middleware version 3 is installed in the fixed middleware gateway machine. The proposed framework is converted as a grid service and integrated with the Globus Toolkit.

In the test bed, Globus3 middleware is running on gateway server with the Grid Information Index Service (GIIS). The N-Agent is running on all the mobile nodes and contact BS-Agent available in the gateway periodically in order to update the information about the host. This information includes SNR, Mobility, battery power and read / writes counter values. Each mobile node runs a script which has read and writes operations for a data object in the cluster. In order to evaluate the performance of the system an employee sample database provided by the MySQL is used. A copy of the database is stored in the middleware gateway server and it is responsible for update propagation. The replica manager component uses the proposed dynamic algorithm and is implemented in the proposed framework. It selects the candidate mobile nodes to have replica of the database. The different tables and their relationships present in the sample database are depicted in the Figure 5.5.
5.4 IMPLEMENTATION

In this section the implementation of the proposed approach is explained. To reach an eventual consistency in which the servers converge to an identical copy an adaptation in the primary commit scheme is used. Thus, a base station server chosen as primary has the responsibility to synchronize and commit the updates, that is, to make them permanent.

Hence, the update change list will be forwarded to the base station as a message. The committed writes generate new versions of data items that must be propagated to the other sites. The sequence diagram with the
interaction between the framework components for update propagation is shown in the Figure 5.6.

Figure 5.9 Sequence Diagram for Update Propagation

Each mobile host is assigned with a set of scripts which includes update transaction on employees and salaries table of the employee database. Each script having multiples of ten transactions starting from ten update and insert commands. It is assumed that after completion of every script execution, the update propagation of update transactions is executed automatically. The time taken for execution of each script is calculated for the performance evaluation. The following section elaborates the implementation of the same.
5.4.1 Creation of the Update List

The following part of the code generates and sends the list of modification to the base station. In order to execute the command line utilities to get the update log file from MySQL data base and to propagate the log file to middleware server, `getRuntime()` and `exec()` commands are used. Also `--log-bin` and `mysqlbinlog` utility is used to create and get the content of the update log file respectively.

```java
import java.io.IOException;

public class UpdateList {
    public static void main(String args[]) {
        try {
            Runtime rt = Runtime.getRuntime();
            Process pros1 = rt.exec("--log-bin first");
            Process pros2 = rt.exec("mysqlbinlog first");
            try {
                BufferedReader reader = new BufferedReader(new InputStreamReader(pros2.getInputStream()));
                String msg = null;
                while ((msg = reader.readLine()) != null) {
                    System.out.println(msg);
                }
            }
            String BSid = getBSid();
            Process pros3 = rt.exec("globus-url-copy" + BSid);
        }
    }
}
```
BufferedReader reader = new BufferedReader(new InputStreamReader(pros3.getInputStream()));

String msg = null;

while ((msg = in.readLine()) != null) {
    System.out.println(msg);
}

catch (IOException e) {
    e.printStackTrace();
}
}

5.4.2 Update Propagation

The base station receives the update log file by listening to the port number assigned for every globus-url-copy request from any candidate node which has the replica. After receiving the file, the BS-Agent executes the update log file to add the updated records in the master copy which is locally available. Finally, it creates the U-Agent instance to propagate update information to all the other replicas. Since BS-Agent considers FIFO ordering, the conflict is resolved based on the timestamp. If any of the timestamp of the update log is less than the previous log, BS-Agent just ignores and it will not forward the update log to the other nodes. This part of the code illustrates the same. Using the ServerSocket() method, the update log file is received and the contents read using the ObjectInputStream object.
/*CODE TO RECEIVE THE MESSAGE*/

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.*;

public class recvMsg {
    public static void main(String args[]) {
        try {
            ServerSocket ss = new ServerSocket();
            Socket Sock = new Socket(getPort());
            Sock = ss.accept();
            ObjectInputStream obins = new ObjectInputStream(Sock.getInputStream());
            String Umsg = obins.readObject();
            System.out.println(Umsg);
            obins.close();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
/* U-AGENT CREATION AND UPDATE PROPAGATION */

public class uAgent extends Agent
{

protected void setup()
{

addBehaviour (new uBehaviour(this));
}

class UAbehaviour
{

public UAbehaviour (Agent a)
{

Super(a);
}

public void action()
{

msg = addBehaviour.getContent();

for (int i ; i < replicas.getCount() ; i++)

Rlist = addBehaviour.addReceiver();

send(msg);
}
}
5.5 PERFORMANCE EVALUATION

The performance of the system is evaluated based on the number of message exchanges and Update Propagation Latency (UPL).

5.5.1 Message Transfer

Using the experimental setup number of message exchanges required by the candidate mobile node to propagate update information is measured and tabulated in 5.1.

<table>
<thead>
<tr>
<th>No of replicated node</th>
<th>Normal case</th>
<th>Using agent based approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

The message transfer includes propagating the update log to the base station and to the other candidate nodes. In the conventional multi-master scheme, the candidate node has to transfer the update log to all the other candidate nodes and to the base station. It consumes more energy and time due to the limited resources such as battery and connectivity.

In order to overcome this, the proposed approach considers the forwarding of update log only to the base station so as to overcome the communication overhead such as finding the candidate nodes and the availability of the nodes faced by the conventional optimistic replication. Among the sixteen mobile nodes the replica manager finds a maximum of 4
candidate nodes using the optimization algorithm. The comparison plot is depicted in the Figure 5.7. It shows that the proposed framework is efficient and reduces the communication overhead of the mobile hosts.

![Number of message transferred](image)

**Figure 5.10** Number of Messages Needed by the Mobile Host To Propagate Update Log

### 5.5.2 Update Propagation Latency

The other metric used for the analysis is UPL which includes the following time measures,

- Update transaction time
  - The time taken to do the modification to a table in the database.
- **Update Log generation time**
  - The time taken by the candidate node to create the update log after the specified number of operations.

- **Update Log transfer time**
  - The time taken for the propagation of update log from the candidate node to base station and vise versa.

- **Update log execution time**
  - The time taken for executing the update operations specified in the update log at each candidate node. It also includes the notification of successful execution to the base station.

The update transaction time depends on the number of update transactions on a table. In the experimental setup each script with multiples of ten update transactions has been used to execute the update operations. Each command modifies two attributes of integer type. From the test, the time taken for different number of operations is depicted in the Table 5.2.

In addition, a log file is created which contains the changes made to the database from the previous log creation time. Since the size of the log file is directly proportional to the number of operations, again the update log generation time depends on the number of operations. The Table 5.3 represents the time taken for log generation at a candidate node in the proposed case.
### Table 5.2 Update Transaction Time

<table>
<thead>
<tr>
<th>Script Name</th>
<th>Number of update operations</th>
<th>Update transaction time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scione</td>
<td>10</td>
<td>0.004</td>
</tr>
<tr>
<td>Scitwo</td>
<td>20</td>
<td>0.006</td>
</tr>
<tr>
<td>Scithree</td>
<td>30</td>
<td>0.012</td>
</tr>
<tr>
<td>Scifour</td>
<td>40</td>
<td>0.016</td>
</tr>
<tr>
<td>Scifive</td>
<td>50</td>
<td>0.020</td>
</tr>
<tr>
<td>Scisix</td>
<td>60</td>
<td>0.025</td>
</tr>
<tr>
<td>Sciseven</td>
<td>70</td>
<td>0.029</td>
</tr>
<tr>
<td>Scieight</td>
<td>80</td>
<td>0.031</td>
</tr>
<tr>
<td>Scinine</td>
<td>90</td>
<td>0.036</td>
</tr>
<tr>
<td>Sciten</td>
<td>100</td>
<td>0.041</td>
</tr>
</tbody>
</table>

### Table 5.3 Update Log Generation Time

<table>
<thead>
<tr>
<th>Script Name</th>
<th>Number of update operations</th>
<th>Log Generation time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scione</td>
<td>10</td>
<td>0.02</td>
</tr>
<tr>
<td>Scitwo</td>
<td>20</td>
<td>0.02</td>
</tr>
<tr>
<td>Scithree</td>
<td>30</td>
<td>0.04</td>
</tr>
<tr>
<td>Scifour</td>
<td>40</td>
<td>0.06</td>
</tr>
<tr>
<td>Scifive</td>
<td>50</td>
<td>0.08</td>
</tr>
<tr>
<td>Scisix</td>
<td>60</td>
<td>0.09</td>
</tr>
<tr>
<td>Sciseven</td>
<td>70</td>
<td>0.12</td>
</tr>
<tr>
<td>Scieight</td>
<td>80</td>
<td>0.14</td>
</tr>
<tr>
<td>Scinine</td>
<td>90</td>
<td>0.16</td>
</tr>
<tr>
<td>Sciten</td>
<td>100</td>
<td>0.18</td>
</tr>
</tbody>
</table>
After the log file is created, it will be propagated to middleware gateway server by executing GridFTP utility. This process depends on the connectivity speed and the distance between the mobile hosts which transfer the log file and the base station. The log file transfer time is calculated based on the system time returned by the programs after execution of the globus-url-copy command for a node having 38dB SNR. Also, the same is presented in the Table 5.4. It is noticed that the file transfer time change depends on the log file size.

**Table 5.4 Update Log Transfer Time**

<table>
<thead>
<tr>
<th>Script Name</th>
<th>Number of update operations</th>
<th>Log transfer time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scione</td>
<td>10</td>
<td>0.000</td>
</tr>
<tr>
<td>Scitwo</td>
<td>20</td>
<td>0.000</td>
</tr>
<tr>
<td>Scithree</td>
<td>30</td>
<td>0.000</td>
</tr>
<tr>
<td>Scifour</td>
<td>40</td>
<td>0.001</td>
</tr>
<tr>
<td>Scifive</td>
<td>50</td>
<td>0.003</td>
</tr>
<tr>
<td>Scisix</td>
<td>60</td>
<td>0.008</td>
</tr>
<tr>
<td>Sciseven</td>
<td>70</td>
<td>0.011</td>
</tr>
<tr>
<td>Scieight</td>
<td>80</td>
<td>0.015</td>
</tr>
<tr>
<td>Scinine</td>
<td>90</td>
<td>0.019</td>
</tr>
<tr>
<td>Sciten</td>
<td>100</td>
<td>0.023</td>
</tr>
</tbody>
</table>

After receiving the update log, each candidate mobile hosts which has the replica must execute an update log file to update its local replica. The time taken for this is measured as update execution time and is shown in the Table 5.5.
Table 5.5 Update Log Execution Time

<table>
<thead>
<tr>
<th>Script Name</th>
<th>Number of update operations</th>
<th>Update log execution time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scione</td>
<td>10</td>
<td>0.11</td>
</tr>
<tr>
<td>Scitwo</td>
<td>20</td>
<td>0.11</td>
</tr>
<tr>
<td>Scithree</td>
<td>30</td>
<td>0.13</td>
</tr>
<tr>
<td>Scifour</td>
<td>40</td>
<td>0.18</td>
</tr>
<tr>
<td>Seifive</td>
<td>50</td>
<td>0.21</td>
</tr>
<tr>
<td>Scisix</td>
<td>60</td>
<td>0.25</td>
</tr>
<tr>
<td>Sciseven</td>
<td>70</td>
<td>0.30</td>
</tr>
<tr>
<td>Scieight</td>
<td>80</td>
<td>0.36</td>
</tr>
<tr>
<td>Scinine</td>
<td>90</td>
<td>0.43</td>
</tr>
<tr>
<td>Sciten</td>
<td>100</td>
<td>0.47</td>
</tr>
</tbody>
</table>

Finally, N-Agent sends an acknowledgement message to notify the successful execution to the base station. After the reception of this acknowledgement from all the candidate nodes, U-Agent destroys its instance for the particular update propagation process. By including all these calculated metrics, the UPL is calculated based on the following formula,

\[
\text{UPL} = \text{No. of. Replica} \times (\text{TS (UAgent Instance destroy)} - \text{TS (NAgent request)})
\]  

(5.1)

The system is evaluated from the point of its behavior when the number of replica changes dynamically. The Table 5.6 shows the update propagation latency for different replica and varying number of update
operations. In order to compare the performance of the proposed approach, a range based static replica allocation method is considered which has a constraint that each host should get copy of the data in at most one hop.

**Table 5.6 Update Propagation Latency**

<table>
<thead>
<tr>
<th>Number of update requests</th>
<th>Number of Replica in SRS</th>
<th>Number of Replica in DRS</th>
<th>UPL for SRS (Seconds)</th>
<th>UPL for DRS (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>6</td>
<td>1</td>
<td>0.804</td>
<td>0.134</td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>2</td>
<td>0.816</td>
<td>0.272</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>2</td>
<td>1.092</td>
<td>0.364</td>
</tr>
<tr>
<td>40</td>
<td>6</td>
<td>3</td>
<td>1.542</td>
<td>0.771</td>
</tr>
<tr>
<td>50</td>
<td>6</td>
<td>3</td>
<td>1.878</td>
<td>0.939</td>
</tr>
<tr>
<td>60</td>
<td>6</td>
<td>3</td>
<td>2.238</td>
<td>1.119</td>
</tr>
<tr>
<td>70</td>
<td>6</td>
<td>4</td>
<td>2.760</td>
<td>1.840</td>
</tr>
<tr>
<td>80</td>
<td>6</td>
<td>4</td>
<td>3.276</td>
<td>2.184</td>
</tr>
<tr>
<td>90</td>
<td>6</td>
<td>4</td>
<td>3.870</td>
<td>2.580</td>
</tr>
<tr>
<td>100</td>
<td>6</td>
<td>4</td>
<td>4.284</td>
<td>2.856</td>
</tr>
</tbody>
</table>

Based on this design criterion, replications are placed statically at the time of integration and in the test bed among sixteen mobile nodes, six mobile hosts have been selected for replica placement. It will be constant during the life time of the environment and also it will not adapt to the current status of the execution environment.
Besides, the replica manager in the proposed architecture adapts the
dynamic changes in the environment such as SNR, Read/Write counters,
storage and mobility for placement and relocation of replica. Based on the
read write pattern the number of replica is increased gradually and will be
stable for fixed read write pattern after finding the optimum number of
replicas.

The consistency performance of the proposed system is analyzed
under variations in the data access patterns. The difference between the range
based static replication scheme (referred as SRS) and the proposed dynamic
replication scheme (referred as DRS) is considered for performance
comparison. The Figure 5.11 shows the plot of UPL comparison.

![Update Propagation Latency](image)

**Figure 5.11 UPL Comparison**

From the Figure 5.11, it is inferred that the UPL of both SRS and
DRS increases with respect to the number of replica and the number of
transactions. The SRS and DRS are both performing with minimum time
difference when the number of update transactions is low. However, when the
transactions are increased, the DRS behaves well, because it considers the dynamic characteristics and maintains only the optimum number of replicas. Hence, the proposed technique scale well with the increasing number of replica sites and update transactions.

5.4 SUMMARY

Thus, the consistency maintenance with the proposed framework is realized and its working efficiency analyzed with the test bed. Evidently, the proposed enhanced agent based optimistic replica synchronization mechanism is proven to be robust in the dynamic mobile grid environment.