Automatic Software Upgrades for Distributed Systems

Sameer Ajmani
ajmani@csail.mit.edu

Barbara Liskov
liskov@csail.mit.edu

Liuba Shrira
liuba@cs.brandeis.edu

Long-lived distributed systems, such as server clusters, content distribution networks, peer-to-peer systems, and sensor networks, require changes (upgrades) to their software over time to fix bugs, add features, and improve performance. These systems are large, so it is impractical for an administrator to upgrade nodes manually (e.g., via remote login). Instead, upgrades must propagate automatically, but the administrator may still require control over the order and rate at which nodes upgrade to avoid interrupting service or to test an upgrade on a few nodes. Upgrades may happen slowly, so there may be long periods of time when some nodes are upgraded and others are not. Nonetheless, the system as a whole should continue to provide service.

The goal of our research is to support automatic upgrades for such systems and to enable them to provide service during upgrades. Earlier approaches to automatically upgrading distributed systems [8–10, 15, 16, 19] or distributing software over networks [1–7, 14, 23] do little to ensure continuous service during upgrades. The Eternal system [25], the Simplex architecture [22], and Google [13] enable specific kinds of systems to provide service during upgrades, but they do not provide general solutions.

An automatic upgrade system must:

- propagate upgrades to nodes automatically
- provide a way to control *when* nodes upgrade
- enable the system to provide service when nodes are running different versions
- provide a way to preserve the persistent state of nodes from one version to the next

To address these requirements, our approach includes an *upgrade infrastructure*, *scheduling functions*, *simulation objects*, and *transform functions*.

The *upgrade infrastructure* enables rapid dissemination of upgrade information and flexible monitoring and control of upgrade progress. The infrastructure consists of four kinds of components: an *upgrade server* that publishes new system upgrades; an *upgrade database* that provides a shared whiteboard for upgrade coordination and monitoring; per-node *upgrade managers* that install new upgrades on nodes; and per-node *upgrade layers* that enable nodes running different versions to communicate.

Upgrade managers run procedures called *scheduling functions* (SFs) to determine when their nodes should upgrade. Different SFs enable different scheduling policies, e.g., one SF can implement rapid (though disruptive) dissemination of critical updates, while another can upgrade nodes gradually to minimize service disruption.

Upgrade layers run adapters called *simulation objects* (SOs) to enable a node to behave as though it were running multiple versions simultaneously. Unlike previous approaches that propose similar adapters [12, 21, 25], ours includes correctness criteria to ensure that simulation objects reflect node state consistently across different versions. These criteria require that some interactions made via SOs must fail; we identify when such failure is necessary and, conversely, when it is possible to provide service between nodes running different versions.

Transform functions (TFs) are procedures that convert a node's persistent state from one version to the next. Our contribution is to show how TFs interact with SOs to ensure that nodes upgrade to the correct new state. This enables more powerful forms of cross-version compatibility than have been proposed in earlier work.

Our approach takes advantage of the fact that long-lived systems are *robust*. They tolerate node failures: nodes are prepared for failures and know how to recover to a consistent state. This means that we can model a node upgrade as a soft restart. Robust systems also tolerate communication problems: remote procedure calls may fail, and callers know how to compensate for such failures. This means that we can use a failure response when calls occur at inopportune times, e.g., when a node is upgrading or when a node's simulation object is unable to carry out the requested action. We want to minimize such failures; our correctness criteria define when this is possible.

We are implementing a prototype of our upgrade infrastructure in C++. We have implemented the upgrade manager and upgrade layer as a session-layer handler using the TESLA toolkit [20]. This enables us to add the upgrade layer to unmodified legacy applications. We have implemented the upgrade server and upgrade database as an SFS file system [18]. The enables nodes to access new upgrades securely over the network via a local file system interface. Over the next few months, we will be evaluating several upgrade scenarios, including upgrades to Thor [17], Chord [24], NFS [11], and SFS [18].

References

[1] APT HOWTO. http://www.debian.org/doc/manuals/apt-howto/.

[2] Cisco Resource Manager. http://www.cisco.com/warp/public/cc/pd/wr2k/rsmn/.

[3] EMC OnCourse. http://www.emc.com/products/software/oncourse.jsp.

[4] Marímba. http://www.marimba.com/.

[5] Red Hat up2date. http://www.redhat.com/docs/manuals/RHNetwork/ref-guide/up2date.html.

[6] Rsync. http://www.rsync.org/.

[7] Managing automatic updating and download technologies in Windows XP. http://www.microsoft.com/WindowsXP/pro/techinfo/administration/manageau%toupdate/default.asp, 2002.

[8] Joao Paulo A. Almeida, Maarten Wegdam, Marten van Sinderen, and Lambert Nieuwenhuis. Transparent dynamic reconfiguration for CORBA, 2001.

[9] C. Bidan, V. Issarny, T. Saridakis, and A. Zarras. A dynamic reconfiguration service for CORBA. In *4th Intl. Conf. on Configurable Dist. Systems*, 1998.

[10] Toby Bloom. *Dynamic Module Replacement in a Distributed Programming System*. PhD thesis, MIT, 1983.

[11] B. Callaghan, B. Pawlowski, and P. Staubach. NFS version 3 protocol specification. RFC 1813, Network Working Group, 1995.

[12] Ophir Frieder and Mark E. Segal. On dynamically updating a computer program: From concept to prototype. *Journal of Systems and Software*, 1991.

[13] Sanjay Ghemawat. Google, inc., personal comm., 2002.

[14] Richard S. Hall et al. An architecture for post-development configuration management in a wide-area network. In *Intl. Conf. on Dist. Computing Systems*, 1997.

[15] Christine R. Hofmeister and James M. Purtilo. A framework for dynamic reconfiguration of distributed programs. Technical Report CS-TR-3119, University of Maryland, College Park, 1993.

[16] J. Kramer and J. Magee. The evolving philosophers problem: Dynamic change management. *IEEE Transactions on Software Engineering*, 16(11), 1990.

[17] Barbara Liskov, Miguel Castro, Liuba Shrira, and Atul Adya. Providing persistent objects in distributed systems. In *European Conf. on Object-Oriented Programming*, 1999.

[18] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP '99)*, pages 124–139, Kiawah Island, South Carolina, December 1999.

[19] Tobias Ritzau and Jesper Andersson. Dynamic deployment of java applications. In *Java for Embedded Systems Workshop*, 2000.

[20] Jon Salz, Alex C. Snoeren, and Hari Balakrishnan. TESLA: A transparent, extensible session-layer architecture for end-to-end network services. In *Proc. of the Fourth USENIX Symposium on Internet Technologies and Systems (USITS)*, March 2003.

[21] Twittie Senivongse. Enabling flexible cross-version interoperability for distributed services. In *Intl. Symposium on Dist. Objects and Applications*, 1999.

[22] Lui Sha, Ragunathan Rajkuman, and Michael Gagliardi. Evolving dependable real-time systems. Technical Report CMS/SEI-95-TR-005, CMU, 1995.

[23] Michael E. Shaddock, Michael C. Mitchell, and Helen E. Harrison. How to upgrade 1500 workstations on Saturday, and still have time to mow the yard on Sunday. In *Proc. of the 9th USENIX Sys. Admin. Conf.*, 1995.

[24] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, August 2001.

[25] L. A. Tewksbury, L. E. Moser, and P. M. Melliar-Smith. Live upgrades of CORBA applications using object replication. In *IEEE Intl. Conf. on Software Maintenance (ICSM)*, 2001.