Operating System Directions for the Next Millennium

William J. Bolosky, Richard P. Draves, Robert P. Fitzgerald,
Christopher W. Fraser, Michael B. Jones, Todd B. Knoblock, Rick Rashid

Microsoft Research
One Microsoft Way
Redmond WA 98052
http://www.research.microsoft.com

Abstract

We believe it is time to reexamine the operating system’s role in computing. Operating systems exist to create an environment in which compelling applications come to life. They do that by providing abstractions built on the services provided by hardware. We argue that advances in hardware and networking technology enable a new kind of operating system to support tomorrow’s applications. Such an operating system would raise the level of abstraction for developers and users, so that individual computers, file systems, and networks become unimportant to most computations in the same way that processor registers, disk sectors, and physical pages are today.

Introduction

The users, operators, and programmers of distributed systems face many problems. Users of the World Wide Web are subjected to random performance and service disruptions. Replacing or upgrading a personal computer, workstation, or server is very difficult. Even a moderate size computer network requires significant expertise to configure and maintain. The principal programming abstractions available today; processes, threads, files, and sockets; do not adequately address the problems of managing locality, availability, or fault tolerance.

We believe that a distributed operating system, based on a few principles pervasively applied, could address these problems. Such a system would enforce extreme location transparency; any code fragment might run anywhere, any data object might live anywhere; and the system would manage the locality, replication, and migration of computations and data. The system would be self-configuring, self-monitoring, and self-tuning. And of course, it would be scalable and secure.

As part of the Millennium project at Microsoft Research, we are attempting to design and build such a system. We do not harbor the conceit that it will be possible to be fully successful in such an endeavor, but we do feel that the time is right for radical experimentation.

This position paper describes our goals for the system and some principles that support those goals. We then discuss relevant technology trends and consider some examples of what such a system would be like. Finally, we discuss related research efforts and conclude.

Goals

Our goal is to dramatically improve users; computational experience. To that end, the system should have at least the following properties:

- **Seamless distribution.** The system should determine where computations execute or data resides, moving them dynamically as necessary. Users should be able to use any computing device that is part of the distributed system as naturally and productively as they would use the machine on their desk or in their den at home.
- **Worldwide scalability.** Logically there should be only *one* system, although at any one time it may be partitioned into many pieces. For example, disconnected or weakly-connected operation creates temporary network partitions.
- **Fault-tolerance.** The system should transparently handle failures or removal of machines, network links, and other resources without loss of data or functionality. This should hold true for both the system itself and for its applications.
- **Self-tuning.** The system should be able to reason about its computations and resources, allocating, replicating, and moving computations and data to optimize its own performance, resource usage, and fault-tolerance.
- **Self-configuration.** New machines, network links, and resources should be automatically assimilated.
- **Security.** Although a single system image is presented, data and computations may be in many different trust domains, with different rights and capabilities available to different security principals. Like the Internet, the system should allow non-hierarchical trust domains with no central authority necessary.
- **Resource controls.** Both providers and consumers may explicitly manage the use of resources belonging to different trust domains. For instance, while some people might be content to allow their data and computations to use any resources available anywhere, some companies might choose, for instance, not to store or compute their year-end financial statement on their competitor’s machines.

Principles

To achieve these goals, we propose a distributed operating system built on the principles of aggressive abstraction, storage-irrelevance, location-irrelevance, just-in-time binding, and introspection:

- **Aggressive abstraction.** The level of abstraction should be raised to the point that application programmers are freed from the mechanics of distributed programming and the exigencies of physical computing components. This would allow them to focus on application concerns; actually solving a problem for a user; rather than system concerns such as communication or fault tolerance. To the greatest extent possible, the system should handle difficult issues like data placement, resource location, fault-tolerance, and load-balancing.
- **Storage-irrelevance.** There should be no storage hierarchy. Once created, information should be accessible until it is no longer needed or referenced.
- **Location-irrelevance.** Objects should be allowed to reference each other and invoke operations without regard for their current location or replication state. The system should have a seamless appearance despite its underlying distributed nature.
- **Just-in-time binding.** Bindings to particular computations, data, and hardware resources should be made only when actually required, preventing applications from creating bindings that would interfere with distribution or fault tolerance. Computations or data could be arbitrarily duplicated and bindings made to one instance would be equivalent to bindings to other instances. Just-in-time binding would enhance resource access and management as the operating system could redirect access to a resource at any time or for any reason to an

equivalent entity.

- **Introspection.** The system should possess some aspects of self-examination and reflection. It should pervasively monitor itself and its applications, and reason about configuration and performance issues. Its models of its own configuration and operation should suggest opportunities for self-tuning as well as generate suggestions for physical configuration changes or upgrades that would improve performance.

Aggressive abstraction addresses a number of the goals that we have laid out. The system should provide applications with highly abstract virtual machine semantics. Defining computations in software terms, independent of any particular platform or configuration, would allow state to be maintained in an abstract form and only be compiled to an efficient executable form at the last possible moment. Because the operating system would retain ultimate control over computation, typical applications would not even be able to express access to low-level system resources. This would allow the operating system to protect computations from each other, transparently migrate and distribute them, and intercede in their access to resources.

Storage and location irrelevance address the issue of seamless interaction with information independent of its properties. Interactive users could walk up to any computer anywhere and have their active applications and personal data follow them. Programs should work the same regardless of how their components are distributed.

The Internet provides good examples today of the kind of just-in-time binding we would extend to all object interactions. Server names and addresses on the Internet often do not refer to specific resources but to collections of systems that take on an Internet "identity" or address for a particular collection of users. Users are becoming accustomed to what can best be described as "binding-by-search" as information is referenced not by name but by keyword or attributes.

Introspection guarantees that the system should take responsibility for determining where a computation executes or data resides. The programmer should not have to decide whether code will execute at a "client" or "server." Instead the system’s assessment of its hardware resources and usage patterns should determine the placement of computations and data. This would allow the operating system to provide fault tolerance, high availability, and self-tuning behavior for applications.

Technology Trends

While it might seem extreme to pervasively apply these principles to the point of removing from programmers many of the decisions currently seen as important in application development, this approach to operating system design is really just an extrapolation of current trends. Throughout the history of computer science, advances in operating system and programming language design have raised the level of abstraction and removed control from application programmers.

This trend is inevitable. It is driven by the continuing exponential increase in the performance of computing hardware. In any software project, there is a tradeoff between the amount of programming effort, the application functionality, and the resulting performance. Better hardware means that there is more performance available to be spent on increased functionality and reduced development cost. Furthermore, the relationship between more powerful abstractions and development cost is non-linear. It is hard to imagine any amount of engineering effort producing a system like today’s World Wide Web using only 1950’s software technology, even if it were running on modern hardware. (Admittedly, there are occasional setbacks to this trend in which people focus on bypassing abstractions for improved performance.)

Many advances in operating systems and networking both demonstrate this trend and provide some technical underpinnings for another step forward. Distributed file systems [e.g., Anderson et al. 96, Kistler & Satya 92] have succeeded in providing efficient location-transparent access to data. Some recent file system work [Wilkes et al. 96, Neefe et al. 96] focuses on auto-configuring and self-tuning for different loads. Network auto-configuration at the LAN level [Rodeheffer & Schroeder 91, Thomson & Narten 96] is increasingly possible. Distributed garbage collection remains a difficult problem, but progress is being made [Ferreira & Shapiro 94]. Systems for distributing parallel work now exist [Geist et al. 94, Livny 95].

Programming languages and compilers provide evidence of a similar evolution. What were once programmer decisions have increasingly been automated. Where once it was routine to write in assembly language, now in most cases a compiler for a higher level language like C++ does almost as well. Debugging, portability, and maintainability are improved, and the compiler is better able to analyze a myriad of details to optimize instruction selection and scheduling, register allocation and spilling, etc. Today, this trend is continuing with the popularity of easy-to-use environments like Java, Visual Basic, and Delphi. The emphasis in the tools industry is shifting from code efficiency to rapid application development with wizards that automatically generate scaffolding or framework code. Virtual machine environments with just-in-time compilation [Adl-Tabatabai 96, Arnold & Gosling 96] extend the programmer’s insulation from processor architecture details, delaying that binding until run time.

What would such a system be like?

Given a distributed system with the properties described above, how might it function in practice? In this section, we present several scenarios that highlight the major aspects of our position.

A New Machine

A user purchases and installs a new personal computer or workstation. The hardware vendor has done a good job with the cables and connectors, so plugging the system together is easy. The user plugs the power and network cables into the wall and flips the power switch. From the moment that a boot ROM, or perhaps a boot loader on disk, downloads code from the network, the new machine joins the Millennium system. The user has full access to Millennium with no human-managed configuration activities required. Millennium evaluates the hardware resources of the new computing device that it has acquired and starts to shift computations and data in response.

A New Network

An administrator sets up a new office network. After connecting the various computers, links, and routers, the network is initially quiescent. The administrator inserts a Millennium installation DVD disk into one of the machines and the system propagates across the network. After evaluating the network topology and hardware resources, Millennium might suggest that one of the more powerful machines (a "server") be moved to a different network location for best performance. At some point, the administrator connects the office network to the Internet, and the office instantiation of the Millennium system merges with the worldwide system.

Hardware Failure

An office computer goes down, perhaps a power supply has burned out. The Millennium system reconfigures around the missing machine. While the computer still ran, computations and data stored on it were either automatically replicated to other systems or logged so that they could be rerun or recomputed. At the time of the failure, Millennium identifies unique state that was not replicated and must be reconstructed and initiates those computations automatically. In general, users may detect a hesitation as the system moves or recreates data they attempt to access, but otherwise they are unaware of the change. An interactive user of the failed machine shifts to another machine and resumes work immediately or simply plugs in a replacement.

Web Service

A little-known web site suddenly achieves popularity, perhaps with a link from Cool Site of the Day^{SM} or a mention in a prominent news story. Word of mouth spreads, and soon the web site's servers are overwhelmed. Or rather, would have been overwhelmed except that heuristics in the Millennium system had noticed the new link and already started replicating the site for increased availability. Monitored traffic increases confirm the situation and soon the site's data has been "pre-cached" across the Internet. As the site's usage drops over the following weeks, Millennium reallocates resources to meet new demands.

Distributed Programming

Consider a simple fragment of pseudo-code:

```
for (p = first(S); p not null; p = next(S))
if (p->age == 18 and p->height < 6 and p->weight < 180)
count++;
```

When executed in a distributed environment, the most efficient strategy for code like this is unclear: Should code or data be replicated or moved? Should loop iterations be executed in parallel? And what happens to error handling? In general, making such decisions requires knowledge of network and system parameters plus knowledge of current conditions.

Millennium network operations intrinsically carry code to execute remotely, but the programmer does not deal with this explicitly. Instead the Millennium system automatically and dynamically exploits locality and concurrency opportunities. In the example code fragment, we believe that it is feasible for compiler techniques to identify the conditional test as code that should execute at the object p's location, in most circumstances. In more complicated scenarios, variations at run-time make it desirable to perform these optimizations dynamically or just-in-time.

Related Work

Several current projects share some of our goals or directions.

The WebOS project [Vahdat et al. 96, Vahdat et al. 97] addresses the goal of bringing distributed computing to the World Wide Web. The heart of the WebOS system is WebFS, a global file system. WebFS provides secure cache-coherent access to data. WebOS also includes a remote execution service. WebOS can be used to build highly available and self-tuning web services.

The Inferno&trade's system [Lucent 96] integrates an operating system and language environment with the goal of making it easier to create distributed services. Like Java [Arnold & Gosling 96], Inferno creates a virtual-machine environment for applications. However, Inferno also includes network protocols and services for distributed programming and a stand-alone implementation suitable for minimal hardware environments. (Sun is moving Java in this direction with the development of JavaOS [Mitchell 96].)

The Globe project [Steen 96, Homburg et al. 96] investigates distributed shared objects as a technique for building large-scale distributed systems. Local object representatives hide details like replication and mobility. Local objects have a standard internal structure that makes it easier to reuse code components. One of Globe's major features is a hierarchical distributed location service that adapts dynamically to different usage patterns.

Legion [Grimshaw et al. 97] proposes using an extensible object model to provide a single-system model for a worldwide network of computers. Legion emphasizes high-performance parallel computing. It supports legacy codes by encapsulating them in Legion-style object oriented wrappers. Legion automatically manages object replication and migration and supplies a single, persistent, global name space.

Conclusion

We have described our goals for a new distributed operating system. Improvements in hardware and networking technology, coupled with continuing operating system and programming language advances, enable the creation of an operating system that raises the level of abstraction presented to its applications. The operating system should provide seamless distribution, worldwide scalability, fault-tolerance, self-tuning, self-configuration, security, and resource controls.

References

[Adl-Tabatabai 96] Ali-Reza Adl-Tabatabai, Geoff Lnagdale, Steven Lucco and Robert Wahbe. "Efficient and Language-Independent Mobile Programs." In *Proceedings of the ACM SIGPLAN '96 Conference on Programming Language Design and Implementation*, pp. 127-136, May 1996.

[Anderson et al. 96] T. Anderson, M. Dahlin, J. Neefe, D. Roselli, D. Patterson, and R. Wang. "Serverless Network File Systems." *ACM Transactions on Computer Systems*, 14(1), February 1996.

[Arnold & Gosling 96] Ken Arnold and James Gosling. The Java Programming Language. Addison-Wesley, 1996.

[Ferreira & Shapiro 94] Paulo Ferreira and Marc Shapiro. "Garbage Collection and DSM Consistency." In *Proceedings of the First Symposium on Operating Systems Design and Implementation*, Monterey, California, November 1994.

[Geist et al. 94] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing, Scientific and Engineering Series, MIT Press, 1994.

[Grimshaw et al. 97] A. Grimshaw, W. Wulf, and the Legion Team. "The Legion Vision of a Worldwide Virtual Computer." *Communications of the ACM*, 40(1), January 1997.

[Homburg et al. 96] Philip Homburg, Maarten van Steen, and Andrew S. Tanenbaum. "An Architecture for a Wide Area Distributed System." In *Proceedings of the Seventh ACM SIGOPS European Workshop*, Connemara, Ireland, September 1996.

[Kistler & Satya 92] J. J. Kistler and M. Satyanarayanan. "Disconnected Operation in the Coda File System." *ACM Transactions on Computer Systems*, 10(1):25, February 1992.

[Livny 95] M. Livny. "The Condor Distributed Processing System." *Dr. Dobbs Journal*, pp. 40-58, February 1995.

[Lucent 96] Lucent Technologies. "Inferno: la Commedia Interattiva." http://inferno.lucent.com/inferno/infernosum.html, 1996.

[Mitchell 96] Jim Mitchell. "JavaOS: Back to the Future." Invited talk at the Second Symposium on Operating Systems

Design and Implementation, Seattle, Washington, October 1996.

[Neefe et al. 96] J. Neefe, D. Roselli, R. Wang, T. Anderson, and M. Dahlin. "Improving the Performance of Log Structured File Systems." http://http.cs.berkeley.edu/~neefe/papers/osdi_submit.ps, 1996.

[Rodeheffer & Schroeder 91] Thomas L. Rodeheffer and Michael D. Schroeder. "Automatic Reconfiguration in Autonet." In *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, Pacific Grove, California, pp. 183-197, October 1991.

[Steen 96] Maarten van Steen. "The GLOBE Project." http://www.cs.vu.nl/~steen/globe/, 1996.

[Tennenhouse & Wetherall 96] D. Tennenhouse and D. Wetherall. "Towards an Active Network Architecture." ACM SIGCOMM Computer Communication Review, pp. 5-18, April 1996.

[Thomson & Narten 96] S. Thomson and T. Narten. "IPv6 Stateless Address Autoconfiguration." IETF Request for Comments 1971, August 1996.

[Vahdat et al. 96] Amin Vahdat, Michael Dahlin, and Thomas Anderson. "Turning the Web Into a Computer." http://now.cs.berkeley.edu/WebOS/webos.ps, 1996.

[Vahdat et al. 97] Amin Vahdat, Paul Eastham, Chad Yoshikawa, Michael Dahlin, and Thomas Anderson. "WebOS: Software Support for Scalable Web Services." Submitted to the *Sixth Workshop on Hot Topics in Operating Systems*, Chatham, Massachussetts, May 1997.

[Wilkes et al. 96] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. "The HP AutoRAID Hierarchical Storage System." *ACM Transactions on Computer Systems*, 14(1): 108-136, February 1996.