Guest Editorial

Eugene H. Spafford Purdue University

Introduction

Consider some of the qualities we would like our computer systems to have. We would like them to be fast, have great capacity, be highly available, resist failures, have good security, be easy to program, be simple to upgrade and scale, accommodate heterogeneity, have good real-time performance, and be easy to operate. At the same time, we want all these qualities at the lowest possible cost.

One approach to having a system embodying some of these qualities is to build larger, faster machines with components of greater tolerance and massive amounts of support software. Unfortunately, these approaches have technological limits – they also tend to be very expensive.

Another approach is to employ more than one processor. It seems obvious that using five or ten or 100 processors on a problem (if used appropriately) should solve it faster and provide greater capacity for load than a single processor would. Having multiple systems means that we can replicate data and services, and thus continue our work if some of them fail or become unavailable. We can firewall security and failure problems at machine boundaries and can add more processors whenever we need to expand or upgrade the overall system. What is perhaps most important is that we can use many inexpensive processors connected together by low-cost technology – rather than spending millions of dollars on the latest turbocharged, liquid helium cooled, super-mega-VLSI nanotechnology single-processor machine that may be obsolete within a year of its installation.

Well, if the solution is so simple, why are most systems single-processor models with minimal connectivity, or multiprocessor systems that act like uniprocessors? It is because this approach raises a new set of challenging problems: consistency of distributed and replicated data, synchronization, load balancing, data and protocol heterogeneity, user interface complexities, and more. These are difficult not only because they admit no ready solutions, but because some of them have such subtle and far-reaching implications it is not even clear what the exact nature of the problems may be.

For the past two decades, researchers in academia and industry have tried to solve these problems. Progress has been made in some areas, but problems remain. Sometimes, solutions to one problem lead to a new set of difficulties that need to be addressed. For instance, availability can be enhanced by replicating data at multiple sites in a distributed system, but this introduces problems in maintaining the consistency of that data.

Still, despite the setbacks, researchers continue to try to build systems of multiple processors, whether tightly coupled across a high-speed bus or loosely connected over transcontinental network links. The interest persists because we still want those fast, powerful, flexible, reliable computer systems. If such a system is ever made practical and effective, many people will be willing to part with large sums of money to have one of their own.

Thus, there has been considerable research on multiprocessor systems over the past 20 years, and especially in the last decade. Yet, despite the efforts in these areas by hundreds of researchers, we have seen only a few dozen operational research or commercial systems based on cooperating processors; the problems involved in building practical systems are formidable.

Contributing to the difficulty of building distributed and multiprocessor systems is the lack of information. There is no lack of information on theory or design ideas, as can be seen by reading the journals and conference proceedings that cover the area. There is, unfortunately, very little published about the underlying experiences and design decisions that go into building these systems. As someone who designed and built an experimental distributed system, I can think of many complex decisions and problems encountered during the process that must be common to similar efforts, but that have not appeared in sources accessible to the community.

How do you debug an experimental distributed system? What language is best to use? How do you manage revision control for multiple architectures? What do you do to observe and manage messages between processes? How do you optimize performance? How do you define performance in such a system? How do you compare designs? What tools are available to aid in the construction and evaluation of such systems? Do you build on top of an existing system and inherit some of its difficulties, or do you go to the trouble of building on the bare machine? All these questions, and more, face those who design and implement distributed and multiprocessor systems. Without archival material to reference, each new project has to address many of the same questions again.

About this Issue

In late 1988, George Leach of AT&T Paradyne approached the USENIX Association with the idea of having a workshop devoted to exchange of information based on experiences with distributed systems. He also contacted me, and together we organized the first Workshop on Experiences with Distributed and Multiprocessor Systems. The workshop was sponsored by USENIX, and by the Software Engineering Research Center, an NSF university and industry cooperative research center co-located at Purdue University and the University of Florida. George also handled the paperwork to get cooperating sponsorship with the ACM and the Computer Society of the IEEE.

The workshop was held October 5-6, 1989, in Ft. Lauderdale. It attracted almost 60 submissions, of which 25 were selected for presentation. Over 120 people attended the workshop, and their comments were so positive that we decided to share some of the material with a wider audience – thus, this special issue of *Computing Systems*. We also will hold the event again, in symposium format, in Atlanta, Georgia in March 1991.

Of the eight articles receiving the highest evaluations by the program committee and the most comments from the attendees, one was about material already described in a *Computing Systems* article ("Chorus Distributed Operating Systems," **1**(4), Fall 1988), one had already been submitted to *Computing Systems* ("SOS: An Object-Oriented Operating System," **2**(4), Fall 1989), and one had been submitted to another journal. The remaining five were revised and appear in this issue.

The Papers

The papers are presented in alphabetical order, by name of the system. The first is on the Clouds distributed system. In it, Partha Dasgupta, Ray Chen, and others in the Clouds group at Georgia Tech describe experiences with the first Clouds prototype that influenced the design of the second-generation kernel, Ra.

The second paper, by James Alberi and Marc Pucci, describes some of the unique features of the Dune experimental distributed system built at Bell Communications Research (Bellcore). They describe how they have designed interprocessor communication to be efficient over a wide range of interconnection technologies.

The third paper is by Joseph Boykin and Alan Langerman at Encore Computer Corporation. It describes how they went about parallelizing parts of the Mach operating system without having to reimplement major portions of code. They describe the philosophy behind their changes as well as analyze the parallelized system's performance.

The fourth paper, by Michael Scott, Tom LeBlanc, Brian Marsh, and others at the University of Rochester, describes some of the decisions and constraints affecting implementation of the Psyche parallel system. Their paper provides some insights into the issues that must be addressed when developing an operating system for large, shared memory multiprocessors.

In the final paper, Henry Massalin and Calton Pu of Columbia University describe a scheduling mechanism they have developed for their Synthesis experimental system (cf. *Computing Systems*, **1**(1), Winter 1988, "The Synthesis Kernel"). Although not specifically about a multiprocessor system, the paper provides some insight into alternative scheduling methods that could be used with such systems.

Acknowledgments

My thanks to the members of the workshop program committee. The task of reviewing the papers was complicated by the unexpectedly large number of submissions, but the committee's efforts ensured an interesting workshop, excerpts of which appear in this special issue. The program committee consisted of: Bharat Bhargava, Joseph Boykin, Rob Kolstad, George Leach, Darrell Long, James Mankovich, Eugene Miya, David Pitts, and myself. Thanks also to Peter Salus, Ellie Young, Judy DesHarnais, and everyone at USENIX, and to Elizabeth Northern and Shirley Shrum at the SERC for their assistance in making the workshop a reality.

Journals are only as good as both the authors and reviewers involved. Even if we can start from the most exceptional papers, it requires patient, detailed examination by reviewers to help shape an issue. The reviewers for this special issue contributed many insightful comments about the submitted papers. Their feedback was helpful to the authors (and to me), and all the papers appearing here include some changes suggested by them. Each paper was examined by at least three people, and I am grateful for all the time and effort they put into their reviews:

Brian Bershad (University of Washington),
Bharat Bhargava (Purdue University),
Kenneth Birman (Cornell University),
Raphael Finkel (University of Kentucky),
James Griffeon (Purdue University),
Kurt Holmquist (AT&T Paradyne),
Chris Kent (Digital Equipment Corporation),
Rob Kolstad (Sun Microsystems),
Hugh Lauer (Kodak Boston Technology Center),
George Leach (AT&T Paradyne),
Darrell Long (University of CA at Santa Cruz),
Enrique Mafla (Purdue University),
Sape Mullender (Centrum voor Wiskunde en Informatica (CWI)),
Mike O'Dell (Bellcore),
Larry Peterson (University of Arizona),
David Pitts (University of Lowell),
Rick Rashid (Carnegie-Mellon University),
Karsten Schwan (Georgia Institute of Technology),
and Michal Young (Purdue University).