Fault-Tolerant Quorum Consensus Scheme
for Replication Control
in Mobile Distributed Database Systems: FTQC

Siwoo Byun
Songchun Moon

Korea Advanced Institute of Science and Technology

Abstract
We propose a new replication control scheme for multiple-copy consistency in mobile distributed database systems. Replicating data at multiple sites could contribute to enhance the availability regarding the distributed data. Replicating data, however, inevitably induces the burden of maintaining replica consistency which requires more complex synchronization mechanism, especially in the presence of communication failures. If a communication link fails, all sites in the overall network are divided into two disjoint groups: the major group called quorum partition and the minor group called non-quorum partition. Note that this phenomenon called network partitioning could lead to access starvation in the non-quorum partition. In order for mobile users to access global databases without access starvation even in the presence of network partitioning, we propose a new scheme called fault-tolerant quorum consensus (FTQC). FTQC is based on the idea that quorum formation is allowed even in the non-quorum partition by circulating a single global token. FTQC guarantees a new form of one-copy serializability, inter-partition one-copy serializability, without sacrificing data availability. Based on the results of the performance evaluation, we conclude that the protocols which exploit FTQC scheme outperform the protocols which never exploit FTQC.

Keywords: mobile database, network partitioning, replicated database management
Introduction

Recent advances in computing and networking technologies have made an extensive use of inexpensive portable computers and enabled sharing of on-line information via wireless communication channels. This new computing paradigm, called mobile computing, allows users to perform on-line transaction processing independent of their physical locations [Alonso and Korth, 1993; Imielinski and Badrinath, 1992; Pitoura and Bhargava, 1994]. Generally, such a mobile computing architecture includes two distinct sets of entities: mobile hosts (MHs) in the wireless network and fixed hosts (FHs) in the wired network (Fig. 1).

![Mobile System Architecture](image)

Legends- LS: Location Server; MSS: Mobile Support Station; MH: Mobile Host; FH: Fixed Host

**Fig. 1** Mobile System Architecture
The MHs can dynamically move within a radio coverage area called a *cell* or between two cells while retaining their network connection. The average cell size is about two miles in diameter, and an MH is crossing through these cells tens of times a day. The FHs are steadily connected to the wired network and some of them, called *mobile support stations*(MSSs), are augmented with a wireless interface to communicate with the MHs. Normally, a single MSS is able to support a number of MHs, and is engaged to provide services such as data passing and message interpretation to the MHs positioned only within its cell. A *location server*(LS) is also a FH that is responsible for keeping track of addresses of MHs to detect the geographical location where the MH is located. Generally, there is a hierarchy of LSs which are connected among themselves and to MSSs by the wired network. Each MH includes several applications such as groupwork tool and one small DBMS which performs basic tasks to manage database consistency regarding transactions issued by the local applications.

The users could request information and receive responses at any places owing to the MH. The MH, however, could suffer from unreliable and ill-timed services due to the characteristics of wireless media. Since the current wireless network generally has unreliable channel and narrow bandwidth, communications between MHs and an MSS are prone to be delayed due to the low transfer rate, and often disconnected due to unpredictable electronic-noise interventions. One way that reduces possibility of the undesirable services is replicating data at multiple MHs.

Replicating data at multiple independent sites could contribute to enhance the availability regarding the distributed data. Replicating data, however, inevitably induces the burden of maintaining replica consistency which requires more complex and more expensive fault-tolerant synchronization mechanism, especially in the presence of communication failures. If a
communication link fails, all sites in the overall network are divided into two disjoint groups: the major group called \textit{quorum partition} and the minor group called \textit{non-quorum partition}. In this case, the sites in the same partition may continue to communicate with each other, but unfortunately no communication can occur between sites in different partitions. This phenomenon is called \textit{network partitioning}. In the presence of network partitioning, the sites in each partition may continue to execute transactions which access only replicas in the same partitions, but executing transactions which access replicas in the different partitions becomes impossible.

Replication control schemes (RCSs) for dealing with multiple-copy consistency in the literature can be divided into two categories: \textit{optimistic} and \textit{pessimistic}. Optimistic schemes [Bernstein and Goodman, 1984; Davidson, 1984] achieve high data availability by allowing more than one partition group to access replica, and any resulting inconsistencies are resolved later at the time of partition reconnection. Pessimistic schemes [Agrawal and Abbadi, 1990; Cheung et al., 1990; Herlihy, 1986; Jajodia and Mutchler, 1990; Kumar, 1991], on the other hand, achieve multiple-copy consistency by rigorous enforcing \textit{one-copy serializability} (ISR) in a way that only a single particular partition group is allowed to access replica, and thus data availability for the other is accordingly limited. A number of studies have been attempted in the pessimistic approach rather than in the optimistic approach in the sense that the high availability of replicas without maintaining consistency is nothing but a meaningless goal, and ensuring ISR became a mandatory requirement for maintaining database consistency in the pessimistic schemes.
Motivation

Most pessimistic RCSs are instances of *quorum consensus* scheme [Herlihy, 1986] which is considered to be the most well-known RCS in the sense that it is able to tolerate communication failure and guarantees 1SR. However, if network partitioning occurs due to communication failure, non-quorum sites suffer from *access starvation*, since only the sites which have successfully formed *read*(or *write*) *quorum* in the quorum partition possess a exclusive privilege to access.

This sort of access starvation in the non-quorum partition happens to continue until the communication link which caused the network partition failure is recovered. This sort of access starvation could also occur in mobile replicated database environment when the links between the MSS and the LS fail. This is illustrated in Example 1. Consider that an MSS is a stationary server that communicates directly with MHs through wireless radio links(Fig. 2). MSS<sub>i</sub> is supposed to serve MH<sub>i</sub> in Cell<sub>i</sub>, MSS<sub>2</sub> for MH<sub>2</sub> and MH<sub>3</sub> in Cell<sub>2</sub>, and MSS<sub>3</sub> for MH<sub>4</sub> and MH<sub>5</sub> in Cell<sub>3</sub>. An MH has its unique identifier, *mobile-id*, and a cell has its unique identifier, *cell-id*. An LS is also a stationary server that is responsible for keeping track of a pair, (*mobile-id*, *cell-id*), to detect the geographical location where the MH is located. For instance, in Fig. 2, LS<sub>i</sub> is responsible for MSS<sub>1</sub>, MSS<sub>2</sub>, and MSS<sub>3</sub>.
Example 1 (Access Starvation Problem in A General Quorum Consensus Approach):

Suppose that there are five copies, \( \{x_1, x_2, x_3, x_4, x_5\} \), of \( x \). Let us assign a non-negative weight, for instance, 1, to each copy of \( x \) for simplicity in QC algorithm. Thus, the total votes of all copies, denoted by \( V_{\text{total}} \), is 5. Whenever an MH executes a write transaction, it must collect write-affirmative votes, denoted by \( V_w \), from the other MHs. It then has to satisfy \( V_w > \frac{V_{\text{total}}}{2} \) to form write quorum. This \( \frac{V_{\text{total}}}{2} \) is called write threshold (WT) and is 2.5, in this case. In case a wired link failure unfortunately occurs in \( L_3 \), this shall partition the global database into two independent components, \( P_a = \{MH_1, MH_2, MH_3\} \) and \( P_b = \{MH_4, MH_5\} \). \( P_a \) then has three copies, \( \{x_1, x_2, x_3\} \), and \( P_b \) has two copies, \( \{x_4, x_5\} \). \( P_a \) could form write quorum because \( V_{W_a} \) which is 3, is greater than the value of WT, but \( P_b \) fails to form write quorum because \( V_{W_b} \) which
is 2, is less than the value of WT. Subsequently, all transactions in $P_b$ are denied to get access to $x$. While whatever transaction in $P_a$ is allowed to access $x$, in $P_b$ transactions merely experience starvation.

In the presence of this sort of access starvation, the transactions in the non-quorum partition which have been submitted to dispatch replicas could be either blocked or aborted at the time for them to get privilege to dispatch the replicas. Therefore, overall transaction processing performance of MHs in that partition shall degrade substantially due to the blockage or abortion of the transactions. This sort of performance degradation is more severe in case mobile computers are employed in advanced database applications such as multiuser software development environment (SDE). Since database accesses in SDE could last from minutes to months in large-scale software engineering projects, the long-duration nature of transactions is more probable in SDE than in the other distributed environments. Therefore, rollback of the long-duration transactions, either for concurrency control or failure recovery, is generally unacceptable due to the costs of lost work. In case network partitioning unfortunately occurs in the course of such a long-term project, the blockage or abortion of the transactions could lead to more substantial performance degradation.
Fault-Tolerant Quorum Consensus Scheme: \textit{FTQC}

Assumptions

In this section, we define the terminologies and establish assumptions used in this paper. The fact that wireless cells have a limited coverage area means that a mobile host will certainly be crossing through several wireless cells when it moves. When a mobile host discovers that it has moved to a new cell, a \textit{handover} process is triggered. The handover is the process of passing the responsibility of communication connectivity from a previous mobile support station to a new mobile support station. It usually takes a few seconds to transfer the state information pertaining to the mobile host. To accommodate a smooth and continuous handover, cells are usually designed to be overlapped in a way that the handover process can be completed within the time while a mobile host is crossing through the overlapped area [Pitoura and Bhargava, 1994].

Consider that an overlapped area between \textit{Cell}_1 and \textit{Cell}_2, denoted by \textit{OA}_{12}, is required for a smooth handover between the two cells(Fig. 3). While a mobile host, say \textit{MH}_1, is moving into \textit{OA}_{12}, \textit{MH}_1 can temporally detect signals from both \textit{Cell}_1 and \textit{Cell}_2. This means that \textit{MH}_1 can be used as a temporal communication link between \textit{Cell}_1 and \textit{Cell}_2, since \textit{MH}_1 can forward messages from one cell to the other cell. Let us call this link a \textit{mobile link}. In Fig. 3, a mobile link in \textit{OA}_{12}, denoted by \textit{ML}_{12}, is being activated while \textit{MH}_1 is moving into \textit{OA}_{12}. In a normal case, there is no need to use the mobile link because \textit{Cell}_1 and \textit{Cell}_2 can communicate reliably via the wired links, \textit{L}_1 and \textit{L}_2. In case either \textit{L}_1 or \textit{L}_2 unfortunately fails, however, we can employ the mobile link, \textit{ML}_{12}, for emergency communication link.
Fault-Tolerant Replication Control

In a distributed database system, site failures or partition failures could unfortunately prevent distributed applications from accessing the global replicated database. All the conventional replication control schemes in the literature, lead to access starvation in the non-quorum partition which degrades the transaction performance substantially, as illustrated in the previous example. In order to overcome the degradation of the transaction performance, therefore, giving opportunity of quorum formation even to the non-quorum partition as well could help avoid or minimize access starvation in the non-quorum partition. The opportunity of quorum formation can be handled by circulating a single global token amongst partitions. Only a single particular partition possessing the global token has the exclusive privilege of quorum formation. When the global token is passed from one partition to the other partition, the global

Fig. 3 An Activated Mobile Link
token includes the latest versions of replicas in the current partition and moves along with the quorum formation privilege in order to ensure 1SR.

The overhead associated with maintaining the global token can be classified as replica-status storing overhead and global-token passing overhead. The replica-status storing overhead is to store the latest version of replica in two additional hosts, that is, the MSS in the quorum partition and the MSS in the non-quorum partition. Each MSS maintains the latest version of replica in order to send this version to the MSS in the other partition when the global-token passing process is triggered. The space requirement for storing replica status is just a few bytes and this storing activity could occur only in the partition possessing the global token, not in both the quorum partition and the non-quorum partition. Therefore, the replica-status storing overhead is considered to be tolerable. The global-token passing overhead is to copy the latest version of replica stored in the MSS in the quorum partition to the MSS in the non-quorum partition in order to give an opportunity of quorum formation to the non-quorum partition and to avoid access starvation in the non-quorum partition. The amount of data transferred is just also a few packets required to send and to receive the latest version of replica. The overhead of maintaining the global token could occur only when the network is partitioned, not always. Therefore, the overall overhead is also considered to be tolerable.

In this paper, to minimize access starvation at the expense of maintaining the global token, we propose the notion of fault-tolerant quorum-consensus (FTQC). FTQC is based on the philosophy that the quorum formation is allowed even in the non-quorum partition by passing the global token via the mobile link. This process of passing the global token from the quorum partition to the non-quorum partition is triggered only in case the non-quorum partition is suffering from access starvation. The quorum-formation activity for accessing the replica in the
quorum partition, in this case, should be suspended while the non-quorum partition is possessing the global token and is accessing the replica, since only a single particular partition is allowed to access the replica to guarantee 1SR.

The replication control model of FTQC is comprised of two distinct components: a global-token coordinator (GTC) and mobile host transaction managers (MHTMs). GTC allows MHTMs, that is, the transaction managers of given mobile hosts to access global replicated databases without incurring access starvation by circulating the global-token. When a mobile host, say $MH_i$, happens to enter an overlapped area between the quorum partition and the non-quorum partition, $MH_i$ is temporally allowed to communicate with both partitions at the same time because the wireless device of $MH_i$ is essentially able to receive signals from both the MSSs belonging to either partition. $MH_i$ which entered the overlapped area becomes GTC and is ready to perform the global-token passing procedure in case access starvation in the non-quorum partition happens to continue. GTC decides that the non-quorum partition is suffering from access starvation if the length of the waiting-transaction list in the non-quorum partition is much longer than that in the quorum partition.

The main role of GTC is scheduling user transactions issued by mobile hosts without incurring access starvation in both partitions. GTC does this by controlling the order of reads and writes according to the degree of access starvation. This order is controlled in a way that GTC forces MHTMs to accept or delay the operations on basis of the global-token passing scheme. The degree of access starvation can be determined by the GTC by periodically inspecting the number of transactions delayed by MHTMs. If the number of delayed transactions in the non-quorum partition is greater than that in the quorum partition, MHTMs in the non-quorum partition are determined to experience a more degree of access starvation than other
MHTMs in the quorum partition. In this case, GTC forces MHTMs in the non-quorum partition to process the delayed transactions.

The global-token passing procedure essentially performs actions of latest-version copy, status propagation, and transaction activation. In the latest-version copy phase, GTC attempts to copy the latest version of replica in the quorum partition to the MSS in the non-quorum partition. If this first phase is successfully finished, MHTMs in the non-quorum partition are eventually ready to access the latest version of replica stored in the MSS. In the status propagation phase, GTC assembles a notification message regarding the successful migration of the global token from the quorum partition to the non-quorum partition, and then attempts to send this notification message to MHTMs in both partitions. If this second phase is successfully performed, MHTMs in both partitions are able to determine whether their partitions possess the global token by inspecting the notification message. The purpose of performing the transaction activation phase is to wake up all the delayed transactions in the non-quorum partition. This is easily achieved by GTC in a way that GTC just requests the MSS in the non-quorum partition to wake up them. The procedure for global-token coordination for FTQC can be shown in a pseudo-code form as Algorithm 1.

---

**Procedure GTC**

**Input:** current global-token status, waiting-transaction lists;

**Output:** updated global-token status, activated transactions;

**Begin**

```
do {
  L_q = Length(waiting_transaction_list in P_q);
  L_nq = Length(waiting_transaction_list in P_nq);
  if ( L_q < L_nq )
    /* Check whether P_nq has more delayed transactions than P_q */
    /* Perform global-token passing process */
    Copy the latest versions of the replica in P_q to P_nq;
    Propagate(has_global_token = false, MSS_q);
    Propagate(has_global_token = true, MSS_nq);
  
  /* Propagate status */
  Propagate(has_global_token = true, MSS_nq);
  
  /* Propagate transaction activation */
  Propagate(activated_transactions = true, MSS_nq);
}
```
for all $x \in WTL_{nq}$ do
    Activate_Transaction($x$);
end_for;
end_if;

} While (until the network partition failure is recovered);
End

Legends-
$MSS_q$: mobile support station in the quorum partition;
$MSS_{nq}$: mobile support station in the non-quorum partition;
$P_q$: quorum partition; $P_{nq}$: non-quorum partition;
$WTL_{nq}$: waiting-transaction list in the non-quorum partition;
Length($l$): return the length of list $l$;
Propagate($s$, $m$): propagate updated status $s$ to the mobile support station $m$;
Activate_Transaction($t$); activate the delayed transaction $t$;

---

**Algorithm 1** Coordinating Global-Token Passing

The main role of each MHTM is reflecting reads and writes on replicated data to the corresponding MHTMs which maintain the copies of the data. MHTM does this by performing actions of privilege checking, quorum formation, and commit decision. Suppose that a mobile host transaction manager, say $MHTM_i$, received a user transaction, say $T_i$, on replicated data. In the privilege checking phase, $MHTM_i$ checks its ownership of global token by inspecting the value of $has\_global\_token$ which was already propagated by GTC during the global-token passing process. Fortunately, if the value of $has\_global\_token$ is true, $MHTM_i$ is allowed to perform next phase. Otherwise, $MHTM_i$ pushes the unique transaction identifier of $T_i$, say $tr\_id(T_i)$, into the waiting-transaction list, and then aborts $T_i$.

In the quorum formation phase, $MHTM_i$ attempts to form a quorum for $T_i$. Only in case $MHTM_i$ fortunately collects a majority of votes from the other MHTMs, $MHTM_i$ is determined to achieve a successful quorum formation and the value of $result\_QF$ subsequently becomes true. Otherwise, the value of $result\_QF$ becomes false. In the commit decision phase, $MHTM_i$ commits or aborts $T_i$ according to the value of $result\_QF$. If value of $result\_QF$ is set to true, $MHTM_i$ is allowed to read or write the data associated with $T_i$ and commits $T_i$. Otherwise,
MHTMᵢ pushes tr_id(Tᵢ) into the waiting-transaction list, and then aborts Tᵢ. The transaction handling procedure for MHTMs in FTQC can be shown in a pseudo-code form as Algorithm 2.

---

**Procedure MHTM**

**Input:** a user transaction t on replicated data;

**Output:** commit message or abort message;

**Begin**

```
do { switch (has_global_token) {
  case true:
    result_QF = Quorum_Formation(t);
    /* attempt to form a quorum for transaction t */
    if (result_QF = true)
      /* Quorum formation was accepted. */
      t' = Operation(t); /* Read or write on replicated data */
      Propagate(t', P_q);
      Commit;
    else /* Quorum formation was rejected. */
      Push_Transaction_Id(t);
      Abort;
  end_if;
  case false:
    Push_Transaction_Id(t);
    Abort;
  end_case;
} While (until the network partition failure is recovered);
**End**
```

**Legends**

Quorum_Formation(i): execute quorum formation process for transaction i;
Operation(x): read or write of transaction x on replicated data;
P_q: quorum partition;
Propagate(s, m): propagate updated status s to the mobile support station m;
Push_Transaction_Id(t): push transaction-id of transaction t into the end of the waiting-transaction list;

---

**Algorithm 2 Handling Mobile Host Transaction**

The correctness criteria for the replication control in FTQC is based on the one-copy serializability which has been commonly accepted for replicated database control [Agrawal and Abbadi, 1990; Agrawal and Abbadi, 1992; Kumar, 1991]. In order to show that FTQC is correct
with respect to the one-copy serializability, we first need to show that two mobile hosts are never allowed to hold overlapping quorums at the same time.

**Lemma 1 (No Overlapping Quorums):** Suppose all mobile hosts obey the FTQC protocol for a given replicated database $D$. If a mobile host forms a quorum on $D$, then no other mobile host holds a conflicting quorum on $D$.

**Proof:** Suppose that a mobile host, say $MH_i$, forms a quorum on $D$, say $Q_i$, and another mobile host, say $MH_j$, forms a conflicting quorum on $D$, say $Q_j$, in the presence of network partition failure. Suppose also that the network partition failure has partitioned $D$ into the quorum partition, say $P_q$, and the non-quorum partition, say $P_{nq}$. There are then four cases that two mobile hosts can induce overlapping quorums.

\[
\begin{align*}
Q_i & \quad \text{quorum formation in } P_{nq} \\
Q_j & \quad \text{quorum formation in } P_{nq} \\
Q_i & \quad \text{quorum formation in } P_q \\
Q_j & \quad \text{quorum formation in } P_{nq} \\
Q_i & \quad \text{quorum formation in } P_{nq} \\
Q_j & \quad \text{quorum formation in } P_q
\end{align*}
\]

For Case 1: By the rule of the privilege checking in MHTM algorithm, the quorum formation activity is allowed only in case a given mobile host possesses the global token. Since both $Q_i$ and $Q_j$ belong to $P_{nq}$ which does not possess the global token, they are not allowed to form any read(or write) quorums. Therefore, Case 1 is impossible.
**For Case 2**: By the rule of the privilege checking, $Q_i$ is allowed to form any quorum because $Q_i$ possesses the global token, whereas $Q_j$ is prevented. Accordingly, both $Q_i$ and $Q_j$ are never allowed to form any quorums at the same time. Therefore, Case 2 is impossible.

**For Case 3**: Similarly in Case 2, both $Q_i$ and $Q_j$ are never allowed to form any quorums at the same time. Therefore, Case 3 is impossible.

**For Case 4**: By the rule of the privilege checking, both $Q_i$ and $Q_j$ are allowed to form any quorums because both of them possess the global token. Even though both of them are ready to form any quorums, their actual quorum operation is mutually exclusive by the rule of the *quorum intersection property* [Thomas, 1979] which is a necessary condition for QC. Accordingly, only a single quorum of the two, $Q_i$ and $Q_j$, is allowed to perform a quorum read/write at the same time. Therefore, Case 4 is impossible. ■

The one-copy serializability is considered to be insufficient for handling the activities of the quorum operations in the presence of network partitioning in that it is limited to handling the activities only in the quorum partition, not in both partitions. Therefore, we need to define a new notion of serializability to handle the activities in both partitions. For this, we start with a precedence definition of quorum operations.

**Definition 1 (Precedence of Quorum Operations)**: Given two quorum operations, $qo_i$ and $qo_j$, which are generated in partition $i$ and $j$, respectively, $qo_i$ precedes $qo_j$ if and only if:

1. if $i = j$, $qo_i$ was generated before the generation of $qo_j$, or
2. if $i \neq j$, the updated version of $qo_i$ was reflected to all the corresponding mobile hosts which maintain the same data in partition $j$ before the generation of $qo_j$. ■
From Definition 1, we now define the notion of inter-partition one-copy serializability as a correctness criteria for ensuring 1SR in both the two partitions without incurring access starvation in the presence of network partitioning.

**Definition 2 (Inter-Partition One-Copy Serializability):** A replication control algorithm guarantees inter-partition one-copy serializability (IP1SR) if there is a total ordering among conflicting quorum operations. In other words, a sequence of conflicting quorum operations $qo_1, qo_2, ..., qo_n$ is correct if, for every $i = 1$ to $n - 1$, $qo_i$ precedes $qo_{i+1}$.

We now describe a theorem that proves the correctness of FTQC. Theorem 1 shows that FTQC protocol is a sufficient condition for the inter-partition one-copy serializability.

**Theorem 1 (Correctness of FTQC):** A replication control algorithm is inter-partition one-copy serializable if it obeys the FTQC protocol.

**Proof:** Suppose that all mobile hosts obey the FTQC protocol for a given replicated database $D$. Suppose also two conflicting quorum operations, $qo_i$ and $qo_j$, that read or write on $D$. Then either $qo_i$ precedes $qo_j$ or $qo_j$ precedes $qo_i$ because two mobile hosts are never allowed to generate overlapping quorums at the same time, as proved in Lemma 1. Accordingly, there exists a total ordering among conflicting quorum operations. Therefore, FTQC protocol guarantees the inter-partition one-copy serializability.
Performance Evaluation

We evaluated the performance of FTQC by means of simulation. Our experiments are conducted by simulating two different RCSs: one with FTQC and the other one with the basic quorum consensus (BQC) which never exploits FTQC. Each scheme has two variant polices, called abort and retry, abbreviated as A and R, according to the action taken when a given transaction failed to access replicated data. If an RCS employs abort-oriented policy, it aborts the transaction after waiting for abort_time_out. Otherwise, it continues to restart the failed transaction. The simulation experiments are made with CSIM[13]. We now describe the simulation model and parameters used for the experiments. Then, we interpret the results from the simulation.

Simulation Model and Parameters Setting

The simulation model, shown in Fig. 4, is basically a closed queuing model which consists of the several components: transaction generator (TG), data manager (DM), mobile host transaction manager (MHTM), and global-token coordinator (GTC). TG has a role to generate transactions one by one. Before generating a new transaction, it waits for the inter_arrival_delay time which is assumed to be exponentially distributed. Similarly, before being activated, GTC waits for the gtc_delay time which is also assumed to be exponentially distributed. When GTC becomes activated, GTC performs the global-token passing procedure according to the access starvation information from MHTMs. MHTM manages every transaction from its beginning to termination. For each transaction, MHTM forwards data read/write request to the local DM by placing them in the I/O queue and then DM executes the operation in-place. DM consists of a variable number of disks, num_disks. In our simulation model, MHTM and GTC utilize CPU
servers, and DM utilize disk servers. Note that the request to CPU and disk resources are serviced in FCFS (first-come first-served) manner.

Table 1 lists the parameters used in the simulation model and their descriptions. The values chosen to conduct our experiments are also listed in Table 1. The number of mobile hosts in the overall network, $mh_{size}$, is set to 10,000. This value allows the necessary amount of replica access operations to be occurred so that we could observe the practical performance behavior. In our experiments, we varied the network partitioning duration time, $partitioning_{time}$, from 1 minute to 10 minutes in step of 1. The percentage of network partitioning, $partitioning_{ratio}$, is fixed at 10 percent. The quorum formation delay, $qf_{delay}$, is set to 5 seconds and represents the average time for collecting affirmative votes from the other mobile hosts via wireless network. The CPU access delay, $cpu_{delay}$, is set to 12 milliseconds, the I/O access delay, $io_{delay}$, is set to 35 milliseconds. In addition, our simulation experiments are performed under limited resource environments in which one CPU resource and two disk resources are utilized in each mobile host.

**Fig. 4** Simulation Model
Table 1 Simulation Model Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mh_size</td>
<td>Number of mobile hosts in the network</td>
<td>10,000</td>
</tr>
<tr>
<td>num_cpus</td>
<td>Number of CPUs in each mobile host</td>
<td>1</td>
</tr>
<tr>
<td>num_disks</td>
<td>Number of disks in each mobile host</td>
<td>2</td>
</tr>
<tr>
<td>io_delay</td>
<td>I/O time for accessing an object</td>
<td>35 msecs</td>
</tr>
<tr>
<td>cpu_delay</td>
<td>CPU time for accessing an object</td>
<td>12 msecs</td>
</tr>
<tr>
<td>qf_delay</td>
<td>Quorum formation delay time</td>
<td>5 secs</td>
</tr>
<tr>
<td>inter_arrival_delay</td>
<td>Transaction inter-arrival delay time</td>
<td>0.36 secs</td>
</tr>
<tr>
<td>gtc_delay</td>
<td>GTC activation delay time</td>
<td>10 secs</td>
</tr>
<tr>
<td>partitioning_time</td>
<td>Network partitioning duration time</td>
<td>1 - 10 mins in step of 1</td>
</tr>
<tr>
<td>partitioning_ratio</td>
<td>Percentage of network partitioning</td>
<td>10 percent</td>
</tr>
<tr>
<td>abort_time_out</td>
<td>Time-out value for transaction abort</td>
<td>60 secs</td>
</tr>
</tbody>
</table>

Results and Analysis

We now present and analyze the results from the simulation conducted for the two RCSs, FTQC and BQC. The abort-oriented policy and retry-oriented policy were applied to each RCS: FTQC-A, FTQC-R, BQC-A, and BQC-R. The graphs in Fig. 5 and Fig. 6 show the average performance results with 10,000 mobile hosts in the presence of network partitioning. In the experiment, we evaluated the throughput characteristics of the two RCSs with regard to the effect of network partitioning duration. According to the experiment results, the throughput of BQC protocols decrease as the partitioning duration is increased, while the throughput of FTQC protocols is sustained continuously. Fig. 5 shows that the best throughput is obtained with FTQC variants, whereas the worst one is obtained with BQC variants. This is because GTC in FTQC successfully activated the long-delayed transactions in the non-quorum partition, suffered from access starvation, by performing transaction-activation phase of global-token passing procedure.
Note that the considerable performance difference is shown between the abort-oriented policy and retry-oriented policy for BQC. The former indicates about two times better performance than the latter. This is because that BQC with retry-oriented policy continues to retry the failed transaction uselessly in the non-quorum partition, while BQC with abort-oriented policy aborts the failed transaction and processes the next transactions. However, these two kinds of policies are incomparable in the sense that they take quite different approaches on handling the event of transaction failure. Note that the amount of lost work is considerably different according to the approach that is actually taken. On the other hand, the performance difference between the two policies for FTQC is considered to be insignificant. This is because that GTC activated most of the waiting transactions in the non-quorum partition before \textit{abort\_time\_out}. In addition, as shown in Fig. 6, FTQC variants have about two times faster mean response time than those of BQC variants over most of the partitioning duration. Therefore, based on throughput perspective and response time perspective, we finally claim that FTQC outperforms BQC.

\begin{figure}[h]
\begin{minipage}[b]{0.5\textwidth}
\centering
\includegraphics[width=\textwidth]{throughput.png}
\caption{Throughput}
\end{minipage}\hfill
\begin{minipage}[b]{0.5\textwidth}
\centering
\includegraphics[width=\textwidth]{response_time.png}
\caption{Response Time}
\end{minipage}
\end{figure}
Conclusion

We have proposed a new replication control scheme, named FTQC, in order to handle multiple-copy consistency in mobile distributed database systems. The major contribution of FTQC is allowing mobile users to access global replicated databases without incurring access starvation even in the presence of network partitioning. FTQC is based on the idea that quorum formation is allowed even in the non-quorum partition by circulating a single global token via mobile link. FTQC guarantees a new form of one-copy serializability, IP1SR, without sacrificing data availability at the expense of maintaining the global token. The performance evaluation showed that FTQC provides better performance and higher availability than BQC does. Based on these results, we conclude that the protocols which exploit FTQC scheme outperform the protocols which never exploit FTQC. Since FTQC has a generic functionality of fault-tolerant replication management in mobile distributed environments, it can be widely employed in modern distributed groupwork environments.

References


**Short biographies of authors:**

Siwoo Byun received his B. S. degree in Computer Science from Yonsei University in 1989 and his M. S. degree in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) in 1991. Currently, he is working towards his Ph.D. at the KAIST in Seoul. His research interests include distributed systems, mobile computing, and fault-tolerant systems.

Songchun Moon received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 1985. He has been working for KAIST since then. He was a distinguished scholar at the Hungarian Academy of Science and still serves for EUROMICRO as a director. He has developed a multi-user relational database management system IM in 1990 which is the first prototype ever in Korea and also a distributed database management system DIME in 1992, which is another first prototype ever in Korea. Currently, he is involved in developing multi-user object-oriented DBMS OOIM.