Communication and Data Sharing for Dynamic Distributed Systems [1]

Nancy Lynch [2] *and Alex Shvartsman* [3]

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. The algorithmic results will be accompanied by lower bound and impossibility results, which describe inherent limitations on what problems can be solved, and at what cost. One example of our approach is a new dynamic atomic shared-memory service for message-passing systems. We specified the new reconfigurable read/write service and developed algorithms implementing the service. 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 we analyzed its performance under a variety of assumptions about timing and failures.

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 services* has emphasized specification and correctness, while research on *distributed algorithms* has emphasized complexity 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.

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. Where needed, supporting metatheory, including general models, performance measures, and proof and performance analysis methods, will also be developed.

The theoretical work in this project complements ongoing work on implementation and testing of distributed system services. Parts of our work will be guided by examples chosen from several prototype applications, including distributed file management, information collection and dissemination, computer-supported cooperative work, distributed games, and multimedia transmission. When developing specifications motivated by existing implementations we also rely on information from the developers about what their services guarantee.

[1] This work is supported by the NSF ITR Grant 0121277, and NSF Grants 9988304 and 9984774.

[2] Laboratory for Computer Science, Massachusetts Institute of Technology, 200 Technology Square, NE43-365, Cambridge, MA 02139, USA. Email: lynch@theory.lcs.mit.edu.

[3] Department of Computer Science and Engineering, University of Connecticut, 191 Auditorium Road, Unit 3155, Storrs, CT 06269, USA. Email: aas@cse.uconn.edu.

Potential impact of the research: 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 distributed systems 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.

Recent work and new directions: In recent years, we have worked on distributed algorithms and their analysis, for a wide variety of individual problems including maintenance of replicated data [2, 3, 4], view-oriented group membership and group communication, [5, 6, 7], analysis of counting networks [8] and computational workload balancing [9, 10, 11, 12, 13]. We have also performed several simulation and implementation studies, e.g., [14, 15]. Several of the services we used were inspired by middleware used in commercial and academic systems, most notably, by *group communication systems* providing multicast services to named groups of client processes [16].

Most of our algorithms are designed to cope with timing anomalies and some forms of processor and communication failure, and one 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 helped considerably in reducing this complexity. This decomposition was useful not only in designing the algorithms, but also in analyzing their correctness and performance.

Our new research directions target communication and data sharing problems in highly dynamic distributed environments. That is, 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. Thus, 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 change 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. In fact, we consider failures and recoveries, both temporary and permanent, to be the norm rather than the exception. Our scope includes a range of possible types of failures, including Byzantine failure of processors and crashes with loss of volatile memory. Different processors and different communication links may operate at drastically different speeds, and even the same processor or communication link may exhibit highly variable speeds over time. Processors may also be connected wirelessly and may be mobile, moving about in space while they communicate with nearby mobile and/or stationary neighbors. Application processes may also migrate around the network. On such substrates, we will consider running distributed applications involving identified groups of participants ("group-oriented applications"). These will include, for example, 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), and multimedia transmission.

Reconfigurable atomic memory service: We now present an example of our new work on algorithms for dynamic systems. We overview our algorithm [17] that implements atomic shared memory and that is designed for highly dynamic settings, in which participants may join, leave, or fail during the course of computation. Examples of such settings are mobile networks or peer-to-peer networks where data survivability is of high concern. One use of this service might be to provide survivable data in a dynamic and volatile setting such as a military operation. In order to achieve availability in the presence of failures, the algorithm replicates the memory objects. In order to maintain memory consistency in the presence of small and transient changes, the algorithm uses *configurations* consisting of sets of read and write quorums. In order to accommodate larger and more permanent changes, the algorithm supports *reconfiguration*, by which the set of owners of the data, and the sets of read and write quorums, are modified.

We first provide a formal specification for a reconfigurable version of atomic shared memory as a global service, which we call RAMBO: Reconfigurable Atomic Memory for Basic Objects. Then we present our algorithm. The algorithm uses a reconfiguration service, which we call $Recon$, and which provides the main algorithm with a consistent sequence of configurations. Our implementation of the reconfiguration service uses a sequence of consensus instances, one for each new configuration. Reconfiguration is loosely-coupled to the main read-write algorithm, in particular, several configurations may be known to the algorithm, and the read and write operations can use them all without any harm. Note that consensus is used infrequently and its termination affects only reconfigurations — termination of read and write operation does not depend on whether or not consensus terminates, nor on the time it takes it to terminate.

The main algorithm performs read and write operations requested by clients using a two-phase strategy where the first phase gathers information from active configurations and the second phase propagates information to the active configurations. Each phase constructs a "fixed point" that involves a quorum from each active configuration. The algorithm keeps little "protocol state", instead relying on background gossiping. 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 on. Our main algorithm includes a facility for carefully "garbage-collecting" old configurations when their use is no longer necessary for maintaining consistency. This is the key part. It involves communicating with quorums of old and new configurations, so that old configuration learns about the new configuration, and the latest value from the old configuration is conveyed to the new configuration.

Our algorithm includes the following innovations: *(1) Dynamic owners of data*: Any and all owners may request reconfiguration and the set of owners can be changed dynamically through reconfiguration. *(2) Dynamic configurations*: *Arbitrary* configurations can be installed, and *no* requirements are imposed on intersection of quorum sets or member sets in distinct configurations. *(3) Loosely-coupled reconfiguration*: The clients of the service continue issuing read/write requests even if reconfiguration is in progress. If there is a finite number of reconfigurations, then read/write operations will complete under reasonable assumptions about failures whether or not any reconfigurations complete. *(4) Efficient "steady-state"*: Assuming bounded message delays, infrequent reconfigurations, and periodic gossip and garbage-collection, reads and writes complete in time constant times the message delay. *(5) Fast "catch-up"*: Clients with out-of-date configurations can catch up with new configurations after a logarithmic number of configuration accesses provided such configurations are not disabled.

Our algorithms are specified using Input/Output Automata, a language for specifying interacting state machines. The overall system is defined as the composition of the constituent components. This allows both for a modular implementation, and compositional reasoning about the system's properties. The main safety property, atomicity of the data, is shown using a combination of invariant assertions and partial-order methods. We show atomicity for arbitrary patterns of asynchrony. We assess performance of the service under certain failure and timing assumptions. In analyzing performance, we assume that certain quorums do not fail and that new configurations are not produced too quickly for the garbage-collection to keep up. Note that gossip has no impact on safety, but periodic gossip improves the performance of reads and writes and the fault-tolerance of the overall system. We are currently developing a complete portfolio of performance results conditioned on different assumptions about the failures and delays. The performance evaluations simply consider collections of safe executions and no additional safety proofs are needed. A system implementation study is also under way.

Closing remarks: Our approach to middleware 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.

Out project will, we believe, contribute substantial progress 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 services and algorithms defined 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 standard object-oriented and component technologies (Birman discusses some of these issues in [1]).

References:

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

[2] 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, Special Issue on Distributed Algorithms.

[3] N. Lynch and A. Shvartsman, "Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts," in *Twenty-Seventh Annual International Symposium on Fault-Tolerant Computing (FTCS'97)*, Seattle, Washington, USA, June 1997, IEEE, pp. 272–281.

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

[5] Alan Fekete, Nancy Lynch, and Alex Shvartsman. Specifying and using a partitionable group communication service. *ACM Transactions on Computer Systems*. vol. 19, no. 2, pp. 171-216, May, 2001.

[6] R. DePrisco, A. Fekete, N. Lynch, and A. Shvartsman, "A dynamic view-oriented group communication service," in *Proceedings of the 17th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Puerto Vallarta, Mexico, June-July 1998, pp. 227–236, Also, technical memo in progress.

[7] 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, Bratislava, Slovak Republic, September 1999, P. Jayanti, Ed., Bratislava, Slovak Republic, 1999, vol. 1693 of *Lecture Notes in Computer Science*, pp. 64–78, Springer-Verlag-Heidelberg.

[8] N. Lynch, N. Shavit, A. Shvartsman, and D. Touitou, "Timing conditions for linearizability in uniform counting networks," *Theoretical Computer Science*, vol. 220, no. 1, pp. 67–91, June 1999, Special Issue on Distributed Algorithms.

[9] S. Dolev, R. Segala, and A. Shvartsman, "Dynamic load balancing with group communication," in *6th International Colloquium on Structural Information and Communication Complexity (SIROCCO'99)*, pp. 111–125. See also Technical Memo MIT/LCS/TM-588, Lab. for Computer Science, Massachusetts Institute of Technology, 1998.

[10] C. Georgiou and A. Shvartsman. Cooperative computing with fragmentable and mergeable groups. In *Proc. of 7th International Colloquium on Structure of Information and Communication Complexity SIROCCO'00*, pp. 141–156, 2000.

[11] C. Georgiou, A. Russell and A. Shvartsman. The Complexity of Synchronous Iterative Do-All with Crashes, in *Proc. of 15th International Symposium on Distributed Computing (DISC'01)*, pp. 151–165, 2001.

[12] R. Khazan, A. Fekete, and N. Lynch, "Multicast group communication as a base for a load-balancing replicated data service," in *12th International Symposium on Distributed Computing*, Andros, Greece, pp. 258–272, 1998.

[13] G. Malewicz, A. Russell, and A. Shvartsman. Distributed cooperation during the absence of communication. In *Proc. of 14th International Symposium on Distributed Computing (DISC'00)*, pp. 119–133, 2000.

[14] O. Cheiner and A. Shvartsman, "Implementing an eventually-serializable data service as a distributed system building block," in *Networks in Distributed Computing*, M. Mavronicolas, M. Merritt, and N. Shavit, Eds. 1999, vol. 45 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 43–72, AMS.

[15] K. W. Ingols, "Availability study of dynamic voting algorithms," M.S. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Insitute of Technology, May 2000.

[16] *Communications of the ACM 39(4), special issue on Group Communications Systems*, April 1996.

[17] Nancy Lynch and Alex Shvartsman. "Reconfigurable Atomic Read/Write Shared Memory", manuscript, 2002.