Sameer Ajmani
Statement of Research Interests
January 5, 2004

I am interested in improving the robustness and security of distributed systems, especially by creating better programming models and runtime environments for building and maintaining such systems.

Past and Current Research

My research to date has included distributed algorithms, machine learning, security, and systems. I will highlight four important research projects that I have been conducting in the last few years: a trusted execution service, a library for certificate management, a peer-to-peer certificate distribution system, and a software upgrade infrastructure for distributed systems.

For my master's research, I designed and implemented a service that enables mutually-distrusting parties to safely share private data in a computation by hosting it on a trusted execution platform [2, 5]. The service uses static analysis to ensure that a computation cannot leak information except via designated channels. These channels are labeled with symbolic names for the participants in the computation (e.g., "auctioneer" and "bidder"), and the service authenticates these channels by translating their labels into public keys via the SDSI public key infrastructure. Evaluation of applications running on the service, such as tax preparation and auction software, showed that the programming model is flexible and that performance is dominated by the time required to do authentication. By bridging the gap between the programming model and the runtime environment, the trusted execution service helps developers avoid unintentional information leaks.

When I was developing the trusted execution service, Java support for SDSI was weak, so I created a new library called JSDSI [1] that implements standard Java Security APIs for certificate management. JSDSI includes new algorithms that I developed for discovering SDSI certificate chains in distributed systems. JSDSI is publicly available on SourceForge and is used by researchers worldwide.

One of the challenges in using any public key infrastructure is locating the required certificates efficiently, since they may be distributed across nodes in the network. To address this challenge, I developed ConChord, a peer-to-peer SDSI certificate storage system [3]. ConChord simplifies the process of locating the information needed to make authentication decisions by allowing cooperating nodes to maintain an index over a very large set of certificates. Evaluation showed that ConChord balances load effectively and reduces the latency of certificate discovery over non-cooperative designs.

For my PhD research, I am developing an infrastructure called Upstart whose purpose is to automate the process of upgrading distributed systems [4]. Long-lived systems need software upgrades to fix bugs, add features, and improve performance. These systems must tolerate node failures and recoveries, so we model a node upgrade as a node restart. But this can lead to problems: if too many nodes upgrade at once, then the system as a whole may fail. But if node upgrades are spread out over time, then nodes running different versions may need to communicate, and they may not understand one another.

We solve these problems by delegating two tasks to the developer. First, the developer defines a schedule for when nodes should upgrade, e.g., "upgrade replicas round robin" or "upgrade servers before clients." We simplify this task by providing a library of common scheduling functions and by enabling the developer to monitor and control upgrade progress.

Second, the developer defines adapters that enable communication between nodes running different versions. The developer only needs to define adapters for the current and new versions; communication with nodes running older versions is handled automatically via chains of adapters. We simplify this task further by providing a program that generates skeleton code for adapters. Once the developer has filled in

the details, Upstart automatically disseminates and installs the upgrade. Nodes verify the authenticity of an upgrade before acting on it, so malicious parties cannot corrupt a system via the upgrade infrastructure.

Understanding the dependences between nodes is vital to making an upgrade run smoothly. For example, an upgrade schedule must limit the number of service replicas that upgrade simultaneously, or else nodes that depend on that service may fail. An adapter must implement the specification expected by the clients of the node, or else the assumptions made by the clients may be violated. We help programmers understand these issues by defining criteria for good scheduling functions and adapters, drawing on behavioral subtyping theory and abstract data models as needed.

I am evaluating a prototype of Upstart on several upgrade scenarios. Initial results are promising: the upgrade infrastructure has low overhead and successfully disseminates and installs new software and adapters. I plan to explore a variety of enhancements and to evaluate a large-scale deployment on PlanetLab.

Future Directions

Short term, I am interested in doing further work on software upgrades. I would like to integrate the upgrade infrastructure with a distributed component framework like J2EE or .NET so that systems developed in the framework can be upgraded easily. An interesting question is how to deal with upgrades that affect not whole classes of nodes (like "all servers") but rather subobjects within nodes or subsystems that span nodes. This can be done by extending the upgrade infrastructure to work at multiple levels within a system, but careful design is needed to avoid introducing additional overhead to handle this generalization. I am eager to investigate this and other areas of this work.

Long term, I want to improve how people build and maintain distributed systems, especially by creating better programming models and runtime environments. Software attestation, which makes it possible to verify what software is running on a remote node, enables intriguing new possibilities in system management. I envision a widespread runtime infrastructure that supports multiple, possibly mutually-distrusting, distributed systems on the same set of physical nodes. These systems can verify the infrastructure via attestation, share data safely via mutually-trusted computations, and reap the benefits of a common infrastructure for monitoring and control. Systems like PlanetLab and Emulab are steps toward this vision, but much remains to be done, especially as the physical infrastructure is extended to include networks of small devices.

There are plenty of unanswered questions in what I have described here, and I believe the answers will be interesting to researchers and industry developers alike. My work has always benefited from collaboration with peers, and I look forward working with others whose goals are as ambitious as my own.

References

[1] Sameer Ajmani. JSDSI: A Java SPKI/SDSI implementation. `http://jsdsi.sourceforge.net`.

[2] Sameer Ajmani. A trusted execution platform for multiparty computation. Master's thesis, MIT, September 2000. Also available as MIT technical report MIT-LCS-TR-846.

[3] Sameer Ajmani, Dwaine E. Clarke, Chuang-Hue Moh, and Steven Richman. ConChord: Cooperative SDSI certificate storage and name resolution. In *First International Workshop on Peer-to-Peer Systems, (IPTPS)*, number 2429 in Lecture Notes in Computer Science, pages 141–154, March 2002.

[4] Sameer Ajmani, Barbara Liskov, and Liuba Shrira. Scheduling and simulation: How to upgrade distributed systems. In *Ninth Workshop on Hot Topics in Operating Systems (HotOS-IX)*, May 2003.

[5] Sameer Ajmani, Robert Morris, and Barbara Liskov. A trusted third-party computation service. Technical Report MIT-LCS-TR-847, MIT, May 2001.