12. Communication and Data Sharing for Dynamic Distributed Systems*

Nancy Lynch[1] and Alex Shvartsman[2,1]

[1] Laboratory for Computer Science, Massachusetts Institute of Technology, 200 Technology Square, NE43-365, Cambridge, MA 02139, USA
[2] Department of Computer Science and Engineering, University of Connecticut, 191 Auditorium Road, Unit 3155, Storrs, CT 06269, USA

12.1 Introduction

This research direction aims to develop and analyze algorithms to solve problems of communication and data sharing in highly dynamic distributed environments. The term *dynamic* here encompasses many types of changes, including changing network topology, processor mobility, changing sets of participating client processes, a wide range of types of processor and network failures, and timing variations. Constructing distributed applications for such environments is a difficult programming problem. In practice, considerable effort is required to make applications resilient to changes in client requirements and to evolution of the underlying computing medium. We focus our work on distributed services that provide useful guarantees and that make the construction of sophisticated distributed applications easier. The properties we study include ordering and reliability guarantees for communication and coherence guarantees for data sharing. To describe inherent limitations on what problems can be solved, and at what cost, the algorithmic results will be accompanied by lower bound and impossibility results.

One example of our approach is the new dynamic atomic shared-memory service for message-passing systems. We formally specified the service and developed algorithms implementing the service. A system implementation is under development. The service is reconfigurable in the sense that the set of owners of data can be changed dynamically and concurrently with the ongoing read and write operations. We proved the correctness of the implementation for arbitrary patterns of asynchrony and crashes, and we analyzed its performance conditioned on assumptions about timing and failures.

12.2 Recent Directions

In recent years, we worked on distributed algorithms and their analysis for a wide range of problems including maintenance of replicated data, e.g., [12.6, 12.7] and view-oriented group membership and communication, e.g., [12.5, 12.8]. We have also carried out implementation studies, e.g., [12.3, 12.13]. Several of our projects were inspired by middleware used in commercial and academic systems, most notably, by *group communication systems* [12.4]. Additional motivation comes from dynamic voting systems,

* This work is supported by the NSF Grants 0121277, 9988304 and 9984774.

e.g., [12.15], protocols for atomic data, e.g., [12.1, 12.10], and dealing with unbounded number of processors, e.g., [12.14].

Many of our algorithms are designed to cope with timing anomalies and some forms of processor and communication failure, and some handled explicit requests to reconfigure the system. In carrying out this work, we found it useful to formulate problems and decompose solutions in terms of precisely-defined *global services*. Most of our work, and other work on fault-tolerant distributed computing, makes it clear that algorithms for such a setting can be extremely complex. The use of abstract global services with well-defined interfaces and behavior to decompose the algorithms helps considerably in reducing this complexity. For example, consensus algorithms have been used as building blocks in other work [12.9]. Additionally, such decomposition is useful in analyzing correctness of algorithms and their performance.

12.3 New Directions: Dynamic Distributed Systems

Our new directions target communication and data sharing problems in highly dynamic distributed environments. The environments we consider will be (even) less well-behaved than the ones we considered earlier, including, for example, unknown universe of processors, explicit requests by participants to join and leave the system, and mobility. We aim for a coherent theory rather than isolated algorithmic results, and we look for common services that can be used as parts of many algorithms, and for lower bound results as well as upper bound (algorithmic) results. We are considering network environments in which the set of processors and their connectivity changes over time. Processors and links may be added and removed from a network, and while they are in the network, they may fail and recover. Different processors and communication links may operate at drastically different speeds, and these speeds may vary over time. Processors may be connected wirelessly and be mobile, moving about in space while they communicate. Application processes may also migrate around the network. On such substrates, we will consider running distributed applications involving identified groups of participants. These will include sharing of files in wide-area networks, distributed multi-player games, computer-supported cooperative work, maintaining and disseminating information about real-world, real-time endeavors with strict data-consistency requirements (such as military operations), multimedia transmission, and others.

Approach. We view the communication and data-sharing problems to be solved as high level *global services*, which span network locations. These services generally will provide performance and fault-tolerance guarantees, conditioned on assumptions about the behavior of the environment and of the underlying network substrate.

Traditionally, research on distributed computing primitives and services has emphasized specification and correctness, while research on distributed and parallel algorithms has emphasized efficiency and performance. Our approach will combine and synthesize these two concerns: It will yield algorithms that perform efficiently and degrade gracefully in dynamic distributed systems, and whose correctness, performance, and fault-tolerance guarantees are expressed by precisely-defined global services. We will include the study of trade-offs involving service guarantees and performance. For example, achieving atomicity is expensive, and there are reasons to believe that weakening consistency slightly may reduce the cost and still provide useful semantics.

Because the setting is very complex, the algorithms will also be very complex, which means that it is necessary to decompose them into smaller, more manageable pieces. In our research, many of those smaller pieces will be viewed as lower-level, *auxiliary global services*. These services will provide lower-level communication and data-sharing capabilities, plus other capabilities such as failure detection, progress detection, consensus, group membership, leader election, reconfiguration, resource allocation, workload distribution, location determination, and routing. These services must also include conditional performance and fault-tolerance guarantees. This decomposition can be repeated any number of times, at lower levels of abstraction. The work we pursue in attaining our goals includes:

- Defining new global services to support computing in complex distributed environments, with particular emphasis on communication and data-sharing services.
- Developing and analyzing algorithms that implement these services in dynamic systems, and algorithms that use the services to implement higher-level services.
- Obtaining corresponding lower bounds and impossibility results.

This work is carried out in terms of a mathematical framework based on interacting state machines. The state machines will include features to express issues of timing, continuous behavior, and probabilistic behavior. Supporting metatheory, including general models, performance measures, and proof and performance analysis methods, will also be developed. Our theoretical work complements ongoing work on implementation and testing of distributed system services. Parts of this work are guided by examples chosen from prototype applications, including distributed file management, information collection and dissemination, computer-supported cooperative work, and distributed games. When developing specifications motivated by existing systems we also rely on information from the developers about what their services guarantee.

Intended impact. Our research will contribute to developing the theory of communication and data sharing in dynamic distributed systems. In terms of practical implications, our research has the potential to produce qualitative improvements in capabilities for constructing applications for dynamic distributed environments. New global services can be used to decompose the task of constructing complex into manageable subtasks. Integrating conditional performance and fault-tolerance guarantees into service specifications will decompose the task of analyzing the performance and fault-tolerance of complex systems, which in turn will make this analysis more tractable. Lower bound and impossibility results will tell system designers when further effort would be futile.

12.4 Reconfigurable Atomic Memory Service

We now present an example of our new work on algorithms for dynamic systems. We overview our algorithm [12.12] that can be used to implement atomic read/write shared memory in a dynamic network setting, in which participants may join or fail during the course of computation. Examples of such settings are mobile networks and peer-to-peer networks. One use of this service might be to provide long-lived data in a dynamic and volatile setting such as a military operation.

We developed a formal specification for a reconfigurable atomic shared memory as a global service. We call this service RAMBO, which stands for "Reconfigurable Atomic Memory for Basic Objects" ("Basic" means "Read/Write"). Then we provided a dynamic distributed algorithm that implements this service. In order to achieve availability in the presence of failures, the objects are replicated. In order to maintain memory consistency in the presence of small and transient changes, the algorithm uses *configurations*, each of which consists of a set of *members* plus sets of *read-quorums* and *write-quorums*. In order to accommodate larger and more permanent changes, the algorithm supports *reconfiguration*, by which the set of members and the sets of quorums are modified. Such changes do not cause violations of atomicity. Any quorum configuration may be installed at any time.

The algorithm carries out three major activities, all concurrently: reading and writing objects, introducing new configurations, and removing ("garbage-collecting") obsolete configurations. The algorithm is composed of a *main algorithm*, which handles reading, writing, and garbage-collection, and a global reconfiguration service, *Recon*, which provides the main algorithm with a consistent sequence of configurations. Reconfiguration is loosely coupled to the main algorithm, in particular, several configurations may be known at one time, and read and write operations can use them all.

The main algorithm performs read and write operations using a two-phase strategy. The first phase gathers information from read-quorums of active configurations and the second phase propagates information to write-quorums of active configurations. This communication is carried out using background gossiping, which allows the algorithm to maintain only a small amount of protocol state information. Each phase is terminated by a *fixed point* condition that involves a quorum from each active configuration. Different read and write operations may execute concurrently: the restricted semantics of reads and writes permit the effects of this concurrency to be sorted out later. A facility is provided for *garbage-collecting* old configurations when their use is no longer necessary for maintaining consistency.

The reconfiguration service is implemented by a distributed algorithm that uses consensus to agree on the successive configurations. Any member of the latest configuration c may propose a new configuration at any time; different proposals are reconciled by an execution of consensus among the members of c. Consensus is, in turn, implemented using a version of the Paxos algorithm [12.11]. Although such consensus executions may be slow—in fact, in some situations, they may not even terminate—they do not delay read and write operations. Garbage-collection uses a two-phase strategy, where the first phase communicates with an old configuration c and the second phase communicates with a new configuration c'. A garbage-collection operation ensures that both a read-quorum and a write-quorum of c learn about c', and that the latest object value from c is conveyed to a write-quorum of c'.

We show atomicity for arbitrary patterns of asynchrony and failure. We analyze performance *conditionally*, based on timing and failure assumptions. For example, assuming that gossip and garbage-collection occur periodically, that reconfiguration is requested infrequently enough for garbage-collection to keep up, and that quorums of active configurations do not fail, we show that read and write operations complete within time $8d$, where d is the maximum message latency.

A complete distributed system implementation is also underway.

12.5 Closing Remarks

Our approach to middleware (of which RAMBO is an example) differs from common practice: although middleware frameworks such as CORBA, DCE and Java/JINI support construction of distributed systems from components, their specification capability is limited to the formal definition of interfaces and informal descriptions of behavior. These are not enough to support careful reasoning about the behavior of systems that are built using such services. Moreover, current middleware provides only rudimentary support for fault-tolerance. In contrast, our services are precisely defined, with respect to both their interfaces and their behavior. The specified behavior may include performance and fault-tolerance. Our component behavior is specified in a compositional way, so that correctness, performance, and fault-tolerance properties of a system can be inferred from corresponding properties of the system's components.

Our project will, we believe, contribute substantially toward a coherent theory of algorithm design and complexity analysis for dynamic distributed environments, as powerful as the theory that currently exists for static distributed systems. The contributions of this project will be mainly theoretical. However, the resulting services and algorithms will also have the potential for impact on design of real systems for dynamic environments. Note that actually incorporating theoretical services like ours into systems will require additional work of another sort: software engineering work to integrate them with other system components built using object-oriented and component technologies (Birman discusses some of these issues in [12.2]).

References

12.1 H. Attiya, A. Bar-Noy and D. Dolev, "Sharing Memory Robustly in Message Passing Systems", *J. of the ACM*, vol. 42, no. 1, pp. 124-142, 1996.

12.2 Kenneth P. Birman. A review of experiences with reliable multicast. *Software, Practice and Experience*, 29(9):741–774, September 1999.

12.3 O. Cheiner and A. Shvartsman, "Implementing an eventually-serializable data service as a distributed system building block," in *Networks in Distributed Computing*, vol. 45, pp. 43–72, AMS.

12.4 *Communications of the ACM*, special section on group communications, vol. 39, no. 4, 1996.

12.5 R. De Prisco, A. Fekete, N. Lynch, and A. Shvartsman, "A dynamic primary configuration group communication service," in *Distributed Computing* Proceedings of DISC'99 - 13th International Symposium on Distributed Computing, 1999, LNCS, vol. 1693, pp. 64–78.

12.6 B. Englert and A. Shvartsman. Graceful quorum reconfiguration in a robust emulation of shared memory. In *Proc. of the 20th IEEE Int-l Conference on Distributed Computing Systems (ICDCS'2000)*, pp. 454-463, 2000.

12.7 A. Fekete, D. Gupta, V. Luchangco, N. Lynch, and A. Shvartsman, "Eventually-serializable data service," *Theoretical Computer Science*, vol. 220, no. 1, pp. 113–156, June 1999.

12.8 A. Fekete, N. Lynch, and A. Shvartsman. “Specifying and using a partitionable group communication service." *ACM Trans. on Computer Systems*. vol. 19, no. 2, pp. 171-216, 2001.

12.9 R. Guerraoui and A. Schiper, “The Generic Consensus Service," *IEEE Trans. on Software Engineering*, Vol. 27, No. 1, pp. 29-41, January, 2001.

12.10 S. Haldar and P. Vitányi, “Bounded Concurrent Timestamp Systems Using Vector Clocks", *J. of the ACM*,Vol. 249, No. 1, pp. 101-126, January, 2002

12.11 Leslie Lamport, "The Part-Time Parliament", *ACM Transactions on Computer Systems*, 16(2) 133-169, 1998.

12.12 N. Lynch and A. Shvartsman, “RAMBO: A Reconfigurable Atomic Memory Service", in *Proc. of 16th Int-l Symposium on Distributed Computing, DISC'2002*, pp. 173-190, 2002.

12.13 K. W. Ingols, “Availability study of dynamic voting algorithms,” M.S. thesis, Dept. of Electrical Engineering and Computer Science, MIT, May 2000.

12.14 M. Merritt and G. Taubenfeld, “Computing with infinitely many processes (under assumptions on concurrency and participation),” In *Proc.14th International Symposium on DIStributed Computing (DISC)*, October 2000.

12.15 E. Yeger Lotem, I. Keidar, and D. Dolev. “Dynamic voting for consistent primary components." In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 63–71, August 1997.