P2P Architecture for Self-* Atomic Memory

Emmanuelle Anceaume* Maria Gradinariu* Vincent Gramoli*
Antonino Virgillito*†

* IRISA - INRIA Rennes, CNRS
Campus de Beaulieu
35042 Rennes
France

† Università di Roma "La Sapienza"
Via Salaria, 113
00198 Roma
Italy

Abstract

We propose an architecture for self-adjusting and self-healing atomic memory *in highly dynamic systems exploiting peer-to-peer (p2p) techniques. Our approach, named* SAM*, brings together new and old research areas such as p2p overlays, dynamic quorums and replica control. In SAM, nodes form a connected overlay. To emulate the behavior of an atomic memory we use intersected sets of nodes, namely* quorums*, where each node hosts a replica of an object. In our approach, a quorum set is obtained by performing a deterministic traversal of the overlay. The SAM overlay features self-* capabilities: that is, the overlay self-heals on the fly when nodes hosting replicas leave the system and the number of active replicas in the overlay dynamically self-adjusts with respect to the object load. In particular, SAM pushes requests from loaded replicas to less solicited replicas. If such replicas do not exist, the replicas overlay self-adjusts to absorb the extra load without breaking the atomicity. We propose a distributed implementation of SAM where nodes exploit only a restricted local view of the system, for the sake of scalability.*

1. Introduction

The real notoriety of peer-to-peer file sharing guarantees the subsequent success of p2p systems in commercial applications. However, in order to move further, the p2p community needs to focus on designing fundamental abstractions that offer strong computational guarantees. Atomic memory is a basic service in distributed computing that offers a persistent storage with linearizable read/write semantics. This service has a broad Internet-scale applications. Consider e-auctions, for example. The atomic memory could be used by each auctioneer to write its bid and read the others bids.

Designing atomic memory in p2p systems faces several problems. P2p systems are by their nature ad-hoc distributed systems without any organization or centralized control. Unlike classical distributed systems, p2p systems encompass processes (peers) that experience highly dynamic behaviors including spontaneous join and leave or change in their local connections. The high dynamism of the network has a tremendous impact on data availability. The use of classical distributed computing solutions, like replication, for example, introduces an extra cost related to: (1) maintaining a sufficient number of replicas despite frequent disconnections; and (2) maintaining the consistency among the replicas. The former problem can be solved using *self-healing* techniques while the latter one finds solutions in the use of *dynamic quorums* (intersecting sets).

Another issue stemming from network dynamism is the replica stress (load). In unstructured graphs, some nodes can be more accessed than others, unbalancing the load among replicas. Likewise, an inadequate number of replicas may increase the replica access latency since access rate directly impacts on it. Due to limitations of the local buffers size a non negligible fraction of replica accesses might be lost. However involving more participants in operation executions increase the probe-complexity. Consequently, the number of replicas should spontaneously adjust to the access rate.

Related work. Starting with Gifford's weighted votes [9], quorum systems [3, 14, 6, 25] have been widely used to provide consistency. Quorum-based approaches have been used to provide mutual exclusion [19] or shared memory emulation [5]. Recently, quorum-based implementations of atomic memory for dynamic systems have appeared [17, 7, 10]. These papers have a common design seed: they use reconfigurable quorum systems, work pioneered by Herlihy [12] for static distributed systems. In [16, 8] the authors showed that using two quorums systems concurrently preserves atomicity. This result has been later exploited in the implementation of the reconfigurable quorum systems for highly dynamic systems. Hence, periodically the system proceeds to modifications of the current quorums set (a.k.a. configuration).

In this work we follow an alternative approach for implementing atomic memory in dynamic systems. This approach is based on recent achievements in the context of dynamic quorums and p2p overlays. Dynamic quorums have been mainly investigated in [22, 2, 20]. Naor and Wieder [22] sought solutions for deterministic quorums using dynamic paths in a planar overlay [21]. Simultaneously, probabilistic quorums were proposed by Abraham and Malkhi [2] based on an overlay designed as a dynamic approximation of De Bruijn graphs [1]. Recently, in [25] the authors have discussed the impact of dynamism on the multi-dimensional quorum systems for read-few/write-many replica control protocols. They briefly describe strategies for the design of multi-dimensional quorum systems that combine local information in order to deal with frequent join and leaves of replicas and quorum sets caching in order to reduce the access latency. AndOr strategies [23] are studied in [20] in order to implement fault-tolerant storage in dynamic environment.

Contributions. In this paper we propose a modular architecture for atomic memory in dynamic systems with self-adjusting and self-healing capabilities, named *SAM* (Self-* Atomic Memory). Our approach brings together several new and old research areas exploiting the best of these worlds: p2p overlays, dynamic quorums and replica control. The architecture is composed of three interconnected modules each being designed to guarantee a specific property: (i) availability, (ii) consistency, and (iii) good load-balancing vs probe-complexity compromise.

First in order to ensure data availability despite the system dynamism, we replicate data among several nodes. Replicas of the same object define a torus overlay (similar to CAN [24] one) which has been proved efficient in the design of quorum systems [13, 23]. A specific module in our solution, $Adjuster$, serves to heal the overlay when replicas fail.

Second for the purpose of data consistency we emulate an atomic memory by the mean of intersected sets of nodes namely *quorums*. In our approach, a quorum set is sampled from a deterministic overlay traversal, encapsulated into the $Traversal$ module. To ensure atomicity—i.e. linearizability—of read/write operations we perform appropriate read and write traversal strategies. The characteristic of our approach is the use of a single round-trip communication phase for read operations. This improves the efficiency of the atomic memory when read operations are frequent compared with write operations. Moreover, when a pick of read/write requests occurs we overlap operations without breaking atomicity.

Finally, the load is balanced among replicas of the system using the *thwart* strategy encapsulated into the $LoadBalancer$ module. This strategy aims at pushing requests from loaded nodes to less solicited ones. If such nodes do not exist a potential replica is chosen within a repository of nodes.Then, the object is actively replicated at this node and this new replica is included in the memory without breaking atomicity by the Adjuster module. Conversely in response to low access rate, the memory is reduced to fit the demand while preserving reasonable probe-complexity.

To summarize, we provide an architecture for on demand atomic memory. In response to the request access rate, the atomic memory is expanded or reduced regarding to load and probe-complexity tradeoff. Dynamism involves only local participation among replicas of the topology. That is, arrivals as well as controlled or unpredictable departures are solved in a scalable manner.

The paper is organized as follows. The system model is proposed in Section 2. An overview of SAM is presented in Section 3. The three modules at the core of our architecture are detailed in Section 4. Finally, in Section 5 we conclude and present some future research topics.

2. Model

A set of processes, called *nodes*, can communicate over a network. This physical network is described by a weakly connected graph. We consider a dynamic system $\mathcal{DS}$ as the tuple $\mathcal{DS} = (I, X)$, where I is a set of possibly infinite node identifiers, and X is an unbounded universe of shared data, referred in the following as *objects*. $\mathcal{DS}$ is subject to unpredictable changes. Nodes can leave or join the system arbitrarily often. Departure includes crash failure while recovery of a failed node is treated like a new arrival. A node is called *active* if it has joined and did not fail or leave the system since then. Otherwise the node is referred as *failed*.

Each object $x \in X$ has a single owner node and is replicated at some other nodes whose identifiers $j \in J$ are such that $J \subset I$. By abuse of notation we refer to *replica* of x as a node maintaining a copy of x. A set of replica is organized as a logical overlay by an underling protocol in which connecting links are created among replicas.

Replicas of an object share the overlay, organized in a torus topology. Basically, a 2-dimensional coordinate space $[0, 1) \times [0, 1)$ is shared by the overlay replicas of an object. This is called the *memory* for object x. Observe that a single overlay can be used for different objects (e.g., x and x'), since a replica of x may be a replica of x'. We say that the bounds b of the coordinate space are given by the minimal abscissa $b.xmin = 0$, and ordinate $b.ymin = 0$ and the maximal abscissa $b.xmax = 1$ and ordinate $b.ymax = 1$. Each replica $i \in I$ has a responsibility region or *zones* in this space. Because of dynamism a node might become responsible of several zones. Each zone z is defined by $[z.xmin, z.xmax) \times [z.ymin, z.ymax)$ interval product. Two zones are *adjacent* if they are adjacent regarding to the coordinate space modulo the bounds (since the overlay can be viewed as a torus). Two replicas of the memory that are responsible for adjacent zones are referred to as *neighbors*. The communication pattern is restricted to vicinity, hence only neighbors can communicate with each other. This torus topology has been proposed in CAN [24] for a quite different purpose. Even though its vicinity property allows scalable reconfiguration in p2p systems we use this as a quorum system for atomicity while CAN uses it as a DHT.

Each node, even replicas, of the system can access any object $x \in X$ (provided it knows this object) using either a read or a write primitive. If so, we refer to them as clients. Operations consist in probing a set of nodes representing a grid row (in case of a read) or a row plus a column (in case of a write).

Here we define linearizability in terms of atomicity as given by Lynch [15].

Definition 1 (Atomicity) *If all the read/write operations that are invoked complete, then operations for object x can be partially ordered by an ordering $\prec$, satisfying the following conditions:*
• *No operation has infinitely many other operations ordered before it;*
• *The partial order is consistent with the external order, $<_{ext}$, of the invocations and responses (i.e., $<_{ext} \subseteq \prec$);*
• *All writes are totally ordered and every read is ordered with respect to all writes;*
• *If a read is preceded by a write then it returns the value of the last write; otherwise it returns the initial value.*

Replicas are accessed by clients through read/write operations on the object. Each read operation traverses the memory overlay following a horizontal trajectory that wraps all the torus. All the zones that intersect this traversal (more precisely, the replicas identified by these zones) define a *consultation quorum*. Each write operation traverses the memory following a horizontal trajectory and then a vertical one. The zones that intersect the vertical traversal (more precisely, the replicas identified by these zones) define a *propagation quorum*. Quorums are formally defined as follows:

Definition 2 (Consultation Quorum Q_c) *A consultation quorum $Q_c \subset I$ is a set of nodes whose zones Z_c are such that*
• $\bigcup_{\forall z' \in Z_c} \{[z'.xmin, z'.xmax)\} = [b.xmin, b.xmax)$
• $\bigcap_{\forall z' \in Z_c} \{[z'.xmin, z'.xmax)\} = \emptyset$
• $\exists z \in Z_c, \forall z' \in Z_c, z.ymin + (z.ymax - z.ymin/2) \in [z'.ymin, z'.ymax)$

Definition 3 (Propagation Quorum Q_p) *A propagation quorum $Q_p \subset I$ is a set of nodes whose zones Z_p are such that*
• $\bigcup_{\forall z' \in Z_p} \{[z'.ymin, z'.ymax)\} = [b.ymin, b.ymax)$
• $\bigcap_{\forall z' \in Z_p} \{[z'.ymin, z'.ymax)\} = \emptyset$
• $\exists z \in Z_p, \forall z' \in Z_p, z.xmin + (z.xmax - z.xmin/2) \in [z'.xmin, z'.xmax)$

Each read quorum intersects each write quorum. When object x is written, it is written at each replica of a writing quorum. When the value of object x is searched, all the replicas of a read quorum are queried. Note that for any consultation quorum Q_c, and propagation quorum Q_p, the following intersecting property holds: $Q_c \cap Q_p \neq \emptyset$.

3. SAM Overview

SAM emulates an atomic memory on top of a replicated system with self-adjusting and self-healing capabilities. It is specified in Input/Output Automata (IOA) Language [18, 15] and is structured as the composition of three main automata:

- $OperationManager_{x,i}$ automaton;
- $Adjuster_{x,i}$ automaton;
- $CommunicationLink_{x,i,j}$ automaton.

Where $i, j \in I$ and $x \in X$.

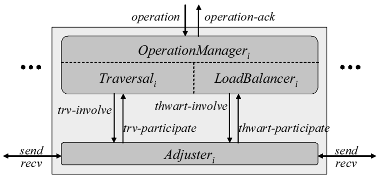

Figure 1. Overview of SAM at node i for a single object

The $OperationManager_{x,i}$ automaton has two goals. Its transitions are divided in two different sets, each serving a specific objective: *traversing* or *thwarting*. We refer to these transitions sets as respectively the $Traversal_{x,i}$ and the $LoadBalancer_{x,i}$. For the sake of simplicity, we merged those two sets of transitions in one $OperationManager_{x,i}$ automaton, since the states used by both are quite identical. Briefly, the $LoadBalancer_{x,i}$ transitions scans the memory to identify non overloaded replicas. The $Traversal_{x,i}$ automaton is in charge of maintaining the consistency of the memory by applying appropriate strategies for executing linearizable operations on the replicas.

The $Adjuster_{x,i}$ autoamton handles the expansion or shrink of the quorum system whenever requested by the $LoadBalancer_{x,i}$ automaton, and the departure of non responding replicas. The $Adjuster_{x,i}$ assigns essentially logical responsibility zones to physical replicas and maintain this correspondence consistent.

The $CommunicationLink_{x,i,j}$ simply handles the messages sent between nodes. It models the network and might lose, reorder or delay messages. In the remaining of the paper, we focus on the $Traversal_{x,i}$, the $LoadBalancer_{x,i}$, and the $Adjuster_{x,i}$. We restrict our attention to only one object x. Thus, the x subscript is omitted. Relationship among the three automata in terms of input/output actions is depicted in Figure 1.

4. SAM Architecture

The adjuster module. The adjuster automaton manages the structure of the memory. Particularly, it handles expansion of the quorum system and copes with departures of existing replicas, either due to voluntary leaves or failures.

The entrance and departure of a replica dynamically changes the decomposition of the regions. These regions are rectangles in the plane. Replicas owners of adjacent regions are called neighbors in the memory and are linked by virtual links. The memory has a torus topology in the sense that the zones over the left and right (resp. upper and lower) borders are neighbors of each other. Initially, only the owner of the object is responsible for the whole space. The bootstrapping process pushes a finite, bounded set of replicas in the network. These replicas are added to the memory using well-known strategies [24, 22]: the owner of the object specifies randomly chosen points in the memory logical overlay, and the zone in which each new replica falls is split in two. Half the zone is left to the owner of the zone, and the other half is assigned to the new replica. In the following we omit a more detailed description of the bootstrapping process.

SAM starts with a read/write request from a client. A client submits a request to one of the replicas of the memory. This replica is referred as the initiating replica. Upon receipt of a read/write request, the initiating replica does not immediately initiate a read/write traversal but it rather enqueues the request. All replicas periodically scan their queues to pick the requests for which a traversal will be initiated. The strategy to pick these requests is as follows: a write traversal is initiated only for the most recently enqueued write request (if any), while a read traversal is initiated for one of the enqueued read request (if any). Old write operations can be safely discarded (no write traversals are initiated for them) as they will not influence anymore the state of the object. There is no such constraint for a read. Once the traversals are initiated, the initiating replica empties its queue, after having kept track of all the read/write requests to later return the status of the read/write operation to the requesting clients.

Despite this request aggregation strategy allowing to reduce the requests that are actually served, the number of enqueued requests at an initiating replica can still grow very fast, incurring in a local overload. To prevent this, if the length of the queue is above a predefined threshold, then received read/write requests are forwarded to another replica that is free enough. The search of a non-overloaded replica consists in visiting the memory along a diagonal line. If such a replica is found, then it becomes the initiating replica for these requests, otherwise the size of the memory is expanded for supporting new requests. Alternatively, when the queue of a replica is empty, it leaves voluntarily the memory overlay. Thus, the size of the memory is shrunk in order to minimize the probe complexity.

When a replica leaves (voluntary or not), the zone is locally healed by relying on a strategy similar to the one proposed in CAN. Replication is actively triggered by inserting a replica within the quorum system only when it is required (i.e., for expansion purpose).

The traversal module. The traversal is in charge of maintaining the consistency of the replicas applying appropriate strategies for executing linearizable operations via consultation and propagation quorums.

In [5, 17], read/write operations are realized in two phases. The first one aims at gathering the tag and value of the last write, and the second at propagating the latest (possibly new) tag-value pair to a write quorum. The *tag* is used as a version number ordering totally the write operations. To ensure uniqueness of this tag, the node identifier of the requester is used as a low-order tie breaker. GeoQuorums [7] aims at reducing the cost of read operations by allowing their execution in only one phase (i.e., the consultation phase). This is achieved by including an additional phase in the write operation, namely the *confirmation* phase, in which the initiator of a write sends a specific confirmation message when the write operation complete.

Similar to GeoQuorums, SAM improves the efficiency of the atomic memory by maintaining a single phase read operation. The write strategy of SAM is composed of a *consultation* and a *propagation* phase, while the read strategy includes only the consultation phase. In the consultation phase, the memory is traversed from one side to another (for example west to east) and all the nodes encountered are requested for the object value and time stamp. The most up-to-date value is returned. In the propagation phase, the memory is traversed in both vertical senses. This guarantees that any write and read quorums intersect.

Propagation proceeds in both senses (north and south) so that each replica is visited twice in this phase: the first time the object is locked, preventing concurrent reads to get a stale value, while the second time the lock is released, indicating the completion of the write operation. In dynamic systems an object may be locked forever due to unforeseen leaves. We deal with this problem by assuming the use of a leasing strategy [11].

The load balancer module. The $LoadBalancer$ receives the read/write requests from clients. If the local load induced by those requests is not too high, then it triggers a traversal (i.e., activates the traversal automaton). Otherwise the load balancer automaton invokes a *thwart* process to find a suitable replica. The *thwart* process checks the memory along a diagonal trajectory until finding a non overloaded replica. If the quorum system needs to be expanded (no overloaded replica has been found), the load balancer automaton activates the adjuster automaton to add another replica to the quorum system. Finally, if for a certain amount of time, no write or read request have been locally submitted, then the load balancer automaton invokes the adjuster automaton for a shrink procedure: the local replica is removed from the memory.

Note that the $LoadBalancer$ also includes an operation aggregation strategy that can overlap operations execution. For this purpose, a timeout indicates the period between two operation-bunch treatments. Although this timeout can be almost null, its tuning depends on application type and programmer's choice.

The thwarter mechanism aims at selecting nodes and test if their load allows them to initiate the traversal. If not, the search is propagated following a *thwart path*. Let i be the *starter* of the thwart procedure. Roughly, the thwart path follows a diagonal direction from the starter. When a replica j, different from the starting node, decides to forward a thwart message, it sends it to a neighbor in the direction of a trajectory d' which starts from starter and is parallel to the diagonal d of the memory overlay rectangle. The thwart message is forwarded to the neighbor corresponding to the edge that intersects the diagonal (or the closest neighbor in the north direction). Since every replica knows its own zone limits and the bounds of the coordinate space, we only need to propagate the starting replica coordinates, to define a diagonal routing strategy that would eventually reach back the starting replica. The thwart message checks all the replicas whose zone intersects the diagonal and this guarantees that the initial zone is eventually reached again if all contacted replicas are overloaded. However, it may happen that the starter fails while the thwart is in progress, or that a new replica is inserted within the starter zone. In order to guarantee the thwart termination, the starter replica initially indicates to all its neighbors that a thwart process is started. The replica that takes over the starter's zone is aware of the thwart process and stops it if it receives the thwart message.

5. Conclusions and Future Work

In this paper we have presented SAM, an architecture for emulating an atomic memory in highly dynamic systems. SAM has self-* capabilities, self-adjusting depending on object load variations and self-healing when replicas leave the system. Our work follows a modular design inspired by theoretical and practical achievements in several research areas: p2p overlays, dynamic quorums and replica control. A major contribution of

SAM is to shrink or expand its memory achieving a compromise between load and probe-complexity. Moreover one of its more interesting characteristic comes from its distributed control and locality principles. That is, properties of SAM (atomicity, self-healing and self-adjusting) are implemented using only local information. SAM is a fundamental abstraction for many distributed complex applications, especially in dynamic systems and provides, for instance, an efficient solution to the persistent storage problem defined in [4].

We are currently working on experimental results of SAM to emphasize its behavior in face of dynamism and high-scale. Last but not least, we are convinced about the importance of mechanism theory to find incentive strategy for data replication. For instance, this would help in motivating some nodes to become replica of a specific object despite the resource consumption it involves.

References

[1] I. Abraham, B. Awerbuch, Y. Azar, Y. Bartal, D. Malkhi, and E. Pavlov. A generic scheme for building overlay networks in adversarial scenarios. In *Proc. of the 17th Int. Parallel and Distributed Processing Symposium*, 2003.

[2] I. Abraham and D. Malkhi. Probabilistic quorums for dynamic systems. In F. E. Fich, editor, *Distributed algorithms*, volume 2848/2003 of *Lecture Notes in Computer Science*, 2003.

[3] D. Agrawal and A. E. Abbadi. Efficient solution to the distributed mutual exclusion problem. In *Proc. of the 8th annual symposium on Principles of distributed computing*, 1989.

[4] E. Anceaume, R. Friedman, M. Gradinariu, and M. Roy. An architecture for dynamic scalable self-managed persistent objects. In *Proc. of Int. Symposium on Distributed Objects and Applications*, 2004.

[5] H. Attiya, A. Bar-Noy, and D. Dolev. Sharing memory robustly in message-passing systems. *Journal of the ACM*, 42(1):124–142, 1995.

[6] S. Y. Cheung, M. H. Ammar, and M. Ahamad. The grid protocol: A high performance scheme for maintaining replicated data. *IEEE Trans. Knowl. Data Eng.*, 4(6):582–592, 1992.

[7] S. Dolev, S. Gilbert, N. Lynch, A. Shvartsman, and J. Welch. GeoQuorums: Implementing atomic memory in ad hoc networks. In *Proc. of 17th International Symposium on Distributed Computing*, 2003.

[8] B. Englert and A. Shvartsman. Graceful quorum reconfiguration in a robust emulation of of shared memory. In *Proc. of Int. Conf. on Distributed Computing Systems*, 2000.

[9] D. K. Gifford. Weighted voting for replicated data. In *Proc. of the 7th ACM Symposium on Operating Systems Principles*, 1979.

[10] S. Gilbert, N. Lynch, and A. Shvartsman. Rambo II: Rapidly reconfigurable atomic memory for dynamic networks. In *Proc. of 17th Int. Conference on Dependable Systems and Networks*, 2003.

[11] C. Gray and D. Cheriton. Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. In *Proc. of the 12th ACM symposium on Operating systems principles*, 1989.

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

[13] R. Holzman, Y. Marcus, and D. Peleg. Load balancing in quorum systems. *SIAM Journal on Discrete Mathematics*, 10(2):223–245, 1997.

[14] A. Kumar. Hierarchical quorum consensus: A new algorithm for managing replicated data. *IEEE Trans. Computers*, 40(9):996–1004, 1991.

[15] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[16] N. Lynch and A. Shvartsman. Robust emulation of shared memory using dynamic quorums. In *Proc. of 27th Int. Symp. on Fault-Tolerant Computing*, 1997.

[17] N. Lynch and A. Shvartsman. RAMBO: A reconfigurable atomic memory service for dynamic networks. In *Proc. of 16th International Symposium on Distributed Computing*, 2002.

[18] N. Lynch and M. Tuttle. An introduction to input/output automata. Technical Report 3, CWI-Quaxterly.

[19] M. Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. on Computer Systems*, 3(2):145–159, 1985.

[20] U. Nadav and M. Naor. Fault-tolerant storage in a dynamic environment. In *Proc. of the 18th Annual Conference on Distributed Computing*, 2004.

[21] M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *15th ACM Symposium on Parallelism in Algorithms and Architectures*, 2003.

[22] M. Naor and U. Wieder. Scalable and dynamic quorum systems. In *Proc. of the 22th annual symposium on Principles of Distributed Computing*, 2003.

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

[24] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of the ACM SIGCOMM*, 2001.

[25] B. Silaghi, P. Keleher, and B. Bhattacharjee. Multi-dimensional quorum sets for read-few write-many replica control protocols. In *In Proc. of the 4th CCGRID/GP2PC*, 2004.