American Journal of Applied Sciences 4 (10): 751-758, 2007
ISSN 1546-9239
© 2007 Science Publications

Preserving Data Consistency through Neighbor Replication on Grid Daemon

[1]A.Noraziah, [2]M.Mat Deris, [3]N.A.Ahmed, [4]M.Y.M.Saman, [5]R.Norhayati, [6]Zeyad M. Alfawaer
[1]University Malaysia Pahang, Faculty of Computer System and Software Engineering,
Locked Bag 12, 25000 Kuantan, Pahang, Malaysia.
[2]University Tun Hussein Onn Malaysia, Faculty of Information and Technology Multimedia,
Parit Raja, 86400 Batu Pahat, Johor, Malaysia.
[3]Huangzhou University of Science and Technology, College of Elec. & Elect. Engineering, Wuhan, China
[4]University Malaysia Terengganu, Faculty of Science & Technology,
Mengabang Telipot, 21030 Kuala Terengganu, Terengganu, Malaysia.
[5]University Malaysia Pahang, Faculty Chemical Engineering & Natural Resources,
Locked Bag 12, 25000 Kuantan, Pahang, Malaysia.
[6]School of Information Science and Engineering Central South University, Changsha, Hunan, China

Abstract: In modern distributed systems, replication receives particular attention for providing high data availability, fault tolerance and enhance the performance of the system. It is an important mechanism because it enables organizations to provide users with access to current data where and when they need it. However, this way of data organization introduces low data consistency and data coherency as more than one replicated copies need to be updated. Expensive synchronization mechanisms are needed to maintain the consistency and integrity of data among replicas when changes are made by the transactions. In this paper, we present Neighbor Replication on Grid (NRG) daemon in order to manage replication and transactions in distributed system. NRG Transaction Model has been implemented in order to preserve the data consistency and availability. Based on experiment and result, it shows that NRG daemon guarantees consistency and obey serializability through the synchronization approach.

Key word: Distributed system, replication, consistency, synchronous, transaction, serializability.

INTRODUCTION

In modern distributed systems, replication receives particular attention for providing high data availability, fault tolerance and enhance the performance of the system[1, 2, 3]. It is an important mechanism because it enables organizations to provide users with access to current data where and when they need it. The failure of system can be transparent from users and applications if they can obtain data from an identical replica. Replication can improve performance by scaling the number of replicas with demand and by offering nearby copies to services distributed over the network.

An ideal distributed file system provides applications strict consistency, i.e., a guarantee that all I/O operations yield identical results at all nodes at all times [4, 5]. In a replication system, the value of each logical item is stored in one or more physical data items, referred to as its *copies*[5]. Each read or write operation on a logical data item must be mapped to corresponding operations on physical copies. Of course this way of data organization introduces low data consistency and data coherency as more than one replicated copies need to be updated. Expensive synchronization mechanisms are needed to maintain the consistency and integrity of data among replicas when changes are made by the transactions. This suggests that proper strategies are needed in managing replication and transactions in distributed systems.

There are many examples of replication schemes in distributed file and database systems. Among them are based on synchronous replication[6, 7, 8], which deploy quorum to execute the operations with high degree of consistency and ensure serializability. Synchronous replication can be categorized into several schemes, i.e., all -data-to-all-sites (full replication) and some-data-items-to-all-sites. However, full replication causes high update propagation, high storage capacity and difficult to maintain the data consistency[1, 9, 10]. A few studies have been done on partial replication techniques based on some data items to all sites using tree structure technique[11, 12]. This technique will cause high update

Corresponding Author: Noraziah Ahmad, University Malaysia Pahang, Faculty of Computer System and Software Engineering, Locked Bag 12, 25000 Kuantan, Pahang, Malaysia propagation overhead. Thus, some-data-items-to-all-sites scheme is not realistic. Furthermore, in many applications, there is update-intensive data, which should be replicated to very few sites. The European DataGrid Project[13] implemented this model to manage the file-based replica. It is based on the sites that have previously been registered for replication. This will cause the inconsistence number of replication occurs in the model. Also, the data availability has very high overhead as all registered replicas must be updated simultaneously.

In this paper, we present Neighbor Replication on Grid (NRG) daemon to manage replication and transactions in order to preserve data consistency and maintain data availability in distributed system. NRG daemon guarantees data consistency and obey serializability through the synchronize replication. The mechanisms for locating and managing replicas, as well as performance details can be found in our previous work[2, 8].

NRG Transaction Model: In this section, we recall the NRG Transaction Model. The following notations are defined:

a) T is a transaction.

b) α and β are groups for the transaction T.

c) $\gamma = \alpha$ or β where it represents different group for the transaction T (before and until get quorum).

d) T_α is a set of transactions that comes before T_β, while T_β is a set of transactions that comes after T_α.

e) D is the union of all data objects managed by all transactions T of NRG and x represents one data object (or data file) in D to be modified by an element of T_α and T_β.

f) *Target set* = {-1, 0, 1} is the result of transaction T (see Table 1).

g) *NRG transaction elements* $T_\alpha = \{T_{\alpha_{x,q_r}} \mid r = 1, 2, ..., k\}$ where $T_{\alpha_{x,q_r}}$ is a *queued* element of T_α transaction.

h) *NRG transaction elements* $T_\beta = \{T_{\beta_{x,q_r}} \mid r = 1, 2, ..., k\}$ where $T_{\beta_{x,q_r}}$ is a *queued* element of T_β transaction.

i) *NRG transaction elements* $T_\gamma = \{T_{\gamma_{x,q_r}} \mid r = 1, 2, ..., k\}$ where $T_{\gamma_{x,q_r}}$ is a *queued* element either in different set of transactions T_α or T_β.

j) $T'_{\gamma_{x,q_1}}$ is a transaction that is transformed from $T_{\gamma_{x,q_1}}$.

k) $T_{\mu_{x,q_1}}$ represents the transaction feedback from a neighbor site. $T_{\mu_{x,q_1}}$ exists if either $T_{\gamma_{x,q_1}}$ or $T'_{\gamma_{x,q_1}}$ exists.

l) Successful transaction at primary site $T(\gamma_{x,q_1}) = 0$, where $\gamma_{x,q_1} \in D$ (i.e., the transaction locked a data x at primary). Meanwhile, successful transaction at neighbor site $T(\mu_{x,q_1}) = 0$, where $\mu_{x,q_1} \in D$ (i.e., the transaction locked a data x at neighbor).

Table 1: Meaning of Target Set.

Target set	Meaning
0	This means that no failure occurred during NRG transaction's execution. By $T(\gamma_{x,q_1}) = 0$, where $\gamma_{x,q_1} \in D$ is the data object processed by the transaction at primary site and $\gamma = \alpha, \beta$. The transaction was successful (i.e., the transaction locked the data file x at primary). *Write counter* track this value. For neighbor site, $T(\mu_{x,q_1}) = 0$ where $\mu_{x,q_1} \in D$.
1	This means *accessing failure*. By $T(\gamma_{x,q_1}) = 1$, we mean that the destination server could not perform the job. Data file x managed by the primary site is already locked. The transaction has not executed. For neighbor site, $T(\mu_{x,q_1}) = 1$, where $\mu_{x,q_1} \in D$.
-1	This means *unknown status*. By $T(\mu_{x,q_1}) = -1$, we mean that the neighbor site cannot tell if the NRG transaction has or has not been executed yet. This could happen when the destination host is down, or the link between primary and neighbor site is down, or both of the situations. In that case, the NRG request transaction or the message may be lost. So we do not know if the transaction has been executed or not at neighbor site. This will be tracked by *unknown status counter*.

Four phases involve in NRG transaction semantic, which are initiate lock; propagate lock and obtain a quorum; release lock, update and commit data; and handling failure (unknown status). Fig. 1 shows the framework of semantics of NRG Transaction Model.

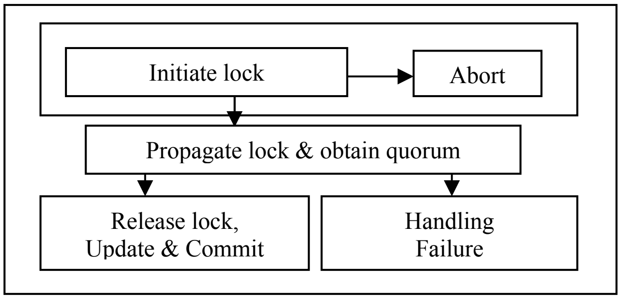


Fig. 1: Framework of semantics of NRG Transaction.

NRG Daemon: NRG daemon has been developed in order to give a better intuition on how to manage replication and transactions through NRG Transaction Model. A daemon is defined as a computer program that runs in the background and ready to perform without user input[14]. Usually, it provides some services either for the system as a whole or for the user applications. NRG daemon is started (and stopped) when a system changes the run levels. It is ordinarily starts when a system boots and runs until system shutdown, unless it forcibly terminated. In particular, it has three system components:

a) NRG Transaction Manager (NTM): Each primary or neighbor replica has its own NRG Transaction Manager (NTM). Every transaction goes through the NTM before it will be processed. The NTM functions include:

- Accepting a set of transactions from clients either $T_\alpha = \{T_{\alpha_{x,q_r}} \mid r = 1, 2, \ldots, k\}$ or $T_\beta = \{T_{\beta_{x,q_r}} \mid r = 1, 2, \ldots, k\}$. When $T_{\gamma_{x,q_1}}$, $\gamma = \alpha, \beta$ come concurrently, queue them based on the small arrival rate.
- Receiving all types of transaction $T_{\gamma_{x,q_1}}$, $\gamma = \alpha, \beta$ from clients, $T'_{\gamma_{x,q_1}}$ from primary and neighbor replica. Each transaction goes through the NTM for the purpose of the determination of type of replica (either to be as primary or neighbor replica processing).
- Performing synchronous commit $T'_{\gamma_{x,q_1}}$ after user has finished update the data. After that, it unlock data file x.
- If a replica is required to release a lock from another primary replica, it aborts $T_{\gamma_{x,q_1}}$ at its replica and rollbacks all the transactions.

b) Receiving Agent: It functioning as listed below:

- Monitoring the users' status access for a particular data file. If any transactions request that particular data, then it automatically redirect output from the command line editing (by using *ps aux* command) to *user_act* log files.
- Data manipulation. The a*wk utility* has been used for filtering the data.
- Recognizes the transaction that obtains a lock.
- Initiates the server status.
- Handles an access permission mode of the particular requested data file x.
- Detects the transactions that must be aborted. Kernel aborts those transactions.
- Compress and decompress the data files.
- Handles job control.

c) Sending Agent: It functioning as follow:

- Requests the neighbor replicas status.
- Propagates a lock synchronously to neighbor replicas.
- Checks the current write and unknown status counters to detect whether the transaction must perform or still require obtaining a quorum.
- Sends the updated counters to replicas.
- If the transaction gets a quorum, releases neighbor's locks for the neighbors that already in other quorum(s).
- Replicate data to neighbors for particular data item x.

NRG daemon runs with the *superuser* privilege. This is because it must access to some sort of the privilege resources such as the configuration files. The daemon runs in the background and does not have a controlling terminal. In particular, it has been configured to be automatically functioning without human intervention.

RESULTS AND DISCUSSION

In this experiment, we will consider no failures during the transaction execution. In remainder of this section, the experiment involves phases in NRG transaction semantic. Without lost of generality, this experiment shows how to preserve the consistency of the same particular data file. As long as the same data is used, one-copy-serializability must be obeyed for all the transaction executions. In addition, it also shows that the data always available and reliable.

To demonstrate NRG Transaction Model, 3 replication servers are deployed. Each server or node is connected to one another through a fast Ethernet switch hub. Replica A with IP 192.168.100.21, replica B with IP 192.168.100.36 and replica D with IP 192.168.100.39 locate data *a*. Table 2 shows the Primary-Neighbors Grid Coordination (PNGC) for replica A, B and D, which will be used by NRG daemon.

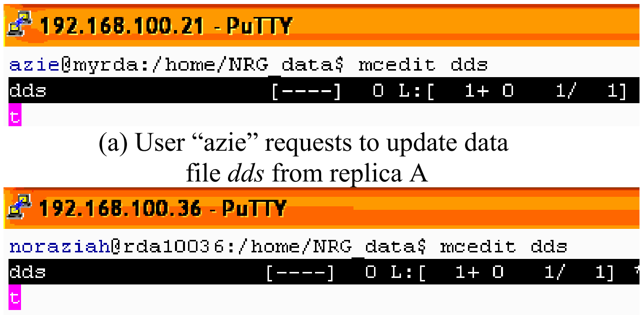
Table 2: Primary-Neighbors Grid Coordination

PRIMARY	NEIGHBOURS	
A: 192.168.100.21	B: 192.168.100.36	D: 192.168.100.39
B: 192.168.100.36	D: 192.168.100.39	A: 192.168.100.21
D: 192.168.100.39	A: 192.168.100.21	B: 192.168.100.36

The experiment of NRG daemon program was done in shell programming and Perl integrated with File Transfer Protocol (FTP) for the communications agent. Bourne Again Shell is selected since it riches with command-line editing facilities and jobs control capabilities. The job control provides greater flexibility in dealing with background processes. Meanwhile, an automated FTP is used in shell programming for sending agent. Red Hat Linux Kernel release 9 and Linux Slackware 2.4.2 are used as a platform to the replicated servers. All applications for users are available to these particular Linux platforms. As such, the applications for users include *mcedit, vi* and *vim editor*.

To simplify a clearer presentation of these experiments, assume that the transactions come to access particular data file *a*. Neighbor binary voting assignment [2] is initiated where $S(B_a) = \{i \mid B_a(i) = 1, 1 \le i \le n\}$ and $B_a(i)$ is the vote assign to site *i,* which has a particular data *a*. Hence, $B_a(A) = B_a(B) = B_a(D) = 1$ with $S(B_a) =$

{A,B,D}. In particular, the clients can also request the data file *a* at any other replica of *S*(B_a)' but use the transparent remote shell (i.e., secure shell) to the replica of *S*(B_a). The smallest total number to be replicated, *d* = 3 has been chosen because it easy to manage the transactions with a small pre-emptive lock, in order to get the write quorum. In particular, the write quorum must be more than a majority quorum. Since the transaction is proportional to the quorum size [8], less synchronization time is required for the transaction execution with a small pre-emptive lock. The transactions execution for any data on other servers is evaluated in the same manner. In particular with NRG Transaction Model, $T_{\gamma_{x,q_1}}$ represents $T_{\gamma_{a,q_1}}$. Two different sets of transactions, $T_\alpha = \{T_{\alpha_{a,q_r}} \mid r = 1,2,...,k\}$ and $T_\beta = \{T_{\beta_{a,q_r}} \mid r = 1,2,...,k\}$ request to update data file *a* at replica A and B in the absence of system failures. Users concurrently request to update the data file *a* (namely as *dds*) from primary replica A and B.



```
azie@myrda:/home/NRG_data$ mcedit dds
dds                    [----]   0 L:[  1+ 0   1/   1]
t
```

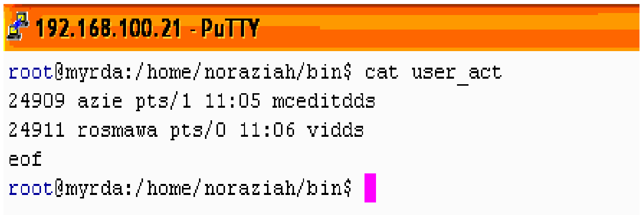
(a) User "azie" requests to update data file *dds* from replica A

```
noraziah@rda10036:/home/NRG_data$ mcedit dds
dds                   [----]   0 L:[  1+ 0   1/   1]
t
```

(b) User "noraziah" requests to update data file *dds* from replica B

Fig. 2: Users concurrently request data file *dds*

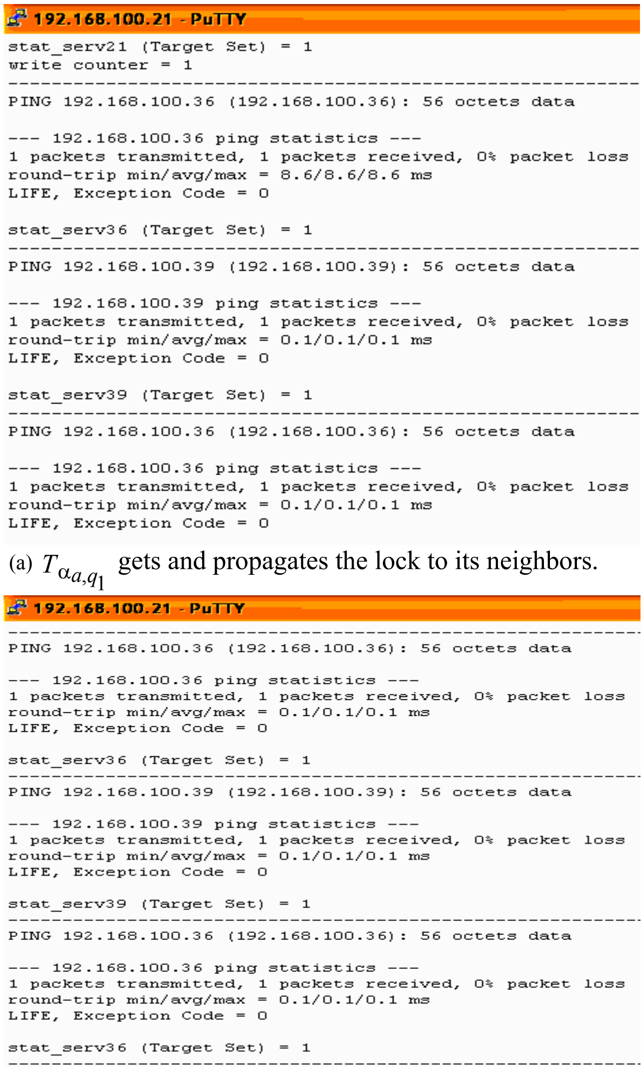
NRG daemon for primary replica A and B monitor all users current status that access particular data *a*. If any user accesses that data, then it redirects the user's information such as the *pid, user name, tty, log time* and *access editor* to the log information. NRG daemon manipulates its log information by using *awk utility*. The user's information that access the data file *dds* at the primary replica A and B are showed in Fig. 3a and 3b respectively. In particular, each primary replica has the *user_act* log file.



```
root@myrda:/home/noraziah/bin$ cat user_act
24909 azie pts/1 11:05 mceditdds
24911 rosmawa pts/0 11:06 vidds
eof
root@myrda:/home/noraziah/bin$
```

Fig. 3: User's information at primary replica A

NTM of primary replica A and B pass to their primary_replica_processing function for recognizing which transaction gets the lock. $T_{\gamma_{x,q_1}}$ is recognized based on its login time. Since several transactions come to access the data file *dds* (data *a*), the first queued element obtains the lock. At primary replica A, $T_{\alpha_{a,q_1}}$ with *pid* 24909 gets the lock (refer Fig. 3). The server status is initiated to 1 for its Target Set as shows in Fig. 4a. Fig. 4a and Fig. 4b show $T_{\alpha_{a,q_1}}$ performs during an initialization and propagation lock phases.



```
stat_serv21 (Target Set) = 1
write counter = 1
-----------------------------------------------------------------
PING 192.168.100.36 (192.168.100.36): 56 octets data

--- 192.168.100.36 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 8.6/8.6/8.6 ms
LIFE, Exception Code = 0

stat_serv36 (Target Set) = 1
-----------------------------------------------------------------
PING 192.168.100.39 (192.168.100.39): 56 octets data

--- 192.168.100.39 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.1/0.1/0.1 ms
LIFE, Exception Code = 0

stat_serv39 (Target Set) = 1
-----------------------------------------------------------------
PING 192.168.100.36 (192.168.100.36): 56 octets data

--- 192.168.100.36 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.1/0.1/0.1 ms
LIFE, Exception Code = 0
```

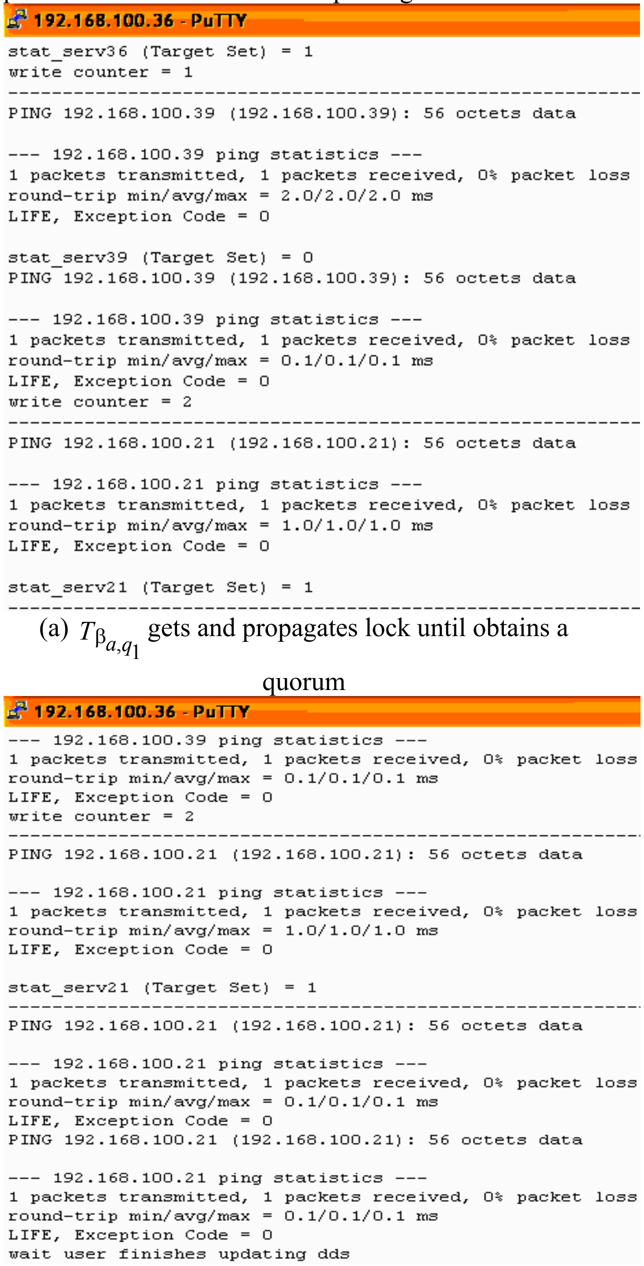
(a) $T_{\alpha_{a,q_1}}$ gets and propagates the lock to its neighbors.

```
-----------------------------------------------------------------
PING 192.168.100.36 (192.168.100.36): 56 octets data

--- 192.168.100.36 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.1/0.1/0.1 ms
LIFE, Exception Code = 0

stat_serv36 (Target Set) = 1
-----------------------------------------------------------------
PING 192.168.100.39 (192.168.100.39): 56 octets data

--- 192.168.100.39 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.1/0.1/0.1 ms
LIFE, Exception Code = 0

stat_serv39 (Target Set) = 1
-----------------------------------------------------------------
PING 192.168.100.36 (192.168.100.36): 56 octets data

--- 192.168.100.36 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.1/0.1/0.1 ms
LIFE, Exception Code = 0

stat_serv36 (Target Set) = 1
-----------------------------------------------------------------
```

(b) $T_{\alpha_{a,q_1}}$ keeps propagating its lock to get a quorum

Fig. 4: $T_{\alpha_{a,q_1}}$ performs during an initialization and propagation lock phases.

Next, NRG daemon kills *pid* of $T_{\alpha_{a,q_2}}$. Kernel broadcasts message to acknowledge. Server status is initiated to 1 for its Target Set as depicts in Fig. 5a. Fig. 5 show $T_{\beta_{a,q_1}}$ performs from an initialization lock phase until wait user finishes updating data file *dds*.



```
192.168.100.36 - PuTTY
stat_serv36 (Target Set) = 1
write counter = 1
--------------------------------------------------------------
PING 192.168.100.39 (192.168.100.39): 56 octets data

--- 192.168.100.39 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 2.0/2.0/2.0 ms
LIFE, Exception Code = 0

stat_serv39 (Target Set) = 0
PING 192.168.100.39 (192.168.100.39): 56 octets data

--- 192.168.100.39 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.1/0.1/0.1 ms
LIFE, Exception Code = 0
write counter = 2
--------------------------------------------------------------
PING 192.168.100.21 (192.168.100.21): 56 octets data

--- 192.168.100.21 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 1.0/1.0/1.0 ms
LIFE, Exception Code = 0

stat_serv21 (Target Set) = 1
--------------------------------------------------------------
```

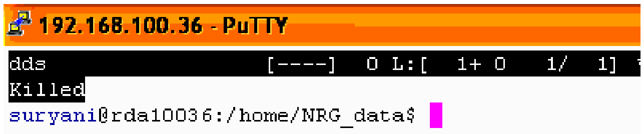
(a) $T_{\beta_{a,q_1}}$ gets and propagates lock until obtains a quorum

```
192.168.100.36 - PuTTY
--- 192.168.100.39 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.1/0.1/0.1 ms
LIFE, Exception Code = 0
write counter = 2
--------------------------------------------------------------
PING 192.168.100.21 (192.168.100.21): 56 octets data

--- 192.168.100.21 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 1.0/1.0/1.0 ms
LIFE, Exception Code = 0

stat_serv21 (Target Set) = 1
--------------------------------------------------------------
PING 192.168.100.21 (192.168.100.21): 56 octets data

--- 192.168.100.21 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.1/0.1/0.1 ms
LIFE, Exception Code = 0
PING 192.168.100.21 (192.168.100.21): 56 octets data

--- 192.168.100.21 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.1/0.1/0.1 ms
LIFE, Exception Code = 0
wait user finishes updating dds
```

(b) $T_{\beta_{a,q_1}}$ obtains quorum and releases lock $T_{\alpha_{a,q_1}}$.

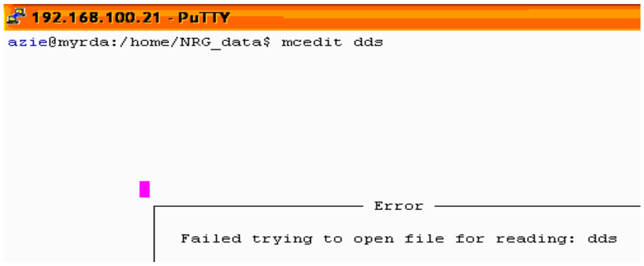
Fig. 5: $T_{\beta_{a,q_1}}$ performs during initiates lock until wait user finishes update data.

Next, NRG daemon kills *pid* of $T_{\beta_{a,q_2}}$, which is the *pid* 24897. After that, kernel broadcasts the messages to user "suryani", as depicts in Fig. 6. Since $T_{\alpha_{a,q_1}}$ at replica A and $T_{\beta_{a,q_1}}$ at replica B obtain the locks, NRG daemon controls the access permission mode of the data file *dds*. Hence, other transactions cannot read or update it at that time as shows in Fig. 7. The error message is generated automatically by the kernel.



```
192.168.100.36 - PuTTY
dds                    [----]  0 L:[  1+ 0   1/  1]
Killed
suryani@rda10036:/home/NRG_data$
```

Fig. 6: NRG daemon kills *pid* of $T_{\beta_{a,q_2}}$ at replica B.



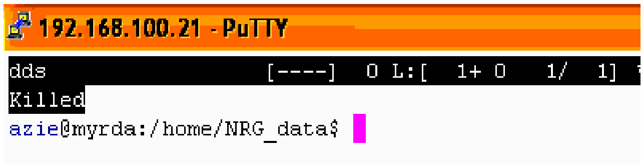
```
192.168.100.21 - PuTTY
azie@myrda:/home/NRG_data$ mcedit dds



                    Error
        Failed trying to open file for reading: dds
```

Fig. 7: The data file *dds* is locked by NRG daemon

Primary replica A and B propagate lock synchronously to its neighbor replicas based on the PNGC (refer Table 2). Primary replica processing for $T_{\alpha_{a,q_1}}$ propagates the locks to its neighbor replicas B and D as depicts in Fig. 4a. It keeps propagates the lock as shows in Fig. 4b. This is because, $T_{\alpha_{a,q_1}}$ still not get a quorum. Meanwhile, the primary replica processing for $T_{\beta_{a,q_1}}$ propagates the locks to its neighbor replicas D and A as depicts in Fig. 5a. Each NTM of neighbor replica calls ***neighbor_replica_processing*** function to check its feasibility lock and sends feedback to the primary.

The first transaction that obtains a quorum denoted as $T'_{\gamma_{a,q_1}}$ released other $T_{\gamma_{a,q_1}}$. In this experiment, $T_{\beta_{a,q_1}}$ obtains a majority quorum when the write counter is equal to two, as depicts in Fig. 5a. Therefore, $T_{\beta_{a,q_1}}$ becomes $T'_{\gamma_{a,q_1}}$. Next, $T'_{\gamma_{a,q_1}}$ at primary replica B releases the lock of $T_{\alpha_{a,q_1}}$ at primary replica A.

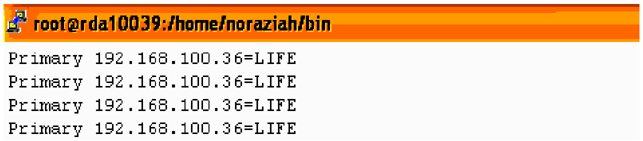
Hence, $T_{\alpha_{a,q_1}}$ is aborted as shows in Fig. 8. Consequently, $T'_{\gamma_{a,q_1}}$ gets all locks from $S(B_a)$ at primary replica B, as depict in Fig. 5a and Fig. 5b.



```
dds                     [----]  0 L:[  1+ 0   1/   1]
Killed
azie@myrda:/home/NRG_data$
```

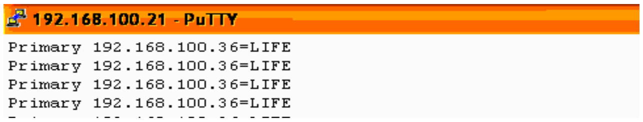
Fig. 8: $T_{\alpha_{a,q_1}}$ is aborted at primary replica A.

The previous primary replica processing for $T_{\alpha_{a,q_1}}$ becomes as neighbor replica processing for $T_{\beta_{a,q_1}}$. Primary replica B obtains lock from neighbor replica D and A as show in Fig. 9a and Fig. 9b respectively.



```
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
```

(a) Primary replica B obtains lock from neighbor replica D

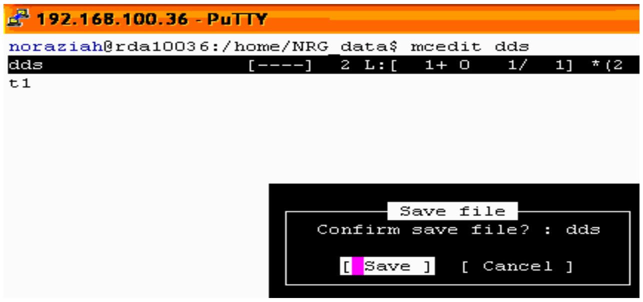


```
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
```

(b) Primary replica B obtains lock from neighbor replica A

Fig. 9: Primary replica B obtains lock from neighbor replica D and A

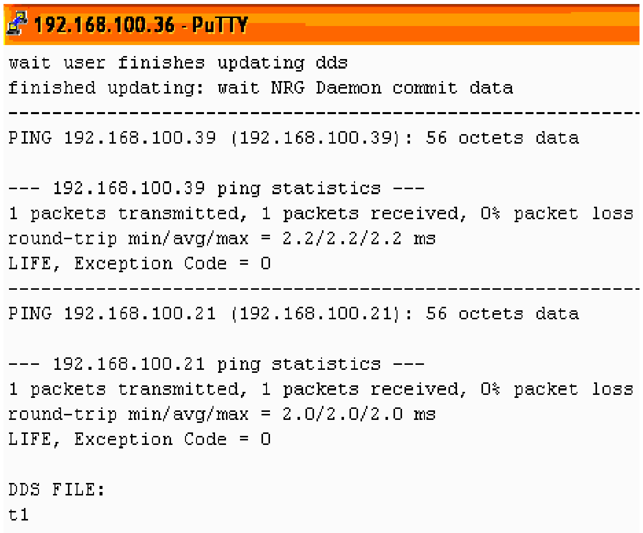
Next, NRG daemon changes an access permission mode of data file *dds* at primary replica B. Therefore, user "noraziah" can start modifying the contents of data file *dds* as depicts in Fig. 10.



```
noraziah@rda10036:/home/NRG_data$ mcedit dds
dds                  [----]  2 L:[  1+ 0   1/   1] *(2
t1



                                 Save file
                          Confirm save file? : dds

                          [ Save ]   [ Cancel ]
```

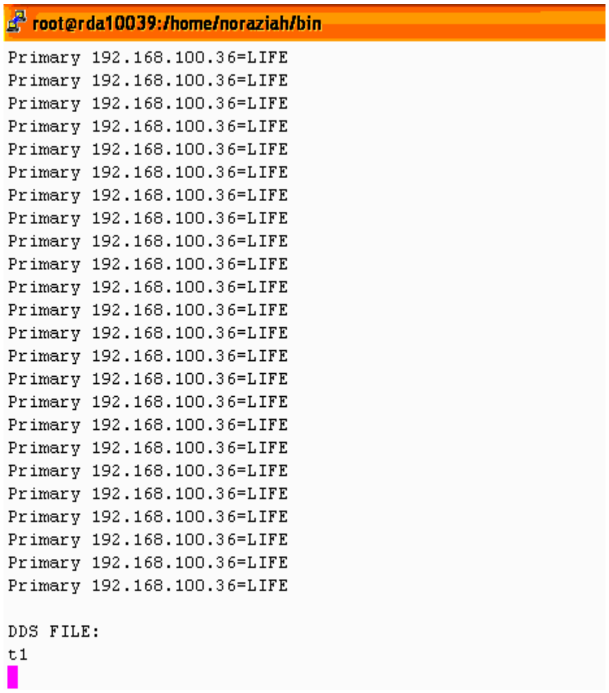
Fig. 10**:** User "noraziah" updates the data file *dds*

When user finished updating data, all replicas of $S(B_a)$ commit $T'_{\gamma_{a,q_1}} \in T_\beta$ synchronously. Fig. 11a, Fig. 11b and Fig. 11c show $T'_{\gamma_{a,q_1}}$ change is committed at primary replica B, neighbor replicas D and A respectively.



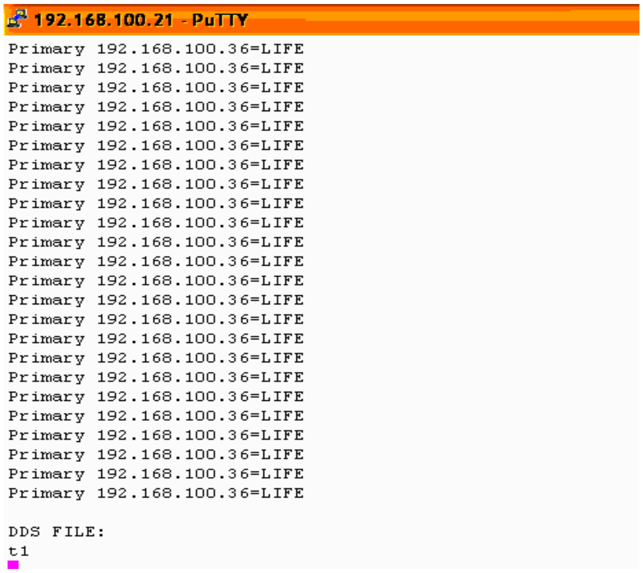
```
wait user finishes updating dds
finished updating: wait NRG Daemon commit data
------------------------------------------------------------
PING 192.168.100.39 (192.168.100.39): 56 octets data

--- 192.168.100.39 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 2.2/2.2/2.2 ms
LIFE, Exception Code = 0
------------------------------------------------------------
PING 192.168.100.21 (192.168.100.21): 56 octets data

--- 192.168.100.21 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 2.0/2.0/2.0 ms
LIFE, Exception Code = 0

DDS FILE:
t1
```

(a) $T'_{\gamma_{a,q_1}}$ change is committed at primary replica B



```
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE

DDS FILE:
t1
```

(b) $T'_{\gamma_{a,q_1}}$ change is committed at neighbor replica D.



```
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE
Primary 192.168.100.36=LIFE

DDS FILE:
t1
```

(c) $T'_{\gamma_{a,q_1}}$ change is committed at neighbor replica A.

Fig. 11: $T'_{\gamma_{a,q_1}}$ change is committed at replicas of $S(B_a)$.

Finally, NRG daemon changes an access permission mode to unlock data file *dds*. Hence, users can read or request to update it at any replica of $S(B_a)$. Table 3 simplifies the result of how NRG handle concurrent transactions $T_{\alpha_{a,q_1}}$ and $T_{\beta_{a,q_1}}$ at all replica of $S(B_a)$.

Table 3: The experiment result of how NRG handle concurrent transactions

REPLICA TIME	A	B	D
t1	unlock(a)	unlock(a)	unlock(a)
t2	begin_transaction	begin_transaction	
t3	write lock(a) counter_w(a)=1	write lock(a) counter_w(a)=1	
t4	wait	wait	
t5	propagate lock:B	propagate lock:D	

t6	propagate lock:D		lock(a)
			from B

t7	propagate lock:B	get lock:D counter_w(a)=2	
t8	propagate lock:D	obtain quorum release lock: A	
t9	abort $T_{\alpha_{a,q_1}}$ & rollback, lock(a) from B		
t10		update a	
t11	commit $T'_{\gamma_{a,q_1}} \in T_\beta$	commit $T'_{\gamma_{a,q_1}} \in T_\beta$	commit $T'_{\gamma_{a,q_1}} \in T_\beta$
t12	unlock(a)	unlock(a)	unlock(a)

CONCLUSION

A fundamental challenge with replication is to maintain data consistency among replicas in distributed systems. The data organization through replication introduces low data consistency and coherency as more than one replicated copies need to be updated. Expensive synchronization mechanisms are needed to maintain the consistency and integrity of data among replicas when updates are made by the transactions. Furthermore, timeliness in synchronization has become show stopper to maximize the usage of system but at the same time contribute to the consistent and reliable computing. NRG Transaction Model resolves this challenge by alleviates lock with small quorum size before capturing update and commit transaction synchronously to the sites that require the same update data item. In particular, we have developed NRG daemon to manage replication and transactions in distributed system. We focus on NRG daemon that guarantees consistency and obey serializability through the synchronize replication. Based on experiment and result, it shows that NRG daemon solves the distributed concurrency transactions and guarantees the data consistency in distributed systems. This is due to the transaction execution is equivalent to one-copy-serializability.

REFERENCES

1. L. Gao, M. Dahlin, A. Nayate, J. Zheng and A. Iyengar, 2005. Improving Availability and Performance with Application-Specific Data Replication: IEEE Trans. Knowledge and Data Engineering, 17(1): 106-200.
2. M. Mat Deris, D. J. Evans, M. Y. Saman, A. Noraziah, 2003. Binary Vote Assignment on Grid For Efficient Access of Replicated Data: Intl. Journal of Computer Mathematics, Taylor and Francis, 80(12): 1489-1498.
3. M. Tang, B. S. Lee, X. Tang and C. K. Yeo, 2006. The impact on data replication on Job Scheduling Performance in the Data Grid: Intl. Journal of Future Generation of Computer Systems, Elsevier, 22: 254-268.
4. P.Bernstein, N.Goodman, 1984. The failure and recovery problem for replicated distributed databases: ACM TODS.
5. J. Zhang and P. Honeyman, 2004. Replication Control in Distributed File Systems: CITI Technical Report 04-01, University of Michigan.

6. J. Holliday, R. Steinke, D. Agrawal and A. El-Abbadi, 2003. Apidemic Algorithms for Replicated Databases: IEEE Trans. On Know. and Data Engineering, 15(3): 1-21.
7. H. Stockinger, 2001. Distributed Database Management Systems and The Data Grid: IEEE-NASA Symposium, 1: 1-12.
8. A. Noraziah, M. Mat Deris, R. Norhayati, M.Y.M. Saman, M. Rabiei, W.N. Shuhadah, 2006. Managing Neighbour Replication Transactions in Distributed System: Intl. Symposium on Distributed Computing and Applications Business Engineering and Science, 1: 95-101.
9. Budiarto, S. Noshio, M. Tsukamoto, 2002. Data Management Issues in Mobile and Peer-to-Peer Environment: Data and Knowledge Engineering, Elsevier, 41:183-204.
10. M. Mat Deris, J.H. Abawajy, H.M. Suzuri, 2004. An Efficient Replicated Data Access Approach for Large Scale Distributed Systems: IEEE/ACM Conf. On Cluster Computing and Grid (CCGRID2004).
11. Nicola and M. Jarke, 2000. Performance Modeling of Distributed and Replicated Databases: IEEE Trans. On Knowledge and Data Engineering, 12(4): 645-671.
12. J.Huang, Qingfeng Fan, Qiongli Wu and YangXiang He, 2005. Improved Grid Information Service Using The Idea of File-Parted Replication: Lecture Notes in Computer Science Springer, 3584: 704-711.
13. P. Kunszt, Erwin Laure, Heinz Stockinger, Kurt Stockinger, 2005. File-based Replica Management, Intl. Journal of Future Generation of Computer Systems, Elsevier, 21: 115-123.
14. J.J.Tackett, D.Gunter, L.Brown, 1995. Special Edition Using Linux. Que Corporation USA.