Timed Quorum Systems for Large-Scale and Dynamic Environments

Vincent Gramoli[1,2,⋆] and Michel Raynal[2]

[1] INRIA Futurs,
Parc Club Orsay Université, 91893 Orsay, France
vgramoli@irisa.fr
[2] Université de Rennes 1 and INRIA Research Centre Rennes,
Campus de Beaulieu, 35042 Rennes, France
raynal@irisa.fr

Abstract. This paper presents Timed Quorum System (TQS), a quorum system for large-scale and dynamic systems. TQS provides guarantees that two quorums, accessed at instances of time that are close together, intersect with high probability. We present an algorithm that implements TQS at its core and that provides operations that respect atomicity with high probability. This TQS implementation has quorums of size $O(\sqrt{nD})$ and expected access time of $O(\log \sqrt{nD})$ message delays, where n measures the size of the system and D is a required parameter to handle dynamism. This algorithm is shown to have complexity sub-linear in size and dynamism of the system, and hence to be scalable. It is also shown that for systems where operations are frequent enough, the system achieves the lower bound on quorum size for probabilistic quorums in static systems, and it is thus optimal in that sense.

Keywords: Time, Quorums, Churn, Scalability, Probabilistic atomicity.

1 Introduction

The need of resources is a main motivation behind distributed systems. Take peer-to-peer (p2p) systems as an example. A p2p system is a distributed system that has no centralized control. The p2p systems have gained in popularity with the massive utilization of file-sharing applications over the Internet, since 2000. These systems propose a tremendous amount of file resources. More generally, there is an increasing amount of various computing devices surrounding us: IDC predicts that there will be 17 billions of traditional network devices by 2012. In such a context, it is common knowledge that scalability has become one of the most important challenges of today's distributed systems.

The scale-shift of distributed systems modifies the way computational entities communicate. Energy dependence, disconnection, malfunctioning, and environmental factors affect the availability of various computational entities independently. This translates into irregular periods of activity during which an entity can receive messages

⋆ Corresponding author: ASAP Research Group, INRIA Futurs, 3-4 rue Jacques Monod, 91893 Orsay, France; fax: +33 1 74 85 42 42.

E. Tovar, P. Tsigas, and H. Fouchal (Eds.): OPODIS 2007, LNCS 4878, pp. 429–442, 2007.

or compute tasks. As a result of this independent and periodic behaviors, these systems are inherently highly dynamic.

Quorum system is a largely adopted solution for communication in message-passing system. Despite the interest for emulating shared-memory in dynamic systems [1,2,3,4], there is no scalable solution due to the cost of their failure handling mechanism or their operation complexity.

This paper describes a Timed Quorum System (TQS) for large-scale dynamic systems. TQS provides guarantees that two quorums, accessed at instances of time that are close together, intersect with high probability. We propose an algorithm that implements TQS and that verifies probabilistic atomicity: a consistency criterion that requires each operation to respect atomicity with high probability. This algorithm is analyzed to show scalability in terms of complexity. More precisely, the expected time complexity is $O(\log \sqrt{nD})$ message delays, where n measures the size of the system and D is a required parameter to handle dynamism. It is also shown that for systems where operations are frequent enough, the algorithm achieves a lower bound, $O(\sqrt{n})$, on quorum size for probabilistic quorum in static systems, and it is thus optimal in that sense. In addition, we show that our solution does not need a reconfiguration mechanism to tolerate the dynamic and fault-prone environment for which it is designed due to the integration of a replication mechanism on top of the operations performed on the replicated object.

Related Work. Dynamic quorum systems are a very active research area. Some dynamic quorum systems rely on failure detectors where quorums are dynamically redefined according to failure detection, This adaption leads to a redefinition of the quorums [1,5] or to the replacement of the failed nodes in the quorums [6,7,8]. For example, in [7], a communication structure is continuously maintained to ensure that quorums intersect at all time (with high probability).

Other solutions rely on periodic reconfigurations [2,4] where the quorum systems are subsequently replaced. These solutions are different from the previous ones since the newly installed quorums do not need to intersect with the previous ones. In [3] a quorum abstraction is defined by two properties: (i) intersection and (ii) progress, in which the notion of time is introduced. First, a quorum of a certain type intersects the quorum of another type contacted subsequently. Second, each node of a quorum remains active between the time the quorum starts being probed and the time the quorum stopped being probed.

As far as we know, TQS is the first quorum system that expresses guarantees that are both timely and probabilistic. Time and probability relax the traditional intersection requirement of quorums. We present a scalable emulation of a probabilistic atomic memory where each operation is atomic with high probability and complexity is sublinear in both the size and the dynamism of the system.

Roadmap. Section 2 presents the model and describes the problem. Section 3 defines the Timed Quorum System. Section 4 presents a shared object by specifying read and write operations based on a TQS. Section 5 shows that this solution implements TQS and verifies probabilistic atomicity, and analyses the complexity of the algorithm. Finally, Section 6 concludes the paper.

2 System Model and Problem Definition

2.1 Model

The computation model is very simple. The system consists of n nodes. It is dynamic in the following sense. Every time unit, cn nodes leave the system and cn nodes enter the system, where c is an upper bound on the percentage of nodes that enter/leave the system per time unit and is called the *churn*; this can be seen as new nodes "replacing" leaving nodes. A node leaves the system either voluntarily or because it crashes. A node that leaves the system does not reenter it later. (Practically, this means that, when a node reenters the system, it is considered as a new node; all its previous knowledge of the system state is lost.) For the sake of simplicity, it is assumed that for any subset S of nodes, the portion of replaced nodes is $c|S|$. As explained below, the model can be made more complex. The *universe* U denotes all the nodes of the system, plus the ones that have already left the system and the ones that have not joined the system yet.

2.2 Problem

Most of the dynamic models assume that dynamic events are dependent from each other: only a limited number of nodes leave and join the system during a bounded period of time. For instance in [4], it is assumed that node departures are dependent: quorum replication ensures that all nodes of at least any two quorums remain active between the occurrence of two reconfigurations. However, in a real dynamic system, nodes act independently. Due to this independence, even with a precise knowledge of the past dynamic events, one can not predict the future behavior of a node. That is, putting this observation into the quorums context, it translates into the impossibility of predicting deterministically whether quorums intersect.

In contrast, TQS requires that quorums intersect with high probability. This allows to use a more realistic model in which there is a certain probability that nodes leave/join the system at the same time. That is, *the goal here is to measure the probability that any two quorums intersect as time elapses*. Observe that, realistically, the probability that k nodes leave the system increases as time elapses. As a result, the probability that a quorum $Q(t)$ probed at time t and that a quorum $Q(t')$ probed at time t' intersect decreases as the period $|t'-t|$ increases. In the following we propose an implementation of TQS where probability of intersection remains high.

More precisely, each quorum of our TQS implementation is defined for a given time t. Each quorum $Q(t)$ has a lifetime Δ that represents a period during which the quorum is *reachable*. Differently from availability defined in [5], reachability does not depend on the number of nodes that are failed in a quorum system because this number is unpredictable in dynamic systems. Instead, a $Q(t)$ quorum is reachable if at least one node of quorum $Q(t)$ is reached with high probability: if two quorums are reachable at the same time, they intersect with high probability. More generally, let two quorums $Q(t)$ and $Q(t')$ of a TQS be reachable during Δ time (their lifetime is Δ); if $|t-t'| \leq \Delta$ then $Q(t)$ and $Q(t')$ intersect with high probability.

2.3 Preliminary Notations and Definitions

This section defines several terms that are used in the algorithm description. Recall that a shared object is accessed through read operations, which return the current value of the object, and write operations, which modify the current value of the object. Initially, any object has a default value v_0 that is replicated at a set of nodes and V denotes the set of all possible values present in the system. An object is accessed by read or write operations initiated by some nodes i at time $t \in T$ that return or modify the object value v. (T is the set of all possible time instances.) If a node initiates an operation, then it is referred to as a *client*. All nodes of the system, including nodes of the quorum system, can initiate a read or a write operation, i.e., all nodes are potential clients and the multi-reader/multi-writer model is used. In the following we only consider a single object accessed by operations.

First, to clarify the notion of currency when concurrency happens, it is important to explain what are the up-to-date values that could be considered as current. We refer to the *last value* as the value associated with the largest tag among all values whose propagation is complete. We refer to the *up-to-date values* at time t as all values v that satisfies one of the following properties: *(i)* value v is the last value or *(ii)* value v is a value whose propagation is ongoing and whose associated tag is at least equal or larger to the tag associated with the last value.

Second, it is important to understand what is a successful phase. The goal of a consultation phase is to return an up-to-date value, whereas the goal of the propagation phase is to propagate an up-to-date value v so that v can be identified as an up-to-date value. Thus, we refer to a *successful phase* as a phase that achieves its goal. Observe that, if the consultation of an operation is unsuccessful, then the subsequent propagation phase of the same operation might propagate a new value with a small tag so that this value will not be identifiable as an up-to-date value. In this case, we say that both the consultation and propagation are unsuccessful phases. A more formal definition of the successful/unsuccessful phase follows.

Definition 1 (Successful Phase). *A consultation phase ϕ is* successful *if and only if it returns an up-to-date value $val(\phi)$. A propagation phase ρ is* successful *if and only if it propagates a tag $tag(\rho)$ largest than any of the tags that were in the system when ρ started. A phase is* unsuccessful *if it is not successful.*

We refer to successful operations as operations whose consultation phase and propagation phase are successful.

TQS ensures that two active quorums will intersect with high probability, however, if no quorum is active, then the value of an object does no longer persist. To ensure that new operations replicate the object value sufficiently, we assume that at last one operation is executed every period Δ. As previously explained this mechanism serves as a continuous replication and replaces the traditional reconfiguration mechanism to cope with accumulated failures.

2.4 Probabilistic Atomic Object

A probabilistic atomic object aims at emulating a memory that offers high quality of service despite large-scale and dynamism. For the sake of tolerating scale-shift and dynamism, we aim at relaxing some properties. However, our goal is to provide each client with a distributed shared memory emulation that offers satisfying quality of service. Quality of service must be formally stated by a consistency criterion that defines the guarantees the application can expect from the memory emulation. We aim at providing quality of service in terms of accuracy of read and write operations. In other words, our goal is to provide the clients with a memory that guarantees that each read or write operation will be successfully executed with high probability. We define the probabilistic atomic object as an atomic object where operation accuracy is ensured with high probability.

Let us first recall properties 2 and 4 of atomicity from Theorem 13.16 of [9] which require that any sequence of invocations and responses of read and write operations applied to x satisfies a partial ordering $\prec$ such that:

– (π_1, π_2)-*ordering*: if the response event of operation π_1 precedes the invocation event of operation π_2, then it is not possible to have $\pi_2 \prec \pi_1$;
– (π_1, π_2)-*return*: the value returned by a read operation π_2 is the value written by the last preceding write operation π_1 regarding to $\prec$ (in case no such write operation π_1 exists, this value returned is the default value).

The definition of probabilistic atomicity is similar to the definition of atomicity: only Properties 2 and 4 are slightly modified, as indicated below.

Definition 2 (Probabilistic Atomic Object). *Let x be a read/write probabilistic atomic object. Let H be a complete sequence of invocations responses of read and write operations applied to object x. The sequence H satisfies probabilistic atomicity if and only if there is a partial ordering $\prec$ on the operations such that the following properties hold:*

1. *For any operation π_2, there are only finitely many successful operations π_1, such that $\pi_1 \prec \pi_2$.*
2. *Let π_1 be a successful operation. Any operation π_2 satisfies (π_1, π_2)-ordering with high probability. (If π_2 does not satisfy it, then π_2 is unsuccessful.)*
3. *if π_1 is a successful write operation and π_2 is any successful operation, then either $\pi_2 \prec \pi_1$ or $\pi_1 \prec \pi_2$;*
4. *Let π_1 be a successful operation. Any operation π_2 satisfies (π_1, π_2)-return with high probability. (If π_2 does not satisfy it, then π_2 is unsuccessful.)*

Observe that the partial ordering is defined on successful operations. That is, either an operation π fails and this operation is considered as unordered or the operation succeeds and is ordered with respect to other successful operations.

Even though an operation succeeds with high probability, there might be a lot of unsuccessful operations in a long enough execution. However, our goal is to provide the operation requester (client) with high guarantee of success for each of its operation request.

3 Timed Quorum System

This section defines Timed Quorum Systems (TQS). Before being created or after its lifetime has elapsed, a quorum is not guaranteed to intersect with any other quorum, however, during its lifetime a quorum is considered as available: two quorums that are available at the same time intersect with high probability. In dynamic systems nodes may leave at any time, but this probability is bounded, thus it is possible to determine the intersection probability of two quorums.

Next, we formally define TQS that are especially suited for dynamic systems. Recall that the universe U contains the set of all possible nodes, including nodes that have not yet joined the system. First, we restate the definition of a *set system* as a set of subsets of a universe of nodes.

Definition 3 (Set System). *A* set system $\mathcal{S}$ *over a universe* U *is a set of subsets of* U.

Then, we define the timed access strategy as a probability distribution over a set system that may vary over time. This definition is motivated by the fact that an access strategy defined over a set $\mathcal{S}$ can evolve. To compare with the existing probabilistic dynamic quorums, in [7] the authors defined a dynamic quorum system using an evolving strategy that might replace some nodes of a quorum while its access strategy remains identical despite this evolution. Unlike the dynamic quorum approach, we need a more general framework to consider quorums that are different not only because of their structure but also because of how likely they can be accessed. The timely access strategy adds a time parameter to the seminal definition access strategy given by Malkhi et al. [10]. A timely access strategy is allowed to evolve over time.

Definition 4 (Timed Access Strategy). *A* timed access strategy $\omega(t)$ *for a set system* $\mathcal{S}$ *at time* $t \in T$ *is a probability distribution on the elements of* $\mathcal{S}$ *at time* t*. That is,* $\omega : \mathcal{S} \times T \rightarrow [0, 1]$ *satisfies at any time* $t \in T$*:* $\sum_{s \in \mathcal{S}} \omega(s, t) = 1$.

Informally, at two distinct instants $t_1 \in T$ and $t_2 \in T$, an access strategy might be different for any reason. For instance, consider that some node i is active at time t_1 while the same node i is failed at time t_2, hence it is likely that if $i \in s$, then $\omega(s, t_1) \neq 0$ while $\omega(s, t_2) = 0$. This is due to the fact that a node is reachable only when it is active.

Definition 5 (Δ-Timed Quorum System). *Let* $\mathcal{Q}$ *be a set system, let* $\omega(t)$ *be a timed access strategy for* $\mathcal{Q}$ *at time* t*, and let* $0 < \epsilon < 1$ *be given. The tuple* $\langle \mathcal{Q}, \omega(t) \rangle$ *is a* Δ-timed quorum system *if for any quorums* $Q(t_1) \in \mathcal{Q}$ *accessed with strategy* $\omega(t_1)$ *and* $Q(t_2) \in \mathcal{Q}$ *accessed with strategy* $\omega(t_2)$*, we have:*

$$\Delta \geq |t_1 - t_2| \Rightarrow \Pr[Q(t_1) \cap Q(t_2) \neq \emptyset] \geq 1 - \epsilon.$$

4 Timed Quorum System Implementation for Probabilistic Atomic Memory

In the following, we present a structureless memory. The quorum systems this memory uses does not rely on any structure, that is, the quorum system is flexible. In contrast with using a logical structured overlay (e.g., [11]) for communication among quorum system nodes, we use an unstructured communication overlay [12]. The lack of structure presents several benefits. First, there is no need to re-adapt the structure at each dynamic event. Second, there is no need for detecting failure. Our solution proposes a periodic replication. To ensure the persistence of an object value despite unbounded leaves, the value must be replicated an unbounded number of times. The solution we propose requires periodic operations and an approximation of the system size. Although we do not focus on the problem of approximating the system size n, we suggest the use of existing protocols approximating closely the system size in dynamic systems [13].

4.1 Replicating During Client Operations

Benefiting from the natural primitive of the distributed shared memory, values are replicated using operations. Any operation has at its heart a quorum-probe that replicates value. On the one hand, it is natural to think of a write operation as an operation that replicates a value. On the other hand, in [14] a Theorem shows that "read must write", meaning that a read operation must replicates the value it returns. This raises the question: if operations replicate, why does a memory need additional replication mechanism? In large-scale systems, it is also reasonable to assume that shared objects are frequently accessed because of the large number of participants. Since operations provide replication and shared objects experience frequent operation requests in large-scale systems, frequent replications can be mainly ensured by client operations.

4.2 Quorum Probe

The algorithm is divided in three distinct parts that represent the state of the algorithm (Lines 1–12), the actions initiated by a client (Lines 13–42), and the actions taken upon reception of messages by a node (Lines 43–63), respectively. Each node i has its own copy of the object called its value val_i and an associated tag tag_i. Field tag is a couple of a counter and a node identifier and represents, at any time, the version number of its corresponding value val. We assume that, initially, there are q nodes that own the default value of the object, the other nodes have their values val set to $\perp$ and all their tags are set to $\langle 0, 0 \rangle$.

Each read and write operation is executed by client i in two subsequent phases, each disseminating a message to $q = O(\sqrt{nD})$ nodes, where $D = (1-c)^{-\Delta}$ represents the inverse of the portion of nodes that stayed in the system during period Δ. This dynamic parameter D is required to handle churn c during period Δ.[1] The two successive phases are called the *consultation phase* and the *propagation phase*. The consultation phase aims at consulting the up-to-date value of the object that is present in the system. (This value is identifiable because it is associated with the largest tag present in the system.) More precisely, client i disseminates a consultation message to q nodes so that each receiver j responds with a message containing value val_j and tag tag_j so that client i can update val_i and tag_i. In fact, i updates val_i and tag_i if and only if the tag_i has either a smaller counter than tag_j or it has an equal counter but a smaller identifier $i < j$ (node identifiers are always distinct); in this case we say $tag_i < tag_j$ for short (cf. Lines 51 and 53). Ideally, at the end of the consultation phase client i has set its value val_i to the up-to-date value. Read and write operations differ from the value and tag that are propagated by the client i. Specifically, in case of a read, client i propagates the value

[1] It is shown in [10] that $q = O(\sqrt{n})$ is sufficient in static systems.

Algorithm 1. Disseminating Memory at node i

1: **State of node** i**:**
2: $q = \frac{\beta\sqrt{n}}{(1-c)^{\frac{\Delta}{2}}}$, the quorum size
3: $\ell, k \in \mathbb{N}$ the disseminating parameters taken such that $\frac{k^{l+1}-1}{k-1} \geq q$
4: $val \in V$, the value of the object, initially $\perp$
5: tag, a couple of fields:
6: $counter \in \mathbb{N}$, initially 0
7: $id \in I$, an identifier initially i
8: $marked$, an array of booleans initially false at all indices
9: $sent\text{-}to\text{-}nbrs1$, $sent\text{-}to\text{-}nbrs2$ two sets of node identifiers, initially $\emptyset$
10: $rcvd\text{-}from\text{-}qnodes$, an infinite array of identifier sets, initially $\emptyset$ at all indices
11: $sn \in \mathbb{N}$, the sequence number of the current phase, initially 0
12: $father \in I$, the id of the node that disseminated a message to i, initially i

13: **Read**$_i$**:**
14: $\langle val, tag\rangle \leftarrow$ **Consult**()
15: **Propagate**($\langle val, tag\rangle$)

16: **Write**(v)$_i$**:**
17: $\langle *, tag\rangle \leftarrow$**Consult**()
18: $tag.counter \leftarrow tag.counter + 1$
19: $tag.id \leftarrow i$
20: $val \leftarrow v$
21: **Propagate**($\langle val, tag\rangle$)

22: **Consult**$_i$**:**
23: $ttl \leftarrow \ell$
24: $sn \leftarrow sn + 1$
25: **while** ($|sent\text{-}to\text{-}nbrs1| < k$) **do**
26: send$\langle$CONS$, val, tag, ttl, i, sn\rangle$ to
27: a set J of ($k - |sent\text{-}to\text{-}nbrs1|$) neighbors $\neq father$
28: $sent\text{-}to\text{-}nbrs1 \leftarrow sent\text{-}to\text{-}nbrs1 \cup J$
29: **end while**
30: $sent\text{-}to\text{-}nbrs1 \leftarrow \emptyset$
31: **wait until** $|rcvd\text{-}from\text{-}qnodes[sn]| \geq q$
32: **return** ($\langle val, tag\rangle$)

and tag pair freshly consulted, while in the case of write, client i propagates the new value to write with a strictly larger tag than the largest tag that i has consulted so far. The propagation phase propagates the corresponding value and tag by dissemination among nodes.

Next, we focus on the dissemination procedure that is at the heart of the consultation and propagation phases. There are two parameters, ℓ, k, that define the way all consultation or propagation messages are disseminated. Parameter ℓ indicates the depth of the dissemination, it is used to set a time-to-live field ttl that is decremented at each intermediary node that participates in the dissemination; if $ttl = 0$, then dissemination is complete. Parameter k represents the number of neighbors that are contacted by each intermediary participating node. Together, parameters ℓ and k define the number of nodes that are contacted during a dissemination. This number is $\frac{k^{\ell+1}-1}{k-1}$ (Line 3) and

33: **Propagate(** $\langle$ *val*,*t* $\rangle)_i$**:**
34: $ttl \leftarrow \ell$
35: $sn \leftarrow sn + 1$
36: **while** $(|sent\text{-}to\text{-}nbrs1| < k)$ **do**
37: send$\langle$PROP$, val, tag, ttl, i, sn\rangle$ to
38: a set J of $(k - |sent\text{-}to\text{-}nbrs1|)$ neighbors $\neq father$
39: $sent\text{-}to\text{-}nbrs1 \leftarrow sent\text{-}to\text{-}nbrs1 \cup J$
40: **end while**
41: $sent\text{-}to\text{-}nbrs1 \leftarrow \emptyset$
42: **wait until** $|rcvd\text{-}from\text{-}qnodes[sn]| \geq q$

43: **Participate**$_i$ **(Activated upon reception of a message):**
44: recv$\langle type, v, t, ttl, client\text{-}id, sn\rangle$ from j
45: **if** $(marked[sn])$ **then**
46: send$\langle type, v, t, ttl, client\text{-}id, sn\rangle$ to a neighbor $\neq j$
47: **else**
48: $marked[sn] \leftarrow$ true
49: $father \leftarrow j$
50: **if** $((type =$ CONS$))$ **then** $\langle v, t\rangle \leftarrow \langle val, tag\rangle$
51: **if** $((type =$ PROP$))$ **then** $\langle val, tag\rangle \leftarrow \langle v, t\rangle$
52: **if** $(type =$ RESP$)$ **then**
53: **if** $(tag < t)$ **then** $\langle val, tag\rangle \leftarrow \langle v, t\rangle$
54: $rcvd\text{-}from\text{-}qnodes[sn] \leftarrow rcvd\text{-}from\text{-}qnodes[sn] \cup \{j\}$
55: $ttl \leftarrow ttl - 1$
56: **if** $(ttl > 0)$ **then**
57: **while** $(|sent\text{-}to\text{-}nbrs2| < k)$ **do**
58: send$\langle type, v, t, ttl, client\text{-}id, sn\rangle$ to
59: a set J of $(k - |sent\text{-}to\text{-}nbrs2|)$ neighbors $\neq father$
60: $sent\text{-}to\text{-}nbrs2 \leftarrow sent\text{-}to\text{-}nbrs2 \cup J$
61: **end while**
62: $sent\text{-}to\text{-}nbrs2 \leftarrow \emptyset$
63: send $\langle$RESP$, val, tag, ttl, \perp, sn\rangle$ to $client\text{-}id$

represents the number of nodes in a balanced tree of depth ℓ and width k: each node having exactly k children. (This value is provable by recurrence on the depth ℓ of the tree.) Observe that ℓ and k are chosen such that the number of nodes that are contacted during a dissemination be larger than q as written Line 3.

There are three kinds of messages denoted by message *type*: CONS, PROP, RESP indicating if the message is a consultation message, a propagation message, or a response to any of the two other messages. When a new phase starts at client i, a time-to-live field ttl is set to ℓ and a sequence number sn is incremented. This number is used in message exchanges to indicate whether a message corresponds to the right phase. Then the phase proceeds in sending continuously messages to k neighbors waiting for their answer (Lines 25–29 and Lines 36–40). When the k neighbors answer, client i knows that the dissemination is ongoing. Then client i receives all messages until a large enough number q of nodes have responded in this phase, i.e., with the right sequence number (Lines 31, 42). If so, then the phase is complete.

Observe that during the dissemination, messages are simply marked (if they have not already been marked), responded (to client i), and re-forwarded to other neighbors (until ttl is null). Messages are marked by the node i that participates in a dissemination for preventing node i from participating multiple times in the same dissemination (Line 45). As a result, if node i is asked several times to participate, it first participates (Lines 48–63) and then it asks another node to participate (Lines 45–47). More precisely, if $marked[sn]$ is true, then node i re-forwards messages of sequence number sn without decrementing the ttl. Observe that phase termination and dissemination termination depends on the number of participants rather than the number of responses: it is important that enough participants participate in each dissemination for the phase to eventually end.

4.3 Contacting Participants Randomly

In order to contact the participants randomly, we implemented a membership protocol [12]. This protocol is based on Cyclon [15], thus, it is lightweight and fault-tolerant. Each node has a set of m neighbors called its view $\mathcal{N}_i$, it periodically updates its view and recomputes its set of neighbors. Our underlying membership algorithm provides each node with a set of $m \geq k+1$ neighbors, so that phases of Algorithm 1 disseminate through a tree of degree $k+1$. This algorithm shuffles the view at each cycle of its execution so that it provides randomness in the choice of neighbors. Moreover, it has been shown by simulation that the communication graph obtained with Cyclon is similar to a random graph where neighbors are picked uniformly among nodes [16].

For the sake of uniformity, the membership procedure is similar to the Cyclon algorithm: each node i maintains a view $\mathcal{N}_i$ containing one entry per neighbor. The entry of a neighbor j corresponds to a tuple containing the neighbor identifier and its age. Node i copies its view, selects the oldest neighbor j of its view, removes the entry e_j of j from the copy of its view, and finally sends the resulting copy to j. When j receives the view, j sends its own view back to i discarding possible pointers to i, and i and j update their view with the one they receive by firstly keeping the entries they received. The age of neighbor j entry denotes the time that elapsed since the last message from j has been received; this is used to remove failed neighbor from the list. This variant of Cyclon exchanges all entries of the view at each step like in [17].

5 Correctness Proof and Performance Analysis

Here, we show that Algorithm 1 implements a Timed Quorum System and that it emulates the probabilistic atomic object abstraction defined in Definition 2. The key points of this proof are to show that quorums are sufficiently re-activated by new operations to face dynamism and that subsequent quorums intersect with very high probability to achieve probabilistic atomicity. The proofs of Lemmas and Theorems can be found in [18].

5.1 Assumptions and Notations

First, we only consider executions starting with at least q nodes that own the default value of the object. In these executions, at least one propagation phase from a successful operation starts every Δ time units and let the time of any phase be bounded by δ time units. We assume that during a propagation that propagates a value v to q nodes and that executes between time t and $t + \delta$, there is at least one instant t' where the q nodes own value v simultaneously. This instant, t', can occur arbitrarily between time t and $t + \delta$. Even if this assumption may not seem realistic since propagation occurs in parallel of churn (i.e., at the time the propagation contacts the q^{th} node the first contacted node may have left the system), our motivations for this assumption comes from the sake of clarity of the proof and we claim that the absence of this assumption leads to the same results.

Second, we assume that our underlying communication protocol provides each node with a view that represents a set of neighbors uniformly drawn at random among the set of all active nodes. This assumption is reasonable since, as already mentioned, the underlying algorithm is based on Cyclon that shuffles node views and provides communication graph similar to a random graph [16].

Next, we show that Algorithm 1 implements a probabilistic object. Observe that the liveness part of this proof relies simply on the activity of neighbors, and the fact that messages are eventually received. More precisely, by examination of the code of Algorithm 1, messages are gossiped among neighbors while neighbors are uniformly chosen. It is clear that operation termination depends on eventual message delivery. As a result, only the safety part of the proof follows. In the following, $val(\phi)$ (resp. $tag(\phi)$) denote, the value (resp. tag) consulted/propagated by phase ϕ.

5.2 Correctness Proof

First, we restate a Lemma appeared in [19] that computes the ratio of nodes that leave the system as time elapses, given a churn of c. The result is the ratio of nodes that leave and join, and helps computing the probability that up-to-date values remain reachable despite dynamism.

Lemma 1. *The ratio of initial nodes that have been replaced after τ time units is at most $C = 1 - (1 - c)^{\tau}$.*

The reader will find the proof of this Lemma 1 in [19]. The following Lemma gives a lower bound on the number of nodes that own the up-to-date value at any time in the system. (Recall that an up-to-date value is either the value with the largest tag and whose propagation is complete, or any value with a larger tag, but whose propagation is ongoing.)

Lemma 2. *At any time t in the system, the number of nodes that own an up-to-date value is at least $q(1 - c)^{\Delta}$, where Δ is the maximum period between two subsequent and successful propagation starting time instances, q is the quorum size, and c is the churn of the system.*

The following fact gives this well-known bound on the exponential function, provable using the Euler's method.

Fact 1. $(1 + \frac{x}{n})^n \leq e^x$ *for* $n > |x|$.

Next Lemma lower bounds the probability that any consultation consults an up-to-date value v. Recall that sometime it might happen that a value v' is unsuccessfully propagated. This may happen when a write operation fails in consulting the largest tag just before propagating value v'. Observe that in any case, a successful consultation returns only successfully propagated values.

Lemma 3. *If the number of nodes that own an up-to-date value is at least* $q(1-c)^{\Delta}$ *during the whole period of execution of consultation* ϕ*, then consultation* ϕ *succeeds with high probability (*$\geq 1 - e^{-\beta^2}$*, with* β *a constant).*

This Corollary simply concludes the two previous Lemmas stating that any consultation executed in the system succeeds by returning an up-to-date value.

Corollary 1. *Any consultation* ϕ *succeeds with high probability (*$\geq 1 - e^{-\beta^2}$*, with* β *a constant).*

Proof. The result is straightforward from Lemma 2 and Lemma 3. □ Last but not least, the two Theorems conclude the proof by showing that Algorithm 1 implements a Δ-TQS and verifies probabilistic atomicity.

Theorem 1. *Algorithm 1 implements a* Δ*-Timed Quorum System, where* Δ *is the maximum period between two subsequent and successful propagation starting time instances.*

Theorem 2. *Algorithm 1 implements a probabilistic atomic object.*

5.3 Performance Analysis

The following Lemmas show the performance of our solution: the first Lemma gives the expected message complexity of our solution while the second Lemma gives the expected time complexity of our solution. Observe first that operations complete provided that sent messages are reliably delivered. Building onto this assumption, an operation complete after contacting $O(\sqrt{nD})$ nodes. The following Lemma shows this result.

Lemma 4. *An operation completes after having contacted* $O(\sqrt{nD})$ *nodes.*

Proof. This is straightforward from the fact that termination of the dissemination process is conditioned to the number of distinct nodes contacted: $q = O(\sqrt{nD})$, with $D = (1-c)^{-\Delta}$ (cf. Line 2). Since there are two disseminating phases in each operation, an operation is executed after contacting $O(\sqrt{nD})$ nodes. □

Next Lemma indicates that an operation terminates in $O(\log \sqrt{nD})$ message delays, in expectation.

Lemma 5. *If messages are not lost, the expected time of an operation is $O(\log \sqrt{nD})$ message delays.*

Proof. The proof relies on the fact that q' nodes are contacted uniformly at random with replacement. In expectation, the number q' that must be contacted to obtain q distinct nodes is $q' = q = O(\sqrt{nD})$. Since nodes are contacted in parallel along a tree of depth ℓ and width k, the time required to contact all the nodes on the tree is $\ell = O(\log_k q)$. That is, it is done in $\ell = O(\log_k \sqrt{nD})$ message delays. □

6 Conclusion

This paper addressed the problem of emulating a distributed shared memory that copes with scalability and dynamism while being efficient. TQS ensures probabilistic intersection of quorums in a timely fashion. Interestingly, we showed that some TQS implementation verifies a consistency criterion weaker but similar to atomicity: probabilistic atomicity. Hence, any operation satisfies the ordering required for atomicity with high probability. The given implementation of TQS verifies probabilistic atomicity, provides lightweight ($O(\sqrt{nD})$ messages) and fast ($O(\log \sqrt{nD})$ message delays) operations, and does not require reconfiguration mechanism since periodic replication is piggybacked into operations.

Since we started tackling the problem that node can fail independently, we are now able to implement probabilistic memory into more realistic models. Previous solutions required that a very few amount of nodes could fail at the same time. More realistically, a model should allow node to act independently. Thus, an interesting question is: what probabilistic consistency can TQS achieve in such a more realistic model?

Acknowledgments

We are grateful to Anne-Marie Kermarrec, Achour Mostéfaoui for fruitful discussions about gossip-based algorithms and dynamic systems, and the anonymous referees for their helpful comments.

References

1. Herlihy, M.: Dynamic quorum adjustment for partitioned data. ACM Trans. Database Syst. 12(2), 170–194 (1987)
2. Lynch, N., Shvartsman, A.: RAMBO: A reconfigurable atomic memory service for dynamic networks. In: Malkhi, D. (ed.) DISC 2002. LNCS, vol. 2508, pp. 173–190. Springer, Heidelberg (2002)
3. Friedman, R., Raynal, M., Travers, C.: Two abstractions for implementing atomic objects in dynamic systems. In: Anderson, J.H., Prencipe, G., Wattenhofer, R. (eds.) OPODIS 2005. LNCS, vol. 3974, pp. 73–87. Springer, Heidelberg (2006)
4. Chockler, G., Gilbert, S., Gramoli, V., Musial, P., Shvartsman, A.: Reconfigurable distributed storage for dynamic networks. In: Anderson, J.H., Prencipe, G., Wattenhofer, R. (eds.) OPODIS 2005. LNCS, vol. 3974, pp. 214–219. Springer, Heidelberg (2006)

5. Naor, M., Wool, A.: The load, capacity, and availability of quorum systems. SIAM Journal on Computing 27(2), 423–447 (1998)
6. Nadav, U., Naor, M.: The dynamic and-or quorum system. In: Fraigniaud, P. (ed.) DISC 2005. LNCS, vol. 3724, pp. 472–486. Springer, Heidelberg (2005)
7. Abraham, I., Malkhi, D.: Probabilistic quorum systems for dynamic systems. Distributed Computing 18(2), 113–124 (2005)
8. Gramoli, V., Anceaume, E., Virgillito, A.: SQUARE: Scalable quorum-based atomic memory with local reconfiguration. In: SAC 2007. Proceedings of the 22nd ACM Symposium on Applied Computing, pp. 574–579. ACM Press, New York (2007)
9. Lynch, N.: Distributed Algorithms. Morgan Kaufmann, San Francisco (1996)
10. Malkhi, D., Reiter, M., Wool, A., Wright, R.: Probabilistic quorum systems. The Information and Computation Journal 170(2), 184–206 (2001)
11. Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: ACM SIGCOMM 2001, San Diego, CA (2001)
12. Ganesh, A.J., Kermarrec, A.M., Massoulié, L.: Peer-to-peer membership management for gossip-based protocols. IEEE Trans. Comput. 52(2), 139–149 (2003)
13. Ganesh, A.J., Kermarrec, A.M., Le Merrer, E., Massoulié, L.: Peer counting and sampling in overlay networks based on random walks. In: Distributed Computing (2007)
14. Attiya, H., Welch, J.: Distributed Computing. In: Fundamentals, Simulations, and Advanced Topics., McGraw-Hill, New York (1998)
15. Voulgaris, S., Gavidia, D., van Steen, M.: Cyclon: Inexpensive membership management for unstructured p2p overlays. Journal of Network and Systems Management 13(2), 197–217 (2005)
16. Iwanicki, K.: Gossip-based dissemination of time. Master's thesis, Warsaw University - Vrije Universiteit Amsterdam (2005)
17. Fernández, A., Gramoli, V., Jiménez, E., Kermarrec, A.M., Raynal, M.: Distributed slicing in dynamic systems. In: ICDCS 2007. Proceedings of the 27th International Conference on Distributed Computing Systems, IEEE Computer Society Press, Los Alamitos (2007)
18. Gramoli, V., Raynal, M.: Timed quorum system for large-scale dynamic environments. Technical Report 1859, INRIA Research Centre Rennes (2007)
19. Gramoli, V., Kermarrec, A.M., Mostefaoui, A., Raynal, M., Sericola, B.: Core persistence in peer-to-peer systems: Relating size to lifetime. In: Meersman, R., Tari, Z., Herrero, P. (eds.) On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops. LNCS, vol. 4278, pp. 1470–1479. Springer, Heidelberg (2006)