SOQ: A Service-Oriented Quorum-Based Protocol for Resilient Real-Time Communication in Partitionable Networks

Bo Zhang and Binoy Ravindran
ECE Dept., Virginia Tech
Blacksburg, VA 24061, USA
{alexzbzb,binoy}@vt.edu

Abstract

We consider efficient real-time communication mechanisms for applications in unreliable and partitionable networks, where network partitions can occur unpredictably and nodes can join and leave arbitrarily. Utilizing quorum systems, we present a quorum-based protocol called SOQ to let nodes update and query service information to a selected set of servers (a quorum). Due to the intersection property of quorums, nodes can obtain latest updated information by simply accessing a quorum. To make the protocol adaptive to network partitions, we propose update/query triggering mechanisms to determine when nodes trigger updates/queries. A quorum access strategy for nodes to judiciously select a quorum to access is designed so that the probability that a query returns the latest service information is maximized. We give in-depth analysis of the protocol, including the communication overhead, load and availability relationship of quorum systems and timeliness analysis of distributed applications. Our experimental studies show that SOQ is resilient to network partitions without incurring large communication overhead. By carefully choosing the size of the quorum system, the distributed application can be executed more quickly and the load of quorum system can be reduced.

I. Introduction

Designing efficient real-time communication mechanisms in unreliable networks, e.g. mobile ad-hoc networks (MANETs) is very challenging. Due to the mobility and dynamic characteristics of such networks, existing real-time communication solutions under fixed infrastructure-based networks cannot provide satisfying performance. The desired mechanism needs to provide end-to-end timing assurance for distributed real-time applications in such networks.

One of the unique challenges in these networks is partitioning, which is the breakdown of a connected network topology into two or more separate, unconnected topologies. In fixed infrastructure-based networks, the occurrence of network partition is highly unlikely and only possible if big parts of the infrastructure fail simultaneously.

One of the application of distributed real-time applications in MANETs is distributed mobile games. [20] Such games are often played by people at arbitrary times (e.g., while waiting for a bus or at a subway station) and by players who form groups dynamically and in an ad hoc manner. At any time, a player may invoke a real-time application on his or her node (e.g., a laptop; a PDA) and call for a service that may reside on another player's (and often multiple players') node. For example, during an online mobile chess game, a player may invoke a request to start a new game with another player in the network. Any player which is not engaged in a game at the same time can provide this service. In this scenario, often significant number of players may move, and thereby potentially partition the network. Thus, a robust real-time communication mechanism would be beneficial which allows two or more separate games to be established without having to restart a game.

Our design of real-time communication protocol for distributed applications is based on the concept of a quorum system, which is a family of subsets that always intersect. For example, consider a network of N nodes. We select n nodes ($n \ll N$) from the network as servers. The remaining nodes are ordinary nodes or clients. A quorum system is formed by the selected servers. Each update by a client is sent to a subset of servers (a quorum). Similarly, each query is also sent to a quorum. In the context of the mobile chess game example, when a client has just finished a game, it sends an update to a quorum: "I'm available to take part in a new chess game". The corresponding servers will keep the record until the next update. Thus, when a client invokes a request to start a new game with another player, it will send the query to a quorum to discover the available service provider. Due to the intersection property of quorum systems, there is high probability that at least one server will keep the latest information of the previous node. Thus, the appropriate service provider can be discovered. The challenge, however, is to design efficient update/query mechanism as well as quorum access strategy so that the probability of a *hit* (a query returning the latest record of a node) is maximized even when the network is partitioned.

SOQ performs efficiently both when the network is connected and when it is partitioned. Successive updates/queries by a node need not be sent to the same subset of servers. This strategy can efficiently balance the load on the servers. On the other hand, because the set of reachable servers changes with time, a node has to determine the reachable subset of servers before each update/query based on its recent information about server reachability.

We make the following major contributions in this paper:

(1) We design a service-oriented quorum-based protocol: SOQ for resilient real-time communication of distributed applications in partitionable network;

(2) We design an efficient update/query triggering policy and quorum access strategy for SOQ to improve the performance of the protocol;

(3) We analyze the protocol from various aspects including its communication overhead, load and availability relationship of quorum systems and timeliness analysis for distributed applications. The bound of the load of quorum incurred by SOQ is determined by partial quorum availability. The expected delay for a distributed application is determined by the quorum placement and access strategy; and

(4) We present experimental studies to evaluate the proposed protocol.When the probability of network partition is large enough, the query accuracy and hit probability drops significantly; the load of the quorum system arises dramatically; We can choose the size of quorum systems to improve the query accuracy and hit probability and reduces the load of quorum systems.

The rest of the paper is organized as follows: In Section II, we discuss the related work. In Section III we introduce the quorum system preliminaries. We present the system model in Section IV. The protocol is described in Section V. We analyze the protocol in Section VI. In Section VII we present our experimental studies. The paper concludes in Section VIII.

II. Related Work

Quorum system is a basic tool for reliable agreement in distributed systems [16], [18]. Researchers have also designed quorum systems in a dynamic environment, e.g. [1], [6], [11]. To apply quorum systems in partitioned networks, Herlihy [9] presented dynamic quorum adjustment method for partitioned data. This method permitted an object's quorum to be adjusted dynamically in response to failures and recoveries. A transaction that is unable to progress using one set of quorums may switch to another, more favorable set, and transactions in different partitions may progress using different sets. Karumanchi et al. [13] proposed strategies that use local knowledge about the reachablility to judiciously select quorums in partitionable mobile ad hoc networks. They designed an update/query protocol to let nodes update their locations when needed. Epidemic quorums [3], [10] have also been applied for managing replicated data, which enables highly available agreement even when a quorum is not simultaneously connected. However, Barreto et al. [3] showed that Epidemic quorum systems may not always be advantageous over classical quorum systems in partitioned networks.

Quorum system is widely used in ad hoc networks. One of the most popular applications is implementing location service. In the work of Haas and Liang [6], a uniform random quorum system is used for mobility management. Nodes form a virtual backbone. When a node moves, it updates its location with one quorum containing the nearest backbone node. Each source node then queries the quorum containing its nearest backbone for the location of the destination. Luo et al. [15] present a Probabilistic quorum system for ad hoc networks (Pan), a collection of protocols for the reliable storage of data in mobile ad hoc networks. A gossip-based protocol is designed for quorum access and an asymmetric quorum construction is applied. These work has noticed the highly dynamic and unpredictable topology changes in ad hoc networks. Our work differs from them for that we focus on quorum-based protocol in network partitions.

To the best of our knowledge, our work is the first one to apply quorum systems for distributed real-time applications. Han et al. [7] [8] present gossip-baseds protocol to counter network failures and message losses. In this work, the possibility of network partitions is “coarsely” handled by lumping the probability of message losses (due to partitions, among other things) in one of the input variables of their analysis. Our work is based on our past work [21], where we designed a quorum-based gossip protocol based on [7]. In [21], we applied quorums to limit the range of each gossip round and thereby reduce the message overhead, resulting in improved timeliness behavior of distributed tasks. Based on this work, in this paper, we design a quorum based update/query protocol which is adaptive to network partitions and do not need to gossip the query in the network. In this way, the message overhead is highly reduced and application timeliness behavior is further improved.

III. Quorum Preliminaries

Given a universe U of elements, a quorum system $\mathcal{Q} = \{Q_1, ..., Q_m\}$ on U is a family of subsets of U such that any two quorums Q_i and Q_j have a non-empty intersection [2]. Quorum systems are widely used in distributed systems for achieving mutual exclusion, consistent data replication, and dissemination of information. In typical quorum-based algorithms, each client accesses the system by accessing all the elements in some quorum Q_i belonging to $\mathcal{Q}$. The intersection property ensures that any Q_i would suffice to operate on behalf of the system. Usually, the client chooses Q_i from a probability distribution $p : \mathcal{Q} \rightarrow [0, 1]$ over $\mathcal{Q}$; this p is called the *access strategy* for the quorum system. The access strategy is typically chosen based on two concerns: the *load* and *availability* of the quorum system.

Definition 1: Given a quorum system $\mathcal{Q}$ on U, the load on $u \in U$ of access strategy p is $\Sigma_{Q_i \ni u} p(Q_i)$. The load of the quorum system is the load of the most heavily loaded element

$u \in U$: $L_{\mathcal{Q}} = \max_{u \in U} \Sigma_{Q_i \ni u} p(Q_i)$.

Definition 2: Given a quorum system $\mathcal{Q}$ on U and a client v_i, the availability of the quorum system is defined as the ratio of the number of quorums reachable from v_i to the total number of quorums.

Since the accesses of one quorum by clients (which are themselves nodes in the network) have to be implemented by messages sent along the network, the performance of quorum-based systems now crucially depends on the delays introduced by these accesses [5]. In fact, we would like the logical quorums $Q_i \in \mathcal{Q}$ to be mapped to closely clustered physical nodes in the network so that we do not incur large delays in trying to reach far-flung parts of the network.

We introduce the concept of *Grid* quorum systems for our discussion, which is defined as follows:

Definition 3: On a universe U of k^2 elements, a Grid quorum system is composed of k^2 elements laid out on a k by k square grid M, and each quorum Q from $\mathcal{Q}$ is formed by taking all the elements from some row and some column of M. Hence each quorum has $2k-1$ elements, and there are k^2 quorums in $\mathcal{Q}$.

The uniform access strategy yields the optimal load for the Grid. For a Grid quorum system of n elements, this strategy will lead to the optimal load of $O(\frac{1}{\sqrt{n}})$. [17]

IV. Models

A. Application Model

We model a distributed application as $T = \{t_1, t_2, ...t_n\}$, where t_i is called an *activity*. An activity is composed of a sequence of *sections*, i.e., $t_i = [s^i_1, s^i_2, ...s^i_m]$. Sections of an activity must be executed sequentially, i.e., $s^i_1 \prec s^i_2 \prec ... \prec s^i_m$. A section constitutes the portion of the service's execution on a node. It is invoked on a *service requestor*, and finished on a *service provider*. An activity's most recent section — the activity's current execution locus — is called the activity's *head*, and the node hosting the head is called the activity's *head node*. The head of a activity is the only section that is active.

When a section is created on its service requestor, it does not know the identity of its service provider. Hence, the service requestor needs to discover the service provider which provides the proper requested service in the network. Such a section involves three steps: (1) service requestor queries; (2) service provider relies; and (3) service executions. The sequence is repeated in a particular order and a set of such sequences running in the network concurrently forms an application.

For example, in our previous mobile chess game scenario, a chess game held between two players is an activity. A set of such concurrent activities, i.e., a chess game tournament, forms a distributed online application.

B. Network Model

We assume a network of N nodes. A set of servers are selected from the network and a quorum system $\mathcal{Q} = \{Q_1, ..., Q_m\}$ is constructed. The sets Q_i are constructed a priori and every node knows the membership of these sets by broadcast the request at the start of the protocol. Given n servers, it is possible to form quorums of size $O(\sqrt{n})$.

A basic unicast routing protocol such as DSR [12] is assumed to be available for packet transmission between nodes. MAC-layer packet scheduling is assumed to be done by a CSMA/CA-like protocol (e.g., IEEE 802.11). We assume that node clocks are synchronized using an algorithm such as [19].

We assume that nodes may fail by crashing, links may fail transiently or permanently, and messages may be lost. Network partitions can occur, which is the breakdown of the connected network topology into two or more separate, unconnected topologies. When the network is partitioned, nodes in different partitions cannot communicate with each other.

V. SOQ: Service-Oriented Quorum-Based Protocol

A. Basic Protocol

The basic protocol of SOQ consists of three parts: service information update, service information query, and section execution. For each section of an activity, the service requestor discovers the appropriate service provider based on service update/query protocol. We design update/query protocol for service information update and query, and section execution protocol for service executed on the service provider.

Service Information Update: When a node v_i wishes to update its service information, it timestamps the datum with its clock value. The format of the datum is $< ID(i), addr(i), service(i)$, $timestamp(i) >$. The ID field is unique for each node, e.g., the MAC address, which is used to identify the node and doesn't change. The *addr* field contains the network address the node designated from the network. When the network partitions or merges with other networks, the network address of the node may change. The *service* field represents the current available service that the node provides. For example, an idle client in the mobile chess game may provide "join a new game" service. The *service* field may change depending on the status of the client.

When an update is triggered on a node, it selects a quorum Q_i from the set of quorums and sends an UPDATE message, timestamped with its local clock value, to all servers in the quorum. The servers, on receiving the UPDATE message, overwrite their old copy of the data item with the new copy. If they do not have an old copy of that data item, they simply add the information received in the message to their database.

Let an UPDATE message be first sent to quorum Q_i. Later, let another UPDATE, for the same data item, be sent to quorum Q_j. Then, all servers in the set $Q_i - Q_j$ have outdated versions of the data item, while all servers in Q_j have the latest version of the data item.

Service Information Query: When a query is triggered on a node, it selects a quorum Q_j and sends a QUERY message to all servers in the quorum. When a server receives a QUERY for a specific service and has a copy of it, the server sends a REPLY containing the information along with the timestamp associated with the datum. Otherwise, the server sends a NULL reply. By receiving all the REPLY messages, the service requestor selects the value of the datum with the greatest timestamp.

As two quorums always intersect in the fixed network, the set of queried servers is bounded to contain at least one server that belonged to the quorum that received the latest update. Hence, each query returns the latest value of the queried data item.

Section Execution: When the current service requestor selects the data, it extracts the *addr* field of the data and sends that node the request to execute the service. The requested node, upon receiving the request, will check whether it can execute the requested service. If so, it executes it and sends back the acknowledgement. If not, it also sends back the acknowledgement that it cannot provide the requested service. The service requestor will trigger the service information query again to discover the service provider. The section is successfully executed until the requested service is executed on the service provider.

B. Update/Query Triggering Mechanism

In unreliable and partitionable networks, we design appropriate update/query triggering mechanisms for SOQ so as to mitigate the impact of network partitions.

Update Triggering: The aim for nodes updating service information is to propagate the information such that other nodes, on querying for information, obtain as recent a version as possible. We propose following three policies to trigger updates:

(1) The service type provided by the node has changed since the last update;

(2) The network address of the node has changed since the last update; and

(3) A certain number of links incident on it have been established or broken since the last update

The first policy is very straight-forward. The service information must be updated whenever it changes. For the second strategy, when the network partitions, the network address of the client may change. Thus, an update is triggered to send the server its new network address so that it can be communicated with other nodes. The last update policy is motivated by the observation that in ad-hoc networks, the frequency of updates should reflect the dynamism exhibited by the network. It is an attempt to capture the relative change in topology.

Query Triggering: The queries will only be sent by service requestors. We propose following three policies to trigger queries:

(1) A section is invoked on a service requestor;

(2) No available service update information received since the last query; and

(3) The selected service provider cannot be reached, or doesn't provide the service any more.

When a section of an activity is invoked, a query is triggered to discover the proper service provider. When there is no service information matched in the servers that it queries, there are two possibilities: (1) There is no node providing the requested service in the service requestor's network partition; or (2) The proper service provider exists, but the queried servers do not have the latest information for some reason. The second phenomenon seems to contradict the quorum intersection property. However, it is quite possible in partitioned networks. For example, let the node update its service to Q_i, and a service requestor sends the query to Q_j. Hence, the updated information will be saved on $Q_i \cap Q_j$. If those servers belong to another network partition, the service requestor will not get the correct information. Thus, the second policy is applied to trigger a query again.

The third policy is applied in the worst case: the selected service provider cannot be reached or does not provide the service. This is because the service provider is selected by stale information. The selected node is either located in another network partition, or cannot provide the requested service. Thus, the service requestor has to trigger a query again.

C. Quorum Access Strategy

From the previous discussion, we observe that in a partitionable network, not all the servers in the selected quorum are reachable from the node that sends the update or query. Thus, it is crucial to design an efficient quorum access strategy to maximize the probability of a *hit* — i.e., a query returning the latest information about a node.

Let node v_i select quorum Q_i to update the service information. If some elements of Q_i are in a different partition from v_i at the time of the update, they will not receive the update. We denote the set of servers which receive the update as $Q_i^{'}$. Subsequently, let another node v_j select a quorum Q_j to query the service information, and we denote the set of servers which receive the query as $Q_j^{'}$. Thus, there is a possibility that the query may not return any information, or return stale information.

To alleviate this problem, we need to select quorums to maximize the *hit* probability. The idea is to select quorums so that the sets $Q_i - Q_i^{'}$ and $Q_j - Q_j^{'}$ are as small as possible. The smaller these sets, the greater the probability of the set of reachable queried servers intersecting with the set that received the latest update.

We use the concept of *disqualified list* from [13] to select servers for updates and queries. Node v_i maintains a disqualified list, DQL_i, containing servers that v_i believes are unreachable. The node selects servers for updates/queries on the basis of DQL_i's composition.

Initially, DQL_i is empty. We propose the following steps for updates and queries based on the disqualified list:

(1) Node v_i first eliminates all quorums that have at least one node in DQL_i;

(2) If there are quorums remaining, one of them is randomly selected and messages are sent to servers in this quorum;

(3) If all quorums have at least one node in DQL_i, then quorums having the smallest number of elements in DQL_i are located. One of them is randomly selected and messages are sent to servers in this quorum;

(4) If at least one server sends an acknowledgement in the case of an update, or a non-NULL time-stamped value in the case of a query, the operation is said to succeed; and

(5) If a reply is not received from a server within $T_{timeout}$, node v_i concludes that this server is not reachable and adds it to the DQL_i. Usually we set $T_{timeout} = \Omega(E[\delta(v_i, v_j)])$, where $\delta(v_i, v_j)$ is the round-trip time between nodes in the network. Once a server has been added to DQL_i, it stays in it for a disqualified duration, δ_{DQL}. At the end of δ_{DQL}, the server is removed from DQL_i. The value of δ_{DQL} can be adjusted to keep balance between updating disqualified list in time and reasonable communication overhead.

VI. Protocol Analysis

A. Communication Overhead

Given n servers, there exists quorum formation schemes that give quorums of size $O(\sqrt{n})$. Even if an update and query requires a constant number of retries before success, the communication complexity is $O(\sqrt{n})$. Note that n is usually much smaller than the total number of nodes N. Thus, we believe that the communication overhead is reasonable.

B. Relationship between Quorum Load and Availability

The load of the servers in the quorum system describes the probability that a server is selected in the given access strategy. The load of the most heavily loaded element evaluates the access strategy. When the quorum system is fully available, i.e., no partition exists in the quorum system, SOQ quorum access strategy transforms to uniform access strategy and the load of all elements is $O(\frac{1}{\sqrt{n}})$. When the quorum system partitions, the load of the servers will increase due to the decreased availability of quorums. The following examples reveal the relationship between load and availability of Grid quorum system operates on SOQ protocol.

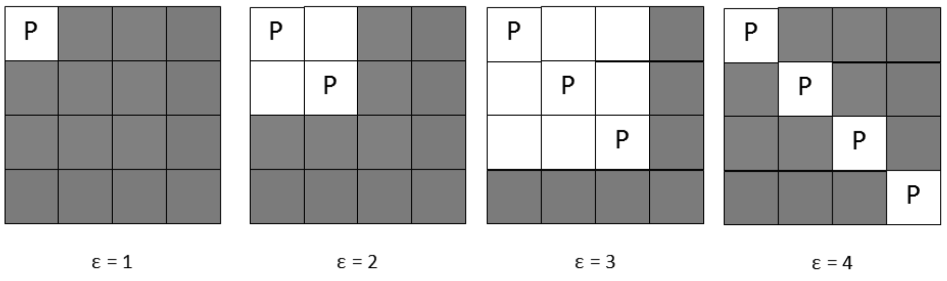

Fig. 1. Server placement in 4 × 4 Grid quorum system with least quorum availability

An example of worst case of quorum availability of 4×4 Grid quorum system is shown in Figure 1. Elements marked by "P" represent servers in other network partitions at the time the node selects quorums to send update/query. Shaded elements represent the set of servers nodes can select to access according to our quorum access strategy. In the quorum system with ε such servers, Figure 1 illustrates the placement of these servers in the Grid quorum system which results in least quorum availability. In this scenario, $\sqrt{n}$ servers in other partitions are adequate to make each quorum contain one element in other network partitions, and nodes in this network partition cannot find a quorum from which it can access all its elements. According to SOQ quorum access strategy, clients will randomly choose a quorum to access. Specifically, for a Grid quorum system of n servers, if we use ε to denote the number of servers in the disqualified list of the service provider, We have following lemmas:

Lemma 1: The number of elements in other network partitions and the size of the quorum system determine the lower bound of the availability of Grid quorum systems.

Proof: In the worst case, when $\varepsilon \leq \sqrt{n}-1$, if we use A_Q to denote the availability of Grid quorum system $\mathcal{Q}$, we have:

$$A_{\mathcal{Q}} = \frac{No.\ of\ available\ quorums}{Total\ No.\ of\ quorums} = \frac{(\sqrt{n}-\varepsilon)^2}{n} \tag{1}$$

When $\varepsilon \geq \sqrt{n}$,

$$A_{\mathcal{Q}} = 0 \tag{2}$$

Hence, in a quorum system where ε servers belong to other network partitions, the lower bound of quorum availability is provided by Equations 1 and 2. ■

Based on the quorum access strategy of SOQ, the load incurred by this server placement is:

$$L_{\mathcal{Q}} = \frac{2(\sqrt{n}-\varepsilon')-1}{(\sqrt{n}-\varepsilon')^2},\ \varepsilon' = \varepsilon \mod \sqrt{n} \tag{3}$$

This server placement, however, does not incur the heaviest load among all possible situations. In fact, when $\varepsilon = i\sqrt{n}, i = 0, 1, 2...$, each quorum under this placement contains i unreachable servers. In this case, the quorum access strategy becomes the uniform access strategy.

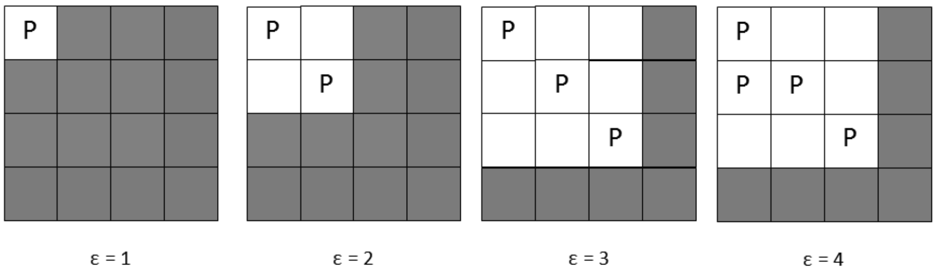

Fig. 2. Server placement in 4 × 4 Grid quorum system with heaviest quorum load

On the other hand, the placement incurring the heaviest load does not necessarily imply the worst quorum availability, as shown in Figure 2. Even if there is always at least 1 quorum available, the load of the quorum will increase to 1 when the number of unreachable servers increases. In this scenario, at least $\sqrt{n}-1$ unreachable servers incur the heaviest load of the quorum system.

Lemma 2: The number of elements in other network partitions and the size of the quorum system determine the upper bound of the load of Grid quorum systems with access strategy of SOQ.

Proof: Based on the quorum access strategy of SOQ, when $\varepsilon \leq \sqrt{n}-1$, we have:

$$L_{\mathcal{Q}} = \frac{2(\sqrt{n}-\varepsilon)-1}{(\sqrt{n}-\varepsilon)^2} \tag{4}$$

When $\varepsilon \geq \sqrt{n}-1$,

$$L_{\mathcal{Q}} = 1. \tag{5}$$

Hence, in a quorum system where ε servers belong to other network partitions, the upper bound of the load of the quorum system is provided by Equations 4 and 5. ■

When $\varepsilon \leq \sqrt{n}-1$, the availability of this placement is the same as that given by Equation 1. When $\varepsilon \geq \sqrt{n}$, we have $A_Q = \frac{1}{n}$. Similar to the previous case, this placement does not incur the worst quorum availability. The placement always yields at least one quorum available, which implies that clients can always find a quorum in which all elements can be accessed.

From these arguments, we note that there exists some relationship between the availability and the load of the quorum system. We can extend the definition of quorum availability to partial quorum availability to deduct this relationship:

Definition 4: Partial Quorum Availability: For a quorum system with quorum size k, partial quorum availability is a vector $\mathbf{A} = \{A_0, A_1, ..., A_{k-1}\}$, where A_i represents the portion of quorums where $k-1$ elements are available.
From this definition, we see that $\sum_{0 \leq i \leq k-1} A_i = 1$.

With this definition, we have the following theorem:

Theorem 3: The lower and upper bounds of the load of the Grid quorum system operating on SOQ is determined by the partial availability and the size of the quorum system.

Proof: According to SOQ quorum access strategy, the set of quorums with the least number of unreachable elements is chosen. For Grid quorum systems, we can immediately derive the relationship between load and partial quorum availability:

$$L_Q = \frac{a_i + a_j - 1}{A_{i^*} n} \tag{6}$$

where

$$i^* = \arg\min_{A_i = 0} i;$$

$$a_i a_j = A_{i^*} n;$$

$$1 \leq a_i, a_j \leq \sqrt{n} - 1$$

Note that a_i and a_j are integer factors of $A_{i^*} n$. Thus, the load of the quorum can have several possibilities based on the knowledge of partial quorum availability. This is because, we only know the number of reachable quorums, but do not know the configuration of these quorums. For example, if $A_{i^*} n = 12$, we have no idea whether these quorums are formed by 3 columns and 4 rows or 2 columns and 6 rows. Generally speaking, different quorum configurations can result in different loads.

Although the load of the quorum cannot be determined through the knowledge of quorum availability, we can lower-bound and upper-bound it based on simple inequalities. Note that a_i and a_j are integers. Thus, we have:

$$\lceil 2\sqrt{a_i a_j} \rceil \leq a_i + a_j \leq a_{i^*} + a_{j^*}$$

where

$$a_{i^*} = \max_{a_i \leq \sqrt{n}-1} a_i$$

Note that $a_i a_j = A_{i^*} n$. Hence, the load of the quorum system with the quorum access strategy is bounded by:

$$\frac{\lceil 2\sqrt{A_{i^*} n} - 1 \rceil}{A_{i^*} n} \leq L_{\mathcal{Q}} \leq \frac{a_{i^*} + a_{j^*} - 1}{A_{i^*} n} \qquad (7)$$

■

C. Timeliness Analysis

The performance of SOQ protocol depends on how fast the application is executed. Hence, the expected time needed for section executions significantly affects the global performance. We have the following theorem:

Theorem 4: The expected delay incurred on each section is bounded by three factors:

(1) The expected delay for a service requestor to access quorums;

(2) The expected number of query attempts for a hit; and

(3) The expected number of queries returning stale information.

Proof: Using D_i to denote the delay incurred on service provider v_i, we have

$$D_i = \sum_{j=1}^{q_i} \delta(v_i, Q_j) + \sum_{k=1}^{h_i} \delta(v_i, v_k) \qquad (8)$$

where $\delta(v_i, Q_j)$ denotes the access delay from node v_i to quorum Q_j. q_i is the number of query attempts for a hit, and h_i is the number of times v_i selecting a service provider.

Nodes access a quorum by reaching all its elements:

$$\delta(v_i, Q_j) = \max_{u \in Q} \delta(v_i, u)$$

Then, the expected delay (under specific quorum access strategy p) for v_i to access $\mathcal{Q}$ is:

$$\Delta(v_i) = \sum_{Q \in \mathcal{Q}} p(Q) \delta(v_i, Q)$$

In the best case, the service requestor gets the latest service information by just sending one query. Thus, the optimal delay of node v_i is:

$$D_i^0 = \delta(v_i, Q_o) + \delta(v_i, v_o) \qquad (9)$$

Therefore, the penalty delay incurred on the service requestor is:

$$D_i^1 = \sum_{j=1}^{q_i-1} \delta(v_i, Q_j) + \sum_{k=1}^{h_i-1} \delta(v_i, v_k)$$

Note that $q_i - 1$ is the number of query attempts before a hit, and $h_i - 1$ is the number of times that a service requestor selects a service provider based on stale information. So we have $q_i \geq h_i$.

By the quorum intersection property and the triangle inequality, we have

$$\delta(v_i, v_k) \leq \Delta(v_i) + \Delta(v_k).$$

Thus, the delay incurred on service provider v_i is bounded by

$$D_i \leq \sum_{j=1}^{q_i} \delta(v_i, Q_j) + h_i\Delta(v_i) + \sum_{k=1}^{h_i} \Delta(v_k).$$

Now, taking expectations over quorum access strategy p, we obtain:

$$E(D_i) \leq E(q_i + h_i)\Delta(v_i) + \sum_{k=1}^{E(h_i)} \Delta(v_k). \tag{10}$$

Hence, $E(D_i)$ is bounded by $\Delta(v_i)$, $E(q_i)$ and $E(h_i)$.

■

$\Delta(v_i)$ is determined by the network topology and quorum system placement. When the servers are selected and the quorum system is determined, it solely depends on the network topology. $E(q_i)$ and $E(h_i)$ depend on the specific quorum access strategy. The SOQ quorum access strategy maximizes the probability of intersection between a query and an update based on the local knowledge of reachability.

From Equation 10, the expected delay for activity t_i is

$$E(D^i) = \sum_{s_j \in t_i} E(D_j).$$

Hence, the expected delay for an application T is

$$E(T) = \max_{t_i \in T} E(D^i).$$

VII. Experimental Studies

In this section, we present results of simulation studies that we conducted to evaluate SOQ. We conducted simulation experiments to evaluate the accuracy of the query, the hit probability, and the load of the quorum system.

We considered a network composed of 200 mobile nodes randomly distributed in a round region with radius of 50 length units. Nodes were allowed to move within this region. Some nodes were selected as servers to form a Grid quorum system. In our experiments, we varied the number of servers to form 4×4, 6×6, and 8×8 Grid quorum systems.

We simulated node mobility by setting the duration for which a node stays at a location to be exponentially distributed with mean 0.5 time units. At the end of this duration, the node randomly selects another location within the round region that is at most 10 length units away from its current location.

A wireless link was assumed to be present between a pair of nodes if they are in the wireless communication range of each other. We varied the different values of wireless range to measure the impact of network partitions.

Distributed activities were created in this network, with a sequence of sections to be executed on network nodes. We varied the number of sections of distributed activities to measure how they affected the performance of SOQ.

Each node in the network is assigned an unique ID which does not change during network partitioning and merging. Mobile nodes are able to self-configure their IP address due to the lack of centralized administration in MANETs and by using IP address autoconfiguration protocol such as [4]. When network partitions or merges, nodes change their IP address when they detect address conflicts, i.e., two different nodes in the same network partition share one IP address.

When a query is triggered on the service requestor, a specific type of service is queried to be executed in the network.

We considered 8 types of service, and each type of service is provided by 4 randomly selected nodes in the network. The duration for which a node provides a service was exponentially distributed with mean 1 time unit. The update and query messages were sent to servers based on quorum access strategy of SOQ. If a node discovers that a server is unreachable, the node places that server in its disqualified list for a period δ_{DQL}. This value can be adjusted to mitigate the tradeoff between the up-to-date maintenance of the disqualified list and reasonable communication overhead. In our experiments, we set $\delta_{DQL} = 0.25$ time units.

The average number of attempts for a successful query with different wireless communication ranges and different Grid quorum system sizes is shown in Figure 3. The data points were calculated from 100 successful queries. A successful query returns a non-NULL data containing the requested service information. When the wireless communication range is equal to 25, 20, and 15 length units, just 1-2 attempts are only needed for a successful query, regardless of the applied Grid quorum system. This is because, for these values of wireless communication range, the probability of network partitioning is relatively low. Even if the network partitions, the

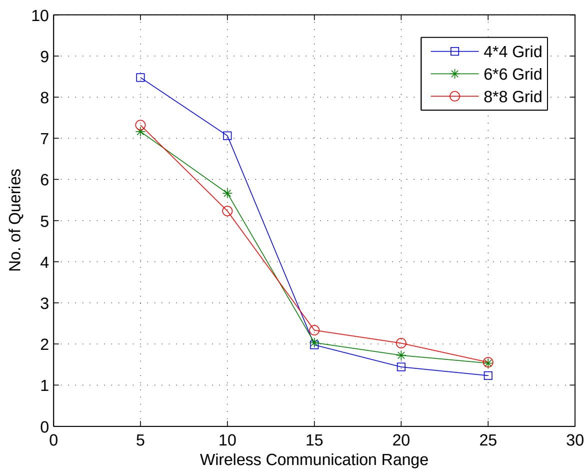

Fig. 3. Average number of attempts for a successful query with different wireless communication range and Grid quorum system

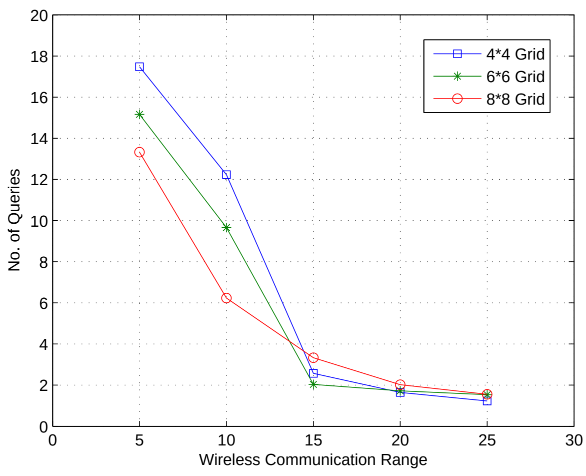

Fig. 4. Average number of attempts for a hit with different wireless communication range and Grid quorum system

partitions may get merged quickly. However, the number of attempts increases dramatically when the wireless communication range equals 10 and 5 time units. For these values, the probability of network partitioning increases significantly and most of the queries return NULL data. For three different sizes of Grid quorum systems, when the wireless communication range is large (e.g., 15, 20, 25), the average number of attempts are almost the same. When the communication range is small (e.g., 5, 10), 4×4 Grid quorum needs more attempts for a successful query because the probability that no server exists in a network partition increases. Thus, the Grid quorum system of smaller size suffers more than quorum systems of larger size.

Under the same condition, Figure 4 shows the average number of query attempts for a hit, calculated from 100 hits. A successful query does not imply a hit due to the existence of stale information. When all queried servers return stale service information or NULL data, the querying node will select the stale information with the highest timestamp. In this case, the query is successful but the selected target is not a proper service provider. In other words, a "successful" query doesn't necessarily return "correct" data. Hence, the average number of attempts for a hit is larger for a successful query, as shown in Figure 4. Similar to Figure 3, 4×4 Grid quorum system suffers the most from increasing probability of network partitioning among the three Grid quorum systems.

Figure 5 illustrates the relationship between the success ratio of distributed activities and different wireless communication ranges. The success ratio of an activity is the probability that an activity is successfully executed in a designated time period d. In our experiments we set

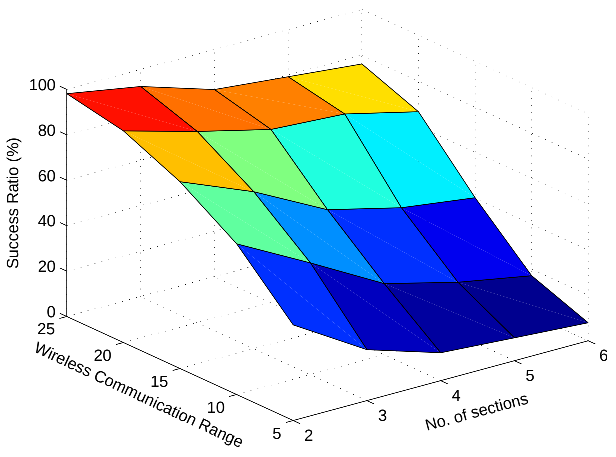

Fig. 5. Success ratio of distributed tasks with different wireless communication range and number of sections

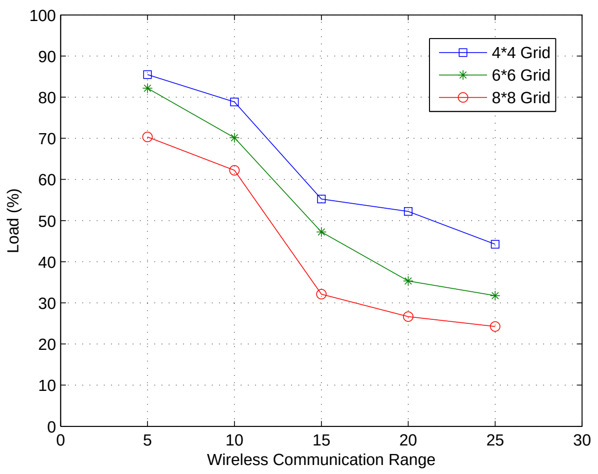

Fig. 6. Load of the Grid quorum system

$d = 100$ time units. Distributed activities are composed of different number of sections. In our experiments, we set the number of sections from 2 to 6, under 5 different values of wireless communication range. Success ratio is calculated from 100 tasks with the same parameters. The figure shows that the task success ratio decreases when the wireless communication range decreases and the number of sections increases. When network partitioning occurs frequently, e.g., when the wireless communication range equals 5 or 10 length units, the success ratio drops dramatically. Generally, the success ratio suffers more from decreasing wireless communication range than from increasing number of sections.

We measured the load of different sizes of Grid quorum systems. Figure 6 shows this simulation result. The data point is the mean value of 100 queries. The figure illustrates that a quorum system of larger size implies a smaller load, which verifies the analytical result. As we expected, the load of the server increases significantly when the probability of network partitioning increases. This is because, when network partitioning occurs, less number of servers are available in one network partition, which increases the probability that a server is selected to access.

We measured the distribution of server load for different wireless communication ranges, as shown in Figure 7 and Figure 8. The load of each server (identified by its server number) is measured as the total query message it received in 100 queries. When the wireless communication range is relatively high (e.g., 20 length units), the distribution is almost the uniform distribution and all servers share the same load. When the value is low (e.g., 10 time units) and the network partitions, different servers receive different number of messages and the load is no longer uniformly distributed across all servers.

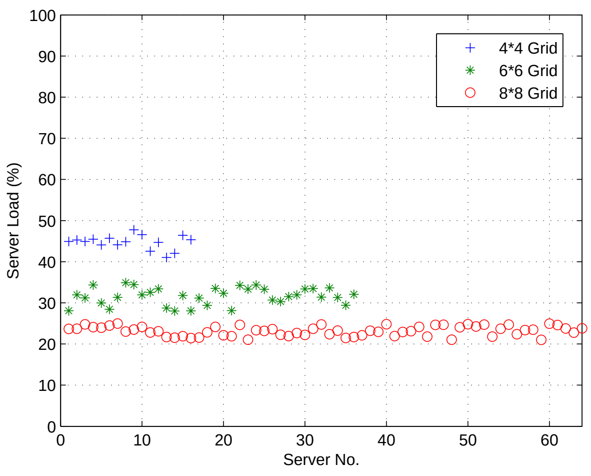

Fig. 7. Distribution of server load when wireless communication range = 20 length units

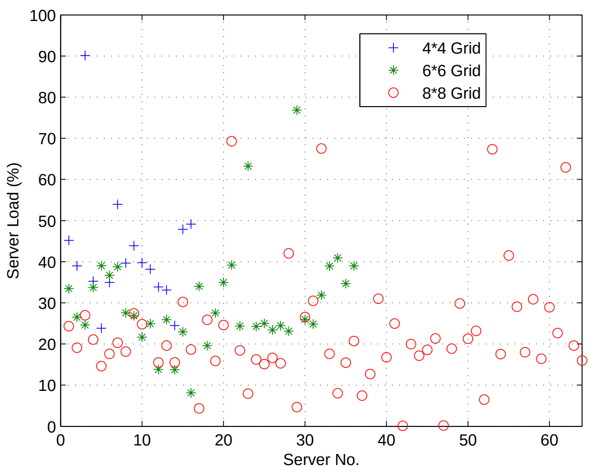

Fig. 8. Distribution of server load when wireless communication range = 10 length units

VIII. Conclusions and Future work

In this paper, we designed SOQ protocol for real-time communication of distributed applications in unreliable and partitionable networks. By employing quorum systems, a subset of the servers suffices to operate on behalf of the quorum system. In this way, the message complexity for communication is highly reduced. To make the protocol adaptive to dynamic network environments, we proposed a set of mechanisms to determine the time when nodes trigger updates and queries. In partitioned networks, it is important to maximize the accuracy that a query returns the latest information about a node. Based on this concern, we proposed a quorum access strategy, which makes the best effort to maximize the hit probability based on the node's knowledge of reachablity. We analyzed SOQ from three aspects: message overhead, load, and availability relationship of quorum system and task timeliness analysis. From theoretical analysis and experimental results, we concluded that:

(1) SOQ is resilient for real-time communications in partitionable network without incurring large communication overhead.

(2) The load of quorum systems incurred by SOQ is determined by the availability of the system;

(3) The delay needed for an application execution incurred by SOQ is determined by the quorum system placement and access strategy;

(4) When the probability of network partition is large enough, the query accuracy and hit probability drops significantly; the load of the quorum system arises dramatically; and

(5) The larger size of quorum systems improves the query accuracy and hit probability when network partitions and reduces the load of quorum systems. On the other hand, it will also increase the communication overhead.

Our work can be extended immediately by relaxing the assumption on the (known) selection of servers. The set of servers can be selected based on two concerns: (1) The capacity of quorum access; and (2) The ability to reflect all possible network partitions. The first one is straightforward. The second one stems from the fact that, if no server belongs to one network partition, there would be no updated data for nodes in this partition. Thus, the quorum system should "witness" any network partition. Such set of elements are called detection set. By using (ϵ, k)-detection model, Kleinberg et al. [14] show that there is an (ϵ, k)-detection set if the size is bounded by a polynomial in k and ϵ, independent of the size of the network.

Another assumption that we made is that the quorum placement is known, which represents the map from elements of quorums to physical nodes in the network. This assumption, however, can be also relaxed. The quorum placement can be determined to minimize the average delay for nodes to access quorums. Gupta et al. [5] provide approximation algorithms for the max-delay and total delay to place quorums in a physical network. However, in dynamic networks, where network topology changes over time, the quorum placement problem becomes much more complicated and challenging.

References

[1] A. E. Abbadi, D. Skeen, and F. Cristian. An efficient, fault-tolerant protocol for replicated data management. In *Proceeding of the 5th ACM SIGACT/SIGMOD Conference on Principles of Database Systems*, pages 215–229, 1985.

[2] Y. Amir and A. Wool. Optimal availability quorum systems: theory and practice. *Inform. Process. Lett.*, 65(5):223–228, 1998.

[3] J. Barreta and P. Ferreira. The availability and performace of epidemic quorum algorithms. *Technical Report 10/2007, INESC-ID*, February 2007.

[4] M. Fazio, M. Villari, and A. Puliafito. Ip address autoconfiguration in ad hoc networks: design, implementation and measurements. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 50(7):898–920, 2006.

[5] A. Gupta, B. M. Maggs, F. Oprea, and M. K. Reiter. Quorum placement in networks to minimize access delays. In *Proceedings of 24th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 87–96, 2005.

[6] Z. Haas and B. Liang. Ad hoc mobility management with uniform quorum systems. *IEEE/ACM Transactions on Networking*, 7(2):409–418, 1999.

[7] K. Han, B. Ravindran, and E.D.Jensen. Rtg-l: Dependably scheduling real-time distributable threads in large-scale, unreliable networks. In *Proceedings of IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2007.

[8] K. Han, B. Ravindran, and E.D.Jensen. Rtmg: Scheduling real-time dstributable threads in large-scale, unreliable networks with low message overhead. In *Proceeding of the 13th International Conference on Parallel and Distributed Systems (ICPADS)*, 2007.

[9] M. Herlihy. Dynamic quorum adjustment for partitioned data. *ACM Transactions on Database Systems*, 12(2):170–194, June 1987.

[10] J. Holliday, R. Steinke, D. Agrawal, and A. E. Abbadi. Epidemic quorums for managing replicated databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(5):1218–1238, 2003.

[11] S. Jajodia and D. Mutchler. Dynamic voting algorithms for maintaining the consistency of a replicated database. *ACM Transactions on Database Systems*, 15(2):230–280, June 1990.

[12] D. B. Johnson, D. A. Maltz, and J. Broch. Dsr: The dynamic source routing protocol for multihop wireless ad hoc networks. In *Ad Hoc Networking*, pages 139–172. Addison-Wesley, 2001.

[13] G. Karumanchi, S. Muralidharan, and R. Prakash. Information dissemination in partitionable mobile ad hoc networks. In *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems (SRDS)*, 1999.

[14] J. Kleinberg, M. Sandler, and A. Slivkins. Network failure detection and graph connectivity. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 76–185, 2004.

[15] J. Luo, J.-P. Hubaux, and P. T. Eugster. Pan: providing reliable storage in mobile ad hoc networks with probabilistic quorum systems. In *MobiHoc*, pages 1–12, 2003.

[16] D. Makhi and M. Reiter. Byzantine quorum system. *Distributed Computing*, 11(4):203–213, Octobers 1998.

[17] M. Naor and U. Wieder. Scalable and dynamic quorum systems. In *Proceedings of 22nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2003.

[18] M. Naor and A. Wool. The load, capacity, and availability of quorum systems. *SIAM Journal on Computing*, 27(2):423–447, April 1998.

[19] K. Romer. Time synchronization in ad hoc networks. In *Mobihoc*, pages 173–182, 2001.

[20] T.Zippan and H.Ritter. A survey of gamers expectations of future mobile gaming trends. *Technical Report B04-04 Freie Universitat Berlin*, April 2004.

[21] B. Zhang, K. Han, B. Ravindran, and E.D.Jensen. Rtqg: Real-time quorum-based gossip protocol for unreliable networks. In *Proceeding of the 3rd International Conference on Availability, Reliability and Security (ARES)*, pages 564–571.