Operating Systems Should Support Business Change

Jeffrey C. Mogul
HP Labs, Palo Alto
Jeff.Mogul@hp.com

Abstract

Existing enterprise information technology (IT) systems often inhibit business flexibility, sometimes with dire consequences. In this position paper, I argue that operating system research should be measured, among other things, against our ability to improve the speed at which businesses can change. I describe some of the ways in which businesses need to change rapidly, speculate about why existing IT infrastructures inhibit useful change, and suggest some relevant OS research problems.

1 Introduction

Businesses change. They merge; they split apart; they reorganize. They launch new products and services, retire old ones, and modify existing ones to meet changes in demand or competition or regulations. Agile businesses are more likely to thrive than businesses that cannot change quickly.

A business can lack agility for many reasons, but one common problem (and one that concerns us as computer scientists) is the inflexibility of its IT systems. "Every business decision generates an IT event" [1]; For example, a decision to restrict a Web site with product documentation to customers with paid-up warranties requires a linkage between that Web site and the warranty database. If the IT infrastructure deals with such "events" slowly, the business as a whole will respond slowly; worse, business-level decisions will stall due to uncertainty about IT consequences.

What does this have to do with operating systems? Surely the bulk of business-change problems must be resolved at or above the application level, but many aspects of operating system research are directly relevant to the significant problems of business change. (I assume a broad definition of "operating system" research that encompasses the entire, distributed operating environment.)

Of course, support for change is just one of many problems faced by IT organizations (ITOs), but this paper focusses on business change because it seems underappreciated by the systems software research community. We are much better at problems of performance, scale, reliability, availability, and (perhaps) security.

2 IT vs. business flexibility

Inflexible IT systems inhibit necessary business changes. The failure to rapidly complete an IT upgrade can effectively destroy the value of a major corporation (e.g., [12]). There is speculation that the Sept. 11, 2001 attacks might have been prevented if the FBI had had more flexible IT systems [17, page 77]. Even when IT inflexibility does not contribute to major disasters, it frequently imposes costs of hundreds of millions of dollars (e.g., [13, 14]).

The problem is not limited to for-profit businesses; other large organizations have similar linkages between IT and their needs for change. For example, the military is a major IT consumer with rapidly evolving roles; hospitals are subject to new requirements (e.g., HIPAA; infection tracking); universities innovate with IT (e.g., MIT's OpenCourseWare); even charities must evolve their IT (e.g., for tracking requirements imposed by the USA PATRIOT Act). The common factor is a large organization that thinks in terms of buying "enterprise IT" systems and services, not just desktops and servers.

3 Why is application deployment so slow?

IT organizations often spend considerably more money on "software lifecycle" costs than on hardware purchases. These costs include software development, testing, deployment, and maintenance. In 2004, 8.1% of worldwide IT spending went to server and storage hardware combined, 20.7% went to packaged software, but 41.6% went to "services," including 15.4% for "implementation" [15]. Even after purchasing packaged software, IT departments spend tons of money actually making it work [12].

Testing and deployment also impose direct hardware costs; for example, roughly a third of HP's internal servers are dedicated to these functions, and the fraction is larger at some other companies [21]. These costs are high because these functions take far too long. For example, it can take anywhere from about a month to almost half a year for an ITO to certify that a new server model is acceptable for use across a large corporation's data centers. (This happens *before* significant application-level testing!)

It would be useful to know why the process takes so long, but I have been unable to discover any careful categorization of the time spent. (This itself would be a good

research project.) In informal conversations, I learned that a major cause of the problem is the huge range of operating system versions that must be supported; although ITOs try to discourage the use of obsolete or modified operating systems, they must often support applications not yet certified to use the most up-to-date, vanilla release. The large number of operating system versions multiplies the amount of testing required.

Virtual machine technology can reduce the multiplication effect, since VMs impose regularity above the variability of hardware platforms. Once a set of operating systems has been tested on top of a given VM release, and that release has been tested on the desired hardware, the ITO can have some faith that any of these operating systems will probably work on that hardware with that VM layered in between. However, this still leaves open the problem of multiple versions of the VM software, and VMs are not always desirable (e.g., for performance reasons).

The long lead time for application deployment and upgrades contributes directly to business rigidity. A few companies (e.g., Amazon, Yahoo, Google) are considered "agile" because their IT systems are unusually flexible, but most large organizations cannot seem to solve this problem.

4 Where has OS research gone wrong?

At this point, the reader mutters "But, but, but ... we operating system researchers are all about 'flexibility!'." Unfortunately, it has often been the wrong kind of flexibility.

To oversimplify a bit, the two major research initiatives to provide operating system flexibility have been microkernels (mix & match services outside the kernel) and extensible operating systems (mix & match services inside the kernel). These initiatives focussed on increasing the flexibility of system-level services available to applications, and on flexibility of operating system implementation. They did not really focus on increasing application-level flexibility (perhaps because we have no good way to measure that; see Section 6).

Outside of a few niche markets, neither microkernels nor extensible operating systems have been successful in the enterprise IT market. The kinds of flexibility offered by either technology seems to create more problems than they solve:

- The ITO (or system vendor) ends up with no idea what daemons or extensions the user systems are actually running, which makes support *much* harder. It is hard to point the finger when something goes wrong.
- The ITO has no clear definition of what configurations have been tested, and ends up with a combinatorial explosion of testing problems. ("Safe" extensions are not really safe at the level of the whole IT system; they just avoid the obvious interface violations. Bad interactions through good interfaces are not checked.)
- The ITO has more difficulty maintaining a consistent execution environment for applications, which means that application deployment is even more difficult.

One might argue that increased flexibility for the operating system designer can too easily lead to *decreased* flexibility for the operating system user; it's easier to build novel applications on bedrock than on quicksand.

In contrast, VM research has led to market success. The term "virtual machine" is applied both to systems that create novel abstract execution environments (e.g., Java bytecodes) and those that expose a slightly abstract view of a real hardware environment (e.g., VMware or Xen [4]). The former model is widely seen as encouraging application portability through the provision of a standardized foundation; the latter model has primarily been viewed by researchers as supporting better resource allocation, availability, and manageability. But the latter model can also be used to standardize execution environments (as exemplified by PlanetLab [5] or Xenoservers [7]); VMs do aid overall IT flexibility.

5 How could OS research help?

In this section I suggest a few of the many operating system research problems that might directly or indirectly improve support for business change.

5.1 OS support for guaranteed sameness

If uncontrolled or unexpected variation in the operating environment is the problem, can we stamp it out? That is, without abolishing all future changes and configuration options, can we prevent OS-level flexibility from inhibiting business-level flexibility?

One way to phrase this problem is: can we *prove* that two operating environments are, in their aspects that affect application correctness, 100.00000000% identical? That is, in situations where we do not want change, can we formally prove that we have "sameness"?

Of course, I do not mean that operating systems or middleware should never be changed at all. Clearly, we want to allow changes that fix security holes or other bugs, improvements to performance scalability, and other useful changes that are irrelevant to the stability of the application. I will use the term "operationally identical" to imply a notion of useful sameness that is not too rigid.

If we could prove that host A is operationally identical to host B, then we could have more confidence that an application, once tested on host A, would run correctly on host B. More generally, A and B could each be clusters rather than individual hosts.

Similarly, if we could prove that A_0 is operationally

identical to $A_1, ..., A_n$, an application tested only on A_0 might be safe to deploy on $A_1, ..., A_n$.

It seems likely that this would have to be a formal proof, or else an ITO probably would not trust it (and would have to fall back on time-consuming traditional testing methods). However, formal proof technology typically has not been accessible to non-experts. Perhaps by restricting an automated proof system to a sufficiently narrow domain, it could be made accessible to typical IT staff.

On the other hand, if an automated proof system fails to prove that A and B are identical, that should reveal a specific aspect (albeit perhaps one of many) in which they differ. That could allow an ITO either to resolve this difference (e.g., by adding another configuration item to an installation checklist) or to declare it irrelevant for a specific set of applications. The proof could then be reattempted with an updated "stop list" of irrelevant features.

It is vital that a sameness-proof mechanism cover the entire operating environment, not just the kernel's API. (Techniques for sameness-by-construction might be an alternative to formal proof of sameness, but it is hard to see how this could be applied to entire environments rather than individual operating systems.) Environmental features can often affect application behavior (e.g., the presence and configuration of LDAP services, authentication services, firewalls, etc. [24]). However, this raises the question of how to define "the entire environment" without including irrelevant details, such as specific host IP addresses, and yet without excluding the relevant ones, such as the correct CIDR configuration.

The traditional IT practice of insisting that only a few configuration variants are allowed can ameliorate the sameness problem at time of initial application deployment. However, environments cannot remain static; frequent mandatory patches are the norm. But it is hard to ensure that every host has been properly patched, especially since patching often affects availability and so must often be done in phases. For this and similar reasons, sameness can deteriorate over time, which suggests that a sameness-proof mechanism would have to be reinvoked at certain points.

Business customers are increasingly demanding that system vendors pre-configure complex systems, including software installation, before shipping them. This can help establish a baseline for sameness, but vendor processes sometimes change during a product lifetime. A sameness-proof mechanism could ensure that vendor process changes do not lead to environmental differences that would affect application correctness.

5.2 Quantifying the value of IT

A business cannot effectively manage an IT system when it does not know how much business value that system generates. Most businesses can only estimate this value, for lack of any formal way to measure it. Similarly, a business that cannot quantify the value of its IT systems might not know when it is in need of IT-level change.

ITOs typically have budgets separate from the profit-and-loss accountability of customer-facing divisions, and thus have much clearer measures of their costs than of their benefits to the entire business. An ITO is usually driven by its local metrics (cost, availability, number of help-desk calls handled per hour). ITOs have a much harder time measuring what value its users gain from specific practices and investments, and what costs are absorbed by its users. As a result, large organizations tend to lack global rationality with respect to their IT investments. This can lead to either excessive or inadequate caution in initiating business changes. (It is also a serious problem for accountants and investors, because "the inability to account for IT value means [that it is] not reflected on the firm's [financial reports]", often creating significant distortions in these reports [23].)

Clearly, most business value is created by applications, rather than by infrastructure and utilities such as backup services [23]. This suggests that most work on value-quantification must be application-specific; why should we think operating system research has anything to offer?

One key issue is that accounting for value, and especially in ascribing that value to specific IT investments, can be quite difficult in the kinds of heavily shared and multiplexed infrastructures that we have been so successful at creating. Technologies such as timesharing, replication, DHTs, packet-switched networks and virtualized CPUs, memory, and storage make value-ascription hard.

This suggests that the operating environment could track application-level "service units" (e.g., requests for entire Web pages) along with statistics for response time and resource usage. Measurements for each category of service unit (e.g., "catalog search" or "shopping cart update") could then be reported, along with direct measurements of QoS-related statistics and of what IT assets were employed. The Resource Containers abstract [2] provides a similar feature, but would have to be augmented to include tracking information and to span distributed environments. Magpie [3] also takes some steps in this direction.

Accounting for value in multiplexed environments is not an easy problem, and it might be impossible to get accurate answers. We might be limited to quantifying only certain aspects of IT value, or we might have to settle for measuring "negative value," such as the opportunity cost of unavailability or delay. (An IT change that reduces a delay that imposes a clear opportunity cost has a fairly obvious value.)

5.3 Pricing for software licenses

Another value-related problem facing ITOs is the cost of software licenses. License fees for many major software products are based on the number of CPUs used, or on total CPU capacity. It is now widely understood that this simple model can discourage the use of technologies that researchers consider "obviously" good, including multi-core and multi-threaded CPUs, virtualized hardware, grid computing [22], and capacity-on-demand infrastructure. Until software vendors have a satisfactory alternative, this "tax on technology innovation with little return" [8] could distort ITO behavior, and inhibit a "business change" directly relevant to our field (albeit a one-time change).

The solution to the software pricing crisis (assuming that Open Source software cannot immediately fill all the gaps) is to price based on value to the business that buys the software; this provides the right incentives for both buyer and seller. (Software vendors might impose a minimum price to protect themselves against incompetent customers.)

Lots of software is already priced per-seat (e.g., Microsoft Office and many CAD tools) or per-employee (e.g., Sun's Java Enterprise System [18]), but these models do not directly relate business value to software costs, and might not extend to software for service-oriented computing.

Suppose one could instead track the number of application-level service units successfully delivered to users within proscribed delay limits; then application fees could be charged based on these service units rather than on crude proxies such as CPU capacity. Also, software vendors would have a direct incentive to improve the efficiency of their software, since that could increase the number of billable service units. Such a model would require negotiation over the price per billable service unit, but by negotiating at this level, the software buyer would have a much clearer basis for negotiation.

Presumably, basing software fees on service units would require a secure and/or auditable mechanism for reporting service units back to the software vendor. This seems likely to require infrastructural support (or else buyers might be able to conceal service units from software vendors). See Section 5.5 for more discussion of auditability.

One might also want a system of trusted third-party brokers to handle the accounting, to prevent software vendors from learning too much, too soon, about the business statistics of specific customers. A broker could anonymize the per-customer accounting, and perhaps randomly time-shift it, to provide privacy about business-level details while maintaining honest charging.

5.4 Name spaces that don't hinder organizational change

Operating systems and operating environments include lots of name spaces; naming is key to much of computer systems design and innovation.[1] We name system objects (files, directories, volumes, storage servers, storage services), network entities (links, switches, interfaces, hosts, autonomous systems), and abstract principals (users, groups, mailboxes, messaging servers).

What happens to these name spaces when an organizations combine or establish a new peering relationship? Often these business events lead to name space problems, either outright conflicts (e.g., two servers with the same hostname) or more abstract conflicts (e.g., different designs for name space hierarchies). Fixing these conflicts is painful, slow, error-prone, and expensive. Alan Karp has articulated the need to "design for consistency under merge" to avoid these conflicts [10].

And what happens to name spaces when an organization is split (e.g., as in a divestiture)? Some names might have to be localized to one partition or another, while other names might have to continue to resolve in all partitions. One might imagine designing a naming system that supports "completeness after division," perhaps through a means to tag certain names and subspaces as "clonable."

When systems researchers design new name spaces, we cannot focus only on traditional metrics (speed, scale, resiliency, security, etc.); we must also consider how the design supports changes in name-space scope.

5.5 Auditability for outsourcing

IT practice increasingly tends towards outsourcing (distinct from "offshoring") of critical business functions. Outsourcing can increase business flexibility, by giving a business immediate access to expertise and sometimes by better multiplexing of resources, but it requires the business to *trust* the outsourcing provider. Outsourcing exposes the distinction between *security* and *trust*. Security is a technical problem with well-defined specifications, on which one can, in theory, do mathematical proofs. Trust is a social problem with shifting, vaque requirements; it depends significantly on memory of past experiences. Just because you can prove to yourself that your systems are secure and reliable does not mean that you can get your customers to entrust their data and critical operations to you.

This is a variant of what economists call the "principal-agent problem." In other settings, a principal could establish its trust in an agent using a third-party auditor, who has sufficient access to the agent's environment to check for evidence of incorrect or improper practices. The auditor has expertise in this checking process that the principal does not, and also can investigate agents who serve multiple principals without fear of in-

formation leakage.

Pervasive outsourcing might therefore benefit from infrastructural support for auditing; i.e., the operating environment would support monitoring points to provide "sufficient access" to third-party auditors. Given that much outsourcing will be done at the level of operating system interfaces, some of the auditing support will come from the operating system. For example, the system might need to provide evidence to prove that principal A cannot possibly see the files of principal B, and also that this has never happened in the past.

6 Operating outside our comfort zone

The problems of enterprise computing, and especially of improving business-level (rather than IT) metrics, is far outside the comfort zone of most operating system researchers. Problems include

- The applications are not the ones we use or write ourselves; it is hard to do operating system research using applications one does not understand.
- Most of these applications are not Open Source; researchers cannot afford them, and some vendors ban unauthorized benchmarking.
- The applications can be hard to install. A typical SAP installation might involve millions of dollars of consultant fees over months or even years to customize it [11].
- We do not have a good description of "real workloads" for these applications.

In addition, many of the problems inhibiting business change are cultural, not technical. That does not mean that we are excused from addressing the technical challenges, but this is an engineering science, so our results need to respect the culture in which they would be used. That means that computer science researchers need to learn about that culture, not just complain about it.

6.1 What about metrics?

Perhaps the biggest problem is that we lack quantified metrics for things like "business flexibility." (Low-level flexibility metrics, such as "time to add a new device driver to the kernel," are not the right concept.) Lacking the metrics, we cannot create benchmarks or evaluate our ideas.

Rob Pike has argued that "In a misguided attempt to seem scientific, there's too much measurement: performance minutiae and bad charts. ... Systems research cannot be just science; there must be engineering, design, and art." [20]. But we *must* measure, because otherwise we cannot establish the value of IT systems and processes; however, we should not measure the wrong things ("performance minutiae") simply because those are the easiest for us to measure.

Metrics for evaluating how well IT systems support business change will not be as simple as, for example, measuring Web server transaction rates, for at least two reasons. First, because such evaluations cannot be separated from context; successful change inevitably depends on people and their culture, as well as on IT. Second, because business change events, while frequent enough to be problematic, are much rarer and less repeatable than almost anything else computer scientists measure. We will have to learn from other fields, such as human factors research and economics, ways to evaluate how IT systems interact with large organizations.

I will speculate on a few possible metrics:

- **For software deployment**: It might be tempting to simply measure the time it takes to deploy an application once it has been tested. However, such timing often depends too much on uncontrollable variables, such as competing demands on staff time. A more repeatable metric would be the number of new problems found in the process of moving a "working" application from a test environment to a production environment. The use of bug rates as a metric was proposed in a similar context by Doug Clark [6], who pointed out that what matters is not reducing the total number of bug reports, but finding them as soon as possible, and before a product ships to customers.

  Nagaraja *et al.* reported on small-scale measurements of how frequently operators made mistakes in reconfiguring Internet applications [16]. They described a technique to detect many such errors automatically, using parallel execution of the old system and the new system, comparing the results, with the new system isolated to prevent any errors from becoming visible. Their approach might be generalizable to testing for environmental sameness.

  One might also crudely measure a system's support for deployment of updated applications by subjecting an application to increasingly drastic changes until something breaks. For example, perhaps the operating environment can support arbitrary increases in the number of server instances for an application, but not in the number of geographically separated sites.
- **For quantifying IT value**: Suppose that an enterprise's IT systems generated estimates of their value. One way to test these estimates would be to compare their sum to the enterprise's reported revenue, but this probably would not work: revenue reports are too infrequent and too arbitrary, and it would require nearly complete value-estimation coverage over all IT systems. Instead, one might be able to find correlations between the IT-value estimates from distinct systems and the short-term per-product revenue metrics maintained by many businesses. If the correlations can be used for prediction (e.g., they persist after a system improvement) then they would validate

the IT-value estimates.

In the end, many important aspects of IT flexibility will never be reduced to simple, repeatable metrics. We should not let this become an excuse to give up entirely on the problem of honest measurement.

7 Grand Challenge ... or hopeless cause?

Section 6 describes some daunting problems. How can we possibly do research in this space? I think the answer is "because we must." Support for CS research, both from government and industry, is declining [9, 19]. If operating system research cannot help solve critical business problems, our field will shrink.

The situation is not dire. Many researchers are indeed addressing business-level problems. (Space prohibits a lengthy description of such work, and it would be unfair to pick out just a few.) But I think we must do better at defining the problems to solve, and at recognizing the value of their solution.

Acknowledgments

In preparing this paper, I had help from many colleagues at HP, including Martin Arlitt, Mary Baker, Terence Kelly, Bill Martorano, Jerry Rolia, and Mehul Shah. The HotOS reviewers also provided useful feedback.

References

[1] Q&A with Robert Napier, CIO at Hewlett-Packard. *CIO Magazine*, Sep. 15 2002. http://www.cio.com/archive/091502/napier.html.

[2] G. Banga, P. Druschel, and J. C. Mogul. Resource containers: A new facility for resource management in server systems. In *Proc. OSDI*, pages 45–58, New Orleans, LA, Feb. 1999.

[3] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for request extraction and workload modeling. In *Proc. 6th OSDI*, pages 259–272, San Francisco, CA, Dec. 2004.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. SOSP-19*, pages 164–177, 2003.

[5] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating System Support for Planetary-Scale Services. In *Proc. NSDI*, pages 253–266, San Francisco, CA, Mar. 2004.

[6] D. W. Clark. Bugs are Good: A Problem-Oriented Approach to the Management of Design Engineering. *Research-Technology Management*, 33(3):23–27, May-June 1990.

[7] K. Fraser, S. M. Hand, T. L. Harris, I. M. Leslie, and I. A. Pratt. The Xenoserver computing infrastructure. Tech. Rep. UCAM-CL-TR-552, Univ. of Cambridge, Computer Lab., Jan. 2003.

[8] Garner, Inc. Gartner says cost of software licenses could increase by at least 50 percent by 2006. http://www3.gartner.com/press_releases/asset_115090_11.html, Nov. 2004.

[9] P. Harsha. NSF budget takes hit in final appropriations bill. *Computing Research News*, 17(1), Jan. 2005. http://www.cra.org/CRN/articles/jan05/harsha.html.

[10] A. H. Karp. Lessons from E-speak. Tech. Rep. HPL-2004-150, HP Labs, Sep. 2004. http://www.hpl.hp.com/techreports/2004/HPL-2004-150.html.

[11] C. Koch. Lump it and like it. *CIO Magazine*, Apr. 15 1997. http://www.cio.com/archive/041597/lump.html.

[12] C. Koch. AT&T Wireless Self-Destructs. *CIO Magazine*, Apr. 15 2004. http://www.cio.com/archive/041504/wireless.html.

[13] C. Koch. When bad things happen to good projects. *CIO Magazine*, Dec. 1 2004. http://www.cio.com/archive/120104/contingencyplan.html.

[14] J. C. McGroddy and H. S. L. Editors. A Review of the FBI's Trilogy Information Technology Modernization Program. http://www7.nationalacademies.org/cstb/pub_fbi.html, 2004.

[15] S. Minton, E. Opitz, J. Orozco, F. Chang, S. J. Frantzen, G. Koch, M. Coughlin, T. G. Copeland, and A. Tocheva. Worldwide IT Spending 2004-2008 Forecast. IDC document #32321, Dec. 2004.

[16] K. Nagaraja, F. Oliveira, R. Bianchini, R. P. Martin, and T. D. Nguyen. Understanding and Dealing with Operator Mistakes in Internet Services. In *Proc. OSDI*, pages 61–76, San Francisco, CA, Dec 2004.

[17] National Commission on Terrorist Attacks Upon the United States. Final report, 2004.

[18] J. Niccolai and D. Tennant. Sun pricing model impresses users. *ComputerWeekly.com*, Sep. 2003. http://www.computerweekly.com/Article125119.htm.

[19] M. Pazzani, K. Abdali, G. Andrews, and S. Kim. Cise update: Adjusting to the increase in proposals. *Computing Research News*, 16(5), Nov. 2004. http://www.cra.org/CRN/articles/nov04/pazzani.html.

[20] R. Pike. Systems software research is irrelevant. http://herpolhode.com/rob/utah2000.pdf, 2000.

[21] D. Rohrer. Personal communication, 2005.

[22] P. Thibodeau. Software licensing emerges as grid obstacle. *ComputerWorld*, May 2004. http://www.computerworld.com/printthis/2004/0,4814,93526,00.html.

[23] J. Tillquist and W. Rodgers. Using asset specificity and asset scope to measure the value of IT. *Comm. ACM*, 48(1):75–80, Jan. 2005.

[24] J. Wilkes, J. Mogul, and J. Suermondt. Utilification. In *Proc. 11th SIGOPS European Workshop*, Leuven, Belgium, Sep. 2004.

Notes

[1] I think Roger Needham said that (more eloquently), but I haven't been able to track down a quote.