The Global Computer

Alan H. Karp
HP Laboratories Palo Alto
HPL-2006-9
January 12, 2006*

distributed computing, single system image, computing utility

The problems collaborating with others across the network are apparent to anyone who has tried doing it. That being the case, we should attempt to create an environment that doesn't have those issues. One approach is to make it appear to people and programs that there is only one computer in the world. In building such a system we'll have to solve problems of scalable performance, naming, security, discovery, and manageability. If we're to succeed in hiding the boundaries between machines, the solutions we come up with will have to be part of a coherent architecture.

* Internal Accession Date Only Approved for External Publication
© Copyright 2006 IEEE Published in The Fourth International Conference on Creating, Connecting and Collaborating through Computing (C5 2006), 26-27 January 2006, Berkeley, CA, USA.

The Global Computer

Alan H. Karp
Hewlett-Packard Laboratories
Alan.Karp@hp.com

Abstract

The problems collaborating with others across the network are apparent to anyone who has tried doing it. That being the case, we should attempt to create an environment that doesn't have those issues. One approach is to make it appear to people and programs that there is only one computer in the world. In building such a system we'll have to solve problems of scalable performance, naming, security, discovery, and manageability. If we're to succeed in hiding the boundaries between machines, the solutions we come up with will have to be part of a coherent architecture.

1. Introduction

It's easier to collaborate with someone on the same computer than with someone on a different computer. Few people dispute the truth of that statement. The question, then, is why not put everyone in the world on the same computer. Since there's no one piece of hardware that can support all the people in the world, the question becomes how to make all the computers in the world look like a single computer, a concept called the Global Computer [1].

This vision isn't a new one. Several people proposed variations on the idea of a computing utility [2] [3] [4]. These concepts are coming to fruition in such technologies as the Grid [5] and the Services Oriented Architecture [6]. However, none of these is quite as grandiose as the vision of a Global Computer, although the Worldwide Computer [7] comes close.

One important difference between these other approaches and the Global Computer is the starting point. The others start from a model of network computing and project it into the machine. The Global Computer starts from the model of a single computer and projects it outward into the network, as was done in formulating Actors [8]. The difference is significant. The former approach makes the network visible at all times. It is as if we used URLs to access all files, even local ones. Starting from a model of local computing makes it possible to hide the complexity of the network environment to a greater extent. It's more like the way we deal with remote files. Once the drive is mapped, all access is done as if the file is local.

This goal of hiding the fact that there is a network applies to all resources and services. Perhaps the simplest example is printing in an office environment. You decide to print, select a printer, and your output appears. There's no need to know which machine controls the printer, nor is there any need to move files explicitly from one place to another. Once things are configured, all the details are handled by the system. It's just as if the printer is a device on your computer.

Contrast this simplicity with the way we share files. One day in 1990, when I was sharing an office with David Bailey, I asked him for one of his FFT routines. I watched as he moved his mouse from one X-window on his SGI workstation to another, did a remote login, inserted the file into an email, and sent it to me. A few seconds later, I got the email, opened it, and copied the file to a different machine. What's surprising is that few people ever give this process a moment's thought. It is just the way it's done.

It is possible to do better. An example is the Transparent Computing Facility (TCF) [9], which came as part of IBM's AIX operating system. TCF provided a shared space for a cluster of machines that included a common file system and execution environment for a heterogeneous collection of IBM mainframes and personal computers. If David and I had been part of the same TCF cluster, David would just have told me the name of the file, and I would have accessed it. Alternatively, I could have just run a program that used the file without needing to know where it was. Most of my early thinking about the Global Computer came from my experience with TCF.

Clearly, it's impossible to make the network completely disappear. There are different failure modes and different latencies. Importantly, "Quantity has a quality all its own," as Stalin is reported to have said.

The sheer scale of the Global Computer will make it look different than a single computer. Nevertheless, the goal is to make the network aspects of the global computer like an automobile windshield. Most of the time you're not aware that it's there, but you can focus on it when you need to.

2. What a computer is

If we are to give the user the illusion that there is only one computer in the world, we need to understand what the essence of a computer is to the user. All would agree that a computer consists of a processor with volatile and permanent storage. There must be a means of communicating with the computer, both to tell it what to do and to receive results. These means include keyboards, monitors, printers, and networks.

Most users also include the software environment, as part of their mental model of a computer. The operating system provides certain abstractions, such files and network connections. There are also applications that people feel are needed for a usable system. If the Global Computer provides these features, it will look like a computer to its users.

Anyone who has ever used a timeshared mainframe will understand what kind of computer it will look like. There's a big, shared pile of resources, consisting of computation, memory, and disk. There's a single file system with a name space understood by all users. Any user can access any file just by naming it, given the right permissions, of course. All programs share a common execution environment. The scheduling of tasks to the processors is rarely of interest to the user.

An interesting question in the era of personal computing is whether or not the concept of a user account is important. Many people today are the sole users of their computers. It is fair to say that the login screen on many personal computers is more about keeping strangers out than about determining which legitimate user is sitting at the keyboard. Your cell phone is another example. It's a computer, but the concept of a user never enters your mind as you make a call.

3. Features and issues

One thing that makes implementing the Global Computer hard is that, unlike the mainframe, there's nobody in charge. More precisely, there are many bodies in charge, each responsible for a small portion of the Global Computer. Finding a way to give them adequate control while preserving the illusion of a single computer is a key challenge.

3.1. Common Execution Environment

One of the goals of the Global Computer is to enable resource sharing. Hence, any program should be able to run on any computer compatible with it. Such an environment would also provide a straightforward means to run parallel jobs. Just run fork() and allow the scheduler to start the process on another machine.

A common execution environment also allows for dynamic load balancing. A program can be run on any compatible machine in the Global Computer. Further, if a program has been compiled for more than one architecture, the scheduler can decide between running the job on a heavily loaded supercomputer or an idle workstation.

An intriguing idea comes from combining a common execution environment with process migration, a feature supported by TCF [10]. Say that you start a job just before going home for the evening. If it isn't finished when people start showing up for work in the morning, it can be migrated to a location where people are just going home. Such *heliophobic computing* keeps your application in the dark.

A common execution environment also means that support people can be anywhere in the world. The result is that you can get 24 hour support, but no one has to work the night shift. Overnight support for France could be handled from Tahiti, that for the US from India, that for Russia, well, from Russia. In other words, keep your applications in the dark, not your support people.

System availability would no longer be a problem for business continuity. A machine capable of running a business critical application would always be available, barring a worldwide disaster. Further, such applications would be using a large number of machines throughout the Global Computer. The loss of one of them would only degrade performance by a small amount.

3.2. Dealing with latency

A common execution environment also implies the need for a sophisticated file replication strategy. No one will use a system if network delays increase access times too much. Caching data locally on first access won't hide the initial delay, but most files are used more than once. Prefetching of files that are commonly used together would further reduce apparent network delays.

Clearly, every file won't be replicated to every machine in the Global Computer. Some files, such as compilers and editors, will be replicated to any machine that can run them. Others will be replicated only when being used by a distributed group.

It's not clear whether extensive use of file replication will increase or decrease network traffic. Replication increases traffic because a change sent over the network might not be needed before the next change is made. Replication decreases network traffic because more file requests will be handled locally.

Replication will probably increase the aggregate amount of disk space used, but maybe by less than might be expected. Common practice is to keep a local copy of every file you might ever need. If you could get a file after a small delay just by naming it, there would be less incentive to do that, especially for very large files. Also, replicated copies may be the only backup needed.

Recently developed technologies, such as OceanStore [11] and BitTorrent [12] should prove useful in the Global Computer. They provide interesting ways to store and retrieve files in a large scale environment. Since the network shows through their interface, they should be used by the infrastructure rather than the application API.

3.3. Naming

Naming is an important issue that is often given little attention in distributed systems. It needs special attention in the Global Computer. After all, we wouldn't want an application to break just because someone renamed a computer.

Two properties are particular important, spatial and temporal integrity. Spatial integrity means that the name used to refer to a resource, such as a file, is independent of the locations of both the resource and the requester. Temporal integrity means that the name of an object doesn't change just because some event has occurred.

If we start from a model of network computing and project inward, naming appears to be a simple matter of partitioning the namespace. That's not so easy when there isn't a central authority. Although the Domain Name System (DNS) does work, even it has a number of problems. There are private DNS servers, and there's a potential for a split as objections to US control arise. Another issue is that the name for a resource behind a firewall is often different from the name used when outside the firewall, a violation of spatial integrity. Firewalls also break the global uniqueness of IP addresses. Most wireless networks have a machine with an IP address of 192.168.1.101 because that's the default router configuration. IP addresses also change as machines move across subnets

Naming is also critical to security if it is possible to hijack a name from its legitimate owner. Some schemes make the hash of the public key of an object part of its name. However, that means that its name changes when the key expires, typically once a year, which violates temporal integrity.

3.4. Management

The Global Computer isn't a mainframe, and people wouldn't be happy if it were. Many moved to departmental scale machines as soon as minicomputers became available and later to personal computers. There were many reasons for this trend, but few would argue that management issues, particularly control and charging were important factors.

A mainframe is a precious resource. It's important that one malicious user not interfere with the work being done on behalf of others. Hence, mainframes were tightly controlled by a central organization. Adding a new user often involved getting multiple approvals and waiting several days. We see the same problem arising, albeit in a more distributed environment, on systems, such as the Grid [5] and PlanetLab [13].

Another reason people moved away from central computing was how much they were being charged. Many organizations charged enough to recover their costs, which included amortization of the hardware, software fees, and personnel. The Global Computer will have to find an equitable method for dealing with this issue as people share their resources.

One thing people did like about the mainframe world was the availability of support. Centralized support means that updates, including security patches, are applied in a timely manner. There is someone to manage software licenses and deal with support contracts. Individuals are largely free from worry about security issues, such as who can use the system and what rights people have.

A common execution environment and view of a single file system also simplifies many management tasks. For example, applying an update to a browser that is replicated to most of the machines in the Global Computer means that many people benefit from the action of performing a single update. Of course, making this all work without breaking anything will be a challenge.

3.5. Security

Security is a large topic, with aspects ranging from authentication to privacy to physical security. Many of the methods that have been developed for the Web are applicable to the Global Computer. Others aren't.

One particular aspect that needs rethinking is access control. Today, our computer systems use Access Control Lists (ACLs) as a means of expressing access rights. There are problems with that approach in a dynamic environment because of the burden on administrators [14] and when crossing administrative boundaries. An approach based on explicit authorizations [15] may be better suited to this environment.

In spite of the fact that a geographically dispersed set of machines is harder to protect than one in a locked room, the security on the Global Computer is likely to be tighter than what we have today. There's a reason that corporate computers are better protected than home computers, a professional security team. Many machines, not just in homes, but in small business and universities as well, are vulnerable because there's no one knowledgeable to provide support.

If there is the appearance of only one computer in the world, it will certainly have full time security and auditing organizations. A common execution environment, view of a single file system, and a flexible means of expressing access control policies will make distributed security support simpler.

There are still challenges. With a common execution environment, I may be running someone else's program on my machine. I'll want to limit what that program can do just as I want to limit the damage that an erroneous or malicious program that I run can do. Similarly, if I run my program on someone else's computer, I want to protect the program from the computer. In general, that's impossible, so the Global Computer will need a means for people to express limits on where their jobs run.

3.6. Discovery

It's hard enough to find something on your own computer. Imagine how hard it will be on the Global Computer. Search engines do an adequate job of allowing people to find specific resources on the Internet, and these tools are being made available for individual machines [16]. These desktop search tools will have to adapt their relevance scoring algorithms to the Global Computer. It's quite likely there will be more collections named "baby photos" than any individual cares to see.

The current generation of search engines is not suitable for automated search, since software is still not very good at dealing with ambiguity. Search for "gates" and you'll see electronic gates, garden gates, and Bill Gates. What's needed is an ontology framework for providing the needed context. Universal Description, Discovery, and Integration (UDDI) [17] is the standard for the Services Oriented Architecture, but it is based on a central authority and has a limited set of taxonomies. The Global Computer will need a richer, more dynamic means of addressing the ontology problem.

3.7. System updates

Updating a single computer is easy. Tell everyone you're shutting down at midnight. You can't do that on the Global Computer. For one reason, there's no midnight. What's needed is a way to do rolling updates. That implies reducing the number of global agreements to the absolute minimum. Best would be an architecture the relied on only pairwise agreements. That might be possible if the system doesn't require all pairs of machines to be able to communicate.

4. Building it

The previous section raised a lot of issues, enough to call into question the practicality of the idea. The only way to put the doubts to rest is to build the Global Computer, or at least enough of it to convince people that it might actually work.

4.1. The first experiment

The first attempt at building the Global Computer involved connecting three IBM supercomputers into a single system image using TCF [1]. There were machines in Palo Alto and San Jose, California and one in Yorktown Heights, New York. That wasn't enough machines to test scalability, and they were all owned by the same company, so we sidestepped most of the security concerns. Nevertheless, we were able to test some of the usability features of the Global Computer.

Anyone who's ever attempted to build something as crazy sounding as a Global Computer will appreciate the importance of not letting management know what you're up to. The advantage is freedom. The disadvantage is lack of funding. We cobbled something together, but the network was far from ideal. The result was a network bandwidth measured in tens of KB/s and latencies of the order of a second. Surprisingly, the system was useful.

Support was provided by one person working part time. The common execution environment and shared

file system simplified much of the work. These features also proved their value in other ways, as illustrated by some anecdotes.

One of our staff members was collaborating with a Yorktown researcher on a circuit simulation project. She installed the package they were using in a file system replicated between Palo Alto and Yorktown. Each of the collaborators had access to the software at local disk speeds, yet there was only one version of the code. When a change was made, it was available in a matter of seconds at the other site.

Another occasion where this environment made our lives easier occurred while preparing for a customer benchmark. Most of the work was being done on a model 3090-E in Palo Alto, but the bid was to be for a model 3090-S. In order to know if the jobs would meet the performance targets set in the benchmark we had to make the runs on an S-model. Since we were using a TCF cluster, the job was trivial. We simply moved the work to a replicated file system. All runs were made on the Yorktown S-model by simply typing "on 1 a.out" in the TCF syntax. We had the advantage of local disk speeds, the ability to keep working locally even when the network was down, and it was easy to compare the performance of the two machines.

One final example of the use of this environment involved parallel processing. One of the Palo Alto researchers had developed an iterative solver for linear equations that is parallelizable with large granularity. Over the course of a week, we converted his shared memory program into a message passing version using Unix fork and pipe commands. It was then a simple matter to change the standard Unix fork() command to an AIX/TCF rfork() command to run across the cluster. In one run the job finished in 75 seconds on all 6 processors available to us. The best we did on the 3 processors in Palo Alto was 130 seconds.

IBM closed its Scientific Centers shortly after this work was completed, so we were unable to expand the test. Nevertheless, the positive experience that came from the image of working on a single computer gave weight to the desirability of the Global Computer.

4.2. A delivered product

About five years after the first experiment, I was given the opportunity to try again at Hewlett-Packard Laboratories. In typical industrial research style, we were given six months to produce a demo good enough to convince management to fund a full scale project. A video we made of that demo [18] was compelling enough to enable us to start work on a fully functional prototype. The prototype, which we dubbed Client Utility, or CU, had to deal with many of the issues that we were able to ignore in the experiment at IBM. Eventually, this prototype turned into E-speak [19], the technology behind Hewlett-Packard's E-services initiative.

Figure 1 shows the view of a single machine in the CU architecture. The central parts, labeled "Core", in analogy to an OS kernel, and "Repository" constituted a *Logical Machine*. Applications were run by *Clients*, essentially processes that connected to the

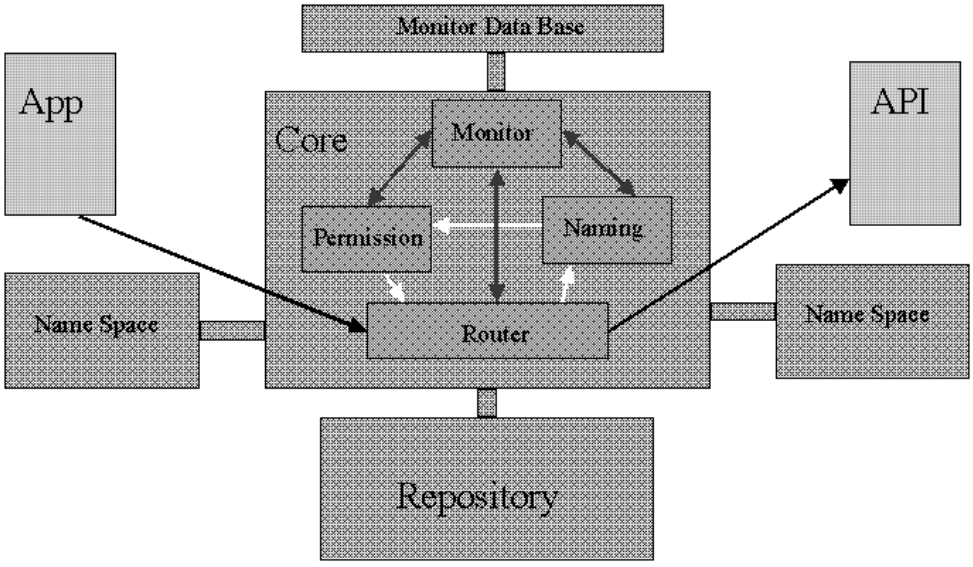

Figure 1. Client Utility on a single machine.

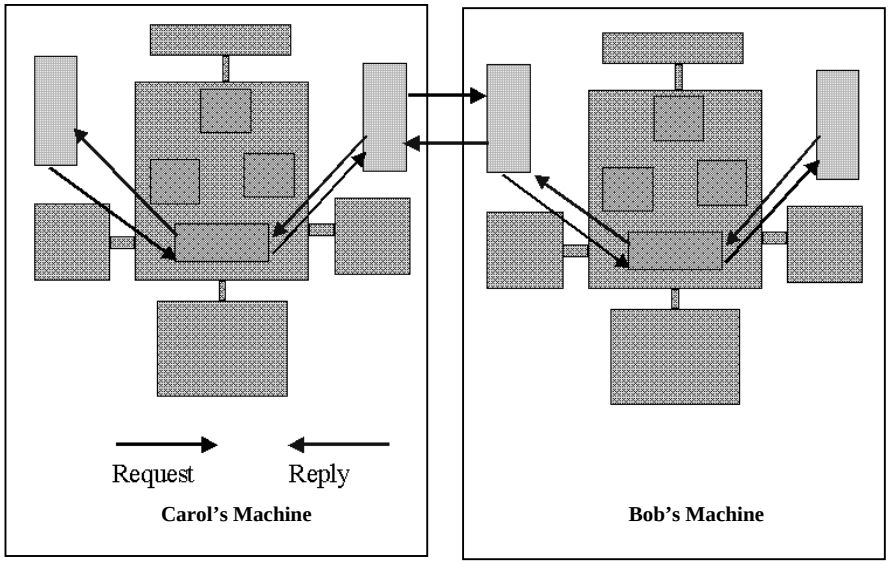

Figure 2. Access between machines.

Core. The Core maintained on behalf of each client a *Protection Domain*, which included the client's quotas and a handle to a name space. The name space contained mappings between strings meaningful to the client and repository handles.

CU worked on a mailbox metaphor. A client would send a message to the Core specifying a name in its namespace. The naming component of the Core would look up the repository entry associated with this mapping in the client's name space. The repository entry contained, among other things, the destination mailbox for requests to this resource. After further processing, the message was delivered to that mailbox.

When two machines connected, they each set up a proxy to handle requests from the other, as shown in Figure 2. A proxy appeared as a local client to its Core. After mutually authenticating, each proxy made the resources in its name space available to clients on the other machine by sending a serialization of the repository entries. The importing proxy registered the resource in its Repository and listed its mailbox as the target for requests.

A client with a name bound to the repository handle of a remote resource would send a message to the Core specifying that name. The Core would forward the request to the target mailbox, where the proxy would pick it up. The proxy would serialize the request and send it to its counterpart. The receiving proxy would deserialize the request and make a local request to its Core. The Core would then forward the request to the target mailbox where it would get processed. Replies followed the same path.

This approach had a number of advantages. Only the proxies knew anything about the network. The programming model was simplified because other clients and the Core only knew about local messages. Since the basic model was message passing, the failure modes for local and remote requests were the same, although the frequency and latency of failures was different. In addition, the receiving client need not have been the actual handler of the request. It may have been a proxy for another machine. The ability to export an imported resource description was all that was needed to broker requests.

There were also some disadvantages. There was additional latency, since a given request might pass through an unknown number of machines. If any one of them was off line, the resource was unreachable. This problem was mitigated by allowing intermediaries to introduce the machines they connected to, so those machines could make a direct connection.

This architecture addressed many of the requirements of the Global Computer. The CU naming scheme consisted of purely local names combined with pairwise translations, similar to the way we talk about "Nancy's husband's car" in casual conversation. This representation is largely independent of local naming decisions. The example still works if I get a new car or Nancy gets a new husband. Allowing such local renamings without breaking applications is an important part of hiding the network from applications. More importantly, this approach sidestepped all issues related to partitioning a global name space (Section 3.3). A disadvantage was that you couldn't send a

reference out of band, as we do when sending a URL in an email.

A name could be bound to more than one handle, which makes sense for stateless services and replicated files. If a particular binding didn't work, the system would try another one. That's one way we provided a common execution environment (Section 3.1). A program would run on any machine that provided a mapping to it. File replication was a higher level service, but the name bindings made using it transparent to the application (Section 3.2). A name binding could even include a resource description, allowing the binding to be delayed until invocation.

Access control was provided by the mappings in the client's name space (Section 3.5). Each mapping represented a right to send messages to a specific resource. If there was no mapping for a particular repository handle, there was no way for the client to send a message to the corresponding resource. Split capabilities [20] provided additional controls, which allowed us to support *Voluntary Oblivious Compliance* (VOC). Consider a simple request, say Bob asking Alice for a file. If Alice doesn't care about the rules, she will respond. Hence, following the rules is voluntary. Say that Alice wants to comply with the rules. In CU, she could just send Bob a name binding for the file, oblivious of the rules. VOC ensures that Bob can only use the handle if that use obeys policy. Supporting VOC on the Global Computer will be a necessity, since there will be many rules, and they will change frequently.

Since finding things is hard on one computer and harder in the Global Computer, we devised an extensible ontology framework [21] (Section 3.6). A repository entry could have descriptions in one or more ontologies, which would make it discoverable. A search request would return name bindings to the set of resources that matched. Treating the ontologies as resources allowed us to use the naming system and split capabilities to control who could find what resources. It also meant that you could advertise a new ontology in a widely known one. That let you extend an existing ontology when you needed to describe something new. CU also included an external advertising service, based on the same extensible ontology system, for finding resources on other logical machines and defining communities.

With these features, management became just another service. It was possible to export a resource for managing a particular application to anyone providing that service. Local versus central control was a matter of policy rather than architecture (Section 3.4).

We built the Client Utility and measured its properties. Local requests were slow, about 1 microsecond on a 200 MHz processor, the state of the art in 1999. Remote requests were about 1 millisecond per hop. These latencies ruled out fine grained use, but they are orders of magnitude smaller than those measured when using the current web services standards.

Scalability did not appear to be a problem. We ran experiments with as many as 300 logical machines on 60 processors running programs that made trivial requests. At the high end of the test range, latencies increased dramatically, but we determined that the cause was the CPU load. We believe we could have pushed the limits further had we been able to use more physical machines.

A system based on this architecture [21] was delivered as the Beta 2.2 version of HP's e-speak product in December of 1999. That year was the start of the dot com boom and the vision of profitable business to business platforms overwhelmed that of the Global Computer. The architecture was retargeted, with less emphasis on transparency and low latency and more on such business related issues as strong authentication. The new architecture [23] kept the basic structure of CU but changed the security model to use SPKI certificates [24].

The e-speak product was used by five companies to run significant parts of their businesses. The largest configuration consisted of some 40 physical machines to support 400 engineers with access to nearly 10,000,000 resources. To the best of our knowledge a legitimate request was never denied in 18 months of operation, nor were any requests honored that should not have been.

In 2001, HP decided to exit the middleware business. That involved shutting down the E-speak Operation and some other business units. Several groups within and outside of HP continued to use their e-speak platforms for a year or more. In fact, one of them came live more than a year later. Maybe e-speak did what they needed.

5. Getting there

There are numerous technical questions that must be answered in building the Global Computer. However, addressing the social and business issues are more important.

The Global Computer must be open. It won't succeed if it is viewed as giving any company an advantage. HP tried to avoid that trap by releasing e-speak under the GNU Public License, but that wasn't enough. A corporate consortium won't do. Experi-

ence has shown that jockeying for competitive advantage often outweighs technical merit. An organization, perhaps along the lines of Mozilla, will be necessary.

The environment itself needs to be open. Relying on digitally signed content from trusted sources gives large companies too much of an edge. We need to be able to use programs from many sources. One solution would be to enable installing and running a program without giving up full control of our machines. Any changes made by such programs should be reversible.

Whatever gets built must deal smoothly with existing applications. The world simply isn't going to rewrite all its software any time soon. Experience has shown that it is often easy to write a wrapper that will allow a legacy application to run in the Global Computer, getting less than full advantage, of course. Perhaps the simplest thing to do is modify the file system to translate between the legacy interface and whatever the Global Computer supports.

There will be many business opportunities on the Global Computer, selling computing resources, dealing with backups, providing support, and others that haven't been thought of yet. Central to enabling these businesses will be risk mitigation. Credit card companies provide this service for catalog and online sales. Some comparable business, perhaps the existing infrastructure, will be needed.

Dealing with communities will be important. For example, typing "who" on a Unix machine gives a list of logged on users. What happens when you type "who" on the Global Computer? Clearly, we want the result to apply to one or more communities of interest. Communities also provide a convenient way to control where our programs run and where our data resides.

Finally, and most importantly, we need to build it. We can talk endlessly about this feature or that deficiency. It's only through practical experience that we'll ever get a Global Computer we can all use.

6. Conclusions

The Global Computer is a compelling vision, one that has appeared sporadically over the past 40 years. The questions of whether the perceived advantages will have practical value and whether it can actually be built were raised each time it was proposed.

We ran two experiments to answer those questions. The first showed that there are cases where the single machine view provides value to users. The second showed that it may be possible to address the problems inherent in building it. Perhaps the third time will be the charm, and we'll all be on the same computer in another 15 years.

7. References

[1] Karp, A., "The Global Computer", IBM Scientific Center Report G320-3544 (1990), Also http://www.hpl.hp.com/personal/Alan_Karp/publications/global.pdf, I will be quoting freely from this technical report without further attribution.

[2] McCarthy, J., MIT Centennial Speech of 1961 cited in *Architects of the Information Society: Thirty-five Years of the Laboratory for Computer Science at MIT*. S.L. Garfinkel Ed. MIT Press, Cambridge MA, (1999)

[3] Fano, R.M., and Corbatò, F.J. Time-sharing on computers. *Scientific American* **215**(3), Sept., pp. 128–40 (1966)

[4] Dertouzos, M. The "information marketplace" in electronic mail and message systems. In *Proc. AFIPS Workshop on Technical and Policy Perspectives.* Washington, D.C., Dec. (1980).

[5] Foster, I., *The Grid: Blueprint for a New Computing Infrastructure, 2nd Edition*, Morgan Kauffman (2004)

[6] Erl, T., *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall (2005)

[7] Anderson, D. P. and Kubiatowicz, J., "The Worldwide Computer", *Scientific American*, March, pp. 40-47, (2002)

[8] Miller, M., private communication (2005)

[9] Popek, G. J. and Walker, B. J., *The LOCUS Distributed System Architecture*, MIT Press, Cambridge, Massechusetts, (1985)

[10] Walker, B. J. and Richard M. Mathews, R. M., "Process Migration in AIX's Transparent Computing Facility", *IEEE Technical Committee on Operating Systems*, 3(1):5--7, (1989)

[11] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Wetley Weimer, W., Wells, C., and Zhao, B., **"OceanStore: An Architecture for Global-Scale Persistent Storage**", Appears in *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, November 2000

[12] BitTorrent, http://www.bittorrent.com/introduction.html

[13] PlanetLab, http://www.planet-lab.org/

[14] Open Group, *CDSA Explained, An indispensable guide to Common Data Security Architecture*, The Open Group, (2001)

[15] Karp, A. H., “Authorization Based Access Control for the Services Oriented Architecture”, Fourth Intl. Conf. on Creating, Connecting and Collaboration through Computing (C5), Berkeley, Ca, January (2006)
[16] Google, http://desktop.google.com/
[17] UDDI, http://www.uddi.org/
[18] Karp, A. H., video of Client Utility prototype, http://www.hpl.hp.com/personal/Alan_Karp/espeak/ClientUtilityDemo.rm (1996)
[19] Karp, A. H., "E-speak E-xplained", *CACM*, **46**, #7, pp. 113-118, July (2003)
[20] Karp, A. H, Rozas, G., Banerji, A., and Gupta, R., "Using Split Capabilities for Access Control", *IEEE Software*, **20**, #1, pp 42-49, January (2003)
[21] Kim, W. and Karp, A. H., "Customizable Description and Dynamic discover for Web Services", *ACM Conference on Electronic Commerce (ACM EC'04)*, New York, June 2004.
[22] Karp, A. H., Rozas, G., Banerji, A., and Gupta, R.,"The Client Utility Architecture: The Precursor to E-speak", HP Labs Technical Report, HPL-2001-136, June (2001), also http://www.hpl.hp.com/personal/Alan_Karp/cuarch/arch_overview_TR.html
[23] E-speak, *Architecture Specification, Release A.0*, http://www.hpl.hp.com/personal/Alan_Karp/espeak/version3.14/Architecture_3.14.pdf (2001)
[24] Ellison, C. M, Frantz, B., Lampson, B., Rivest, R., Thomas, B. M., and Ylonen, T., “SPKI Certificate Theory”, RFC 2693, (1999)