The Global Computer

Document Number G320-3544

January 10, 1991

Alan H. Karp

IBM Scientific Center
1530 Page Mill Road
Palo Alto, CA 94304
(KARP at PALOALTO)

Abstract

Imagine having access to more computer power than in hundreds of supercomputer centers, having available more memory than most computers have disk space, having more disk space than in most tape libraries. Further, imagine being able to travel anywhere in the world, sit down at any available terminal, and have your entire environment available – profiles, mail, configurations. Imagine, now, how nice it would be if you could access all your files as quickly as if you were at home. Such a system is not an idle dream; we are building it today and have part of it running now.

Index terms: Network computing, distributed computing, collaboration

Introduction

Until recently programs were written by people in a single location. Now, however, networks have made it possible for collaborators to be spread out over the entire globe, and it is not unusual to find papers listing authors on three continents. Unfortunately, it is hard to collaborate when people are on different computers. The problem is so severe that the NSF is funding research to address the problem.[2]

Since it is easier to collaborate with someone on the same computer than with someone on a different computer, the solution is obvious.

> *Put everyone in the world on the same computer.*

This distributed operating system spans the world so let's call it the **Global Computer**. I hope to convince you that the Global Computer is no idle dream. In fact, we are beginning to assemble it today.

A good example of how the Global Computer should work is the way printers are handled in most Unix shops. A user simply says `print file` and the file magically appears on a printer. There is no need to know which machine controls the printer, nor is there any need to move files around the network. All the details are handled by the system. Each user acts exactly as if the printer is locally connected. If the Global Computer is implemented correctly, all computer services – printing, executing, accessing files – will be this easy to use.

What are some of the advantages of the Global Computer? First of all there is the aggregate resource available. Each user will have the computer power of hundreds of supercomputers, more physical memory than most computers have disk space, and more disk space than in many tape libraries. In addition, there will no longer be a problem of incompatible versions of software since all updates can be centrally managed. Backups can also be provided at a central location with special facilities to handle the volume of data.

The user will see some less tangible benefits as well. Imagine traveling to another city and being able to sit down at any terminal, log on, and have access to all your files. Since there is only one computer, you will not have to beg access to someone else's account so you can remotely log in to your home computer. Since you are doing all your work in your own account, you have your own environment with things configured the way you like. Furthermore, if the Global Computer is implemented properly, you will be able to access your data as quickly as if you were home.

System support services will also improve. Since the Global Computer spans the world, it will always be daylight somewhere. Thus, systems support personnel working first shift in Brazil can help a user running in the middle of the night in Manila. When they fix a piece of broken software, it will be available to the user automatically. Language differences will be a problem. Perhaps the Global Computer will spur the development of automatic, on-the-fly translation.

One thing the Global Computer is not is a panacea. There are problems collaborating with people even when everyone is on the same machine; the Global Computer will do nothing to alleviate these problems. In fact, the success of the Global Computer will be measured by how well it reduces the problem of collaboration to exactly this set.

Application Development

The applications that are of interest to users of the Global Computer typically consist of tens or hundreds of thousands of lines of code, hundreds of subroutines, and a number of data files, some quite large. These programs have had many contributors, most of whom have moved to other locations. For example, it is not unusual for a university professor to have a single application program worked on by many generations of graduate students. Each makes some improvements and then moves on. After a while there is no one who understands more than a small part of the code.

The problem is exacerbated by the trend toward interdisciplinary research. Not only does each individual not know much about most of the program, no individual even understands much of the underlying science.

Consider the example of a climate modeling code. A few contributors will be experts in the microphysics; others, the macrophysics; someone else, the numerical methods; and yet another person, the visualization techniques. The problems of maintaining and upgrading the program are bad enough if everyone works on the same computer; they are magnified when the collaborators must work on different machines.

Look at what it takes to improve an existing code. One of the contributors will edit a few routines, rebuild the executable, run some small tests, and examine the output. This procedure will be repeated many times until the researcher is convinced that the changes will have the desired effect. Next, the new code will be run through the test suite to check both correctness and performance. The test suite will also be run several times, typically to tune the code's performance. Often, bugs or a serious performance problem will require going back to the small test runs one or more times.

Now look at the problems working on a major, interdisciplinary program with contributors from many locations. First of all, each person will almost certainly have a complete copy of the code, the required input data sets, and the output from at least a few runs. Since there are multiple copies of the code, researchers will have to reconcile differences periodically.

For example, in order to improve the microphysics in a climate code, it might be necessary to know how much methane is in the air. This change will almost certainly require a change to the input data and corresponding routine. The next day, another researcher may be improving the macrophysics by including, say, the effect of ocean waves on the radiation flow. If

this change also requires a modification to the input data format, there will now be two, incompatible versions of the application.

This problem is much more manageable if everyone is working on the same computer. Since it is no harder to access a shared copy of the input routine than it is a private version, it is more likely that both users will work from a single copy. There will be no conflict unless both researchers try to make the changes at the same time. The problem is much more difficult when there are several copies floating around, and people are out of touch with each other. For example, software management systems are much more likely to be used if everyone is on the same computer. It is just too tempting to update a local copy than to get the "official" version from a remote machine.

The same problem arises when the small tests have been completed and it is time to run the test suite. While the small runs can be made on each person's local computer, a comprehensive test suite will require a supercomputer. Each run can take several hours of machine time, and several runs are typically made. Two researchers at different locations may each be trying to run incompatible new versions on the same supercomputer. While there are mechanisms to avoid collisions, such as using separate directories for each individual, everything is much simpler if all the work is done on the same computer.

What is a Computer?

It is clear that no conventional computer architecture can be used to build the Global Computer; it will by necessity be made up of a network of machines made to look to the user like one computer. If we are to succeed, we must agree on what the applications programmer thinks is a computer.

I contend that a computer is really a simple device when viewed from the outside. It consists of

1. A means of talking to the machine; a keyboard.
2. A means for the computer to talk to us; a video display.
3. Something to do the work; one or more CPUs.
4. Volatile storage; both real and virtual memory.
5. Nonvolatile storage; magnetic tapes and magnetic and optical disks.
6. Software
   (a) File systems so users can name their files.
   (b) Command environments so users can manipulate their files.
   (c) Text editors for creating and modifying files.
   (d) Compilers for turning source files into executable files.
   (e) Applications for turning input files into output files.

If we can present these pieces to the users, they will think they have a computer no matter what is under the covers; only their perception of the quality of the computer will be affected by our implementation. If we do it poorly, our machine will be awkward to use and slow; if we do it well, users will not know or care that they are running jobs on a distributed machine.

Key Building Blocks

There are three key pieces to a good implementation of the Global Computer – a global name space, a common execution environment, and file replication with caching. Without the first two, we won't have a Global Computer; without the third, no one will be willing to use it until networks become considerably faster than they are today.

Global Name Space

One of the problems with working with a variety of systems is finding the files you need. A global name space on the Global Computer simply means that you (and your application) will be able to find any file no matter what machine is being used. In other words, the absolute path name of any file is knowable. This simple feature makes it possible to run any program on any machine since the program and any input and output files can always be found. In addition, procedures that build applications, *e.g.*, `makefile`, can run on any machine.

I include in the concept of a global name space that of uniqueness; there is only one version of any file. While it would be possible for different machines to have different versions of a file at the same location in the file system, the Global Computer requires that all copies be identical. If they are not, the user will need to know which version is on which machine, and one of the key features of the Global Computer will be lost.

Since each file is unique, a global name space means that software updates can be made where appropriate. For example, the Global Computer could have a group in Nigeria responsible for the C compiler and a group in Norway handling Fortran. Other groups in other locations could maintain application specific packages. The point is that, once a module is updated, all users on the Global Computer see the new version. Backups can also be handled easily at any location, particularly those set up with special devices to handle the volume of data.

A global name space also makes working while traveling easier. Today you must beg for a user id on the local machine or for someone to give you access to an account so you can log in to your home system. Even if you find a way to get to your home machine, the network delays are a problem since all commands have to traverse the network. With the Global Computer you don't have these problems. Since there is only one password file, you need only find an unused terminal to log on. All commands are executed on the local machine. Further, if file replication is done properly,

all the files needed will be on locally connected disks.

Common Execution Environment

A common execution environment means that a program can be run on any of the machines making up the Global Computer that meets the program's requirements. If the Global Computer is to be truly useful, it will consist of machines from a variety of vendors, preferably all of them. In such a heterogeneous environment a program that requires Cray vector registers will be able to run only on those machines. Since we don't want the user to have to know all the requirements of the job, there will be a mechanism for attaching this information to each program. If an application has been enabled for a variety of machines, the system will be capable of finding a suitable machine and starting the corresponding version.

Other things are possible with a common execution environment. Since the Global Computer appears to be one machine to the user, we should be able to use all the tools we would use on a single machine. In Unix we can fork processes and pipe output between them. A common execution environment will allow us the same freedom across physical machines. Thus, parallel processing of applications will be just as easy (hard?) as if done on a single computer.

Another nice feature of a common execution environment is the ability to do dynamic load balancing. Jobs can be scheduled to run where they will complete the fastest. For example, consider a code enabled for both a vector supercomputer and a high performance workstation. If the supercomputer is heavily used, and the workstation is idle, the scheduler can send the job to the workstation. The user will not need to know where the job ran.

Consider also an application that takes many hours of CPU time. It might be started at night when there are few users on the system. However, if it does not finish by morning, it will either interfere with interactive users or be pushed so far into the background that it will make little progress during the day. With a common execution environment we should be able to find a computer somewhere else in the world that is lightly loaded, presumably because it is the middle of the night there. We can then migrate the application to that machine. I like to call this *heliophobic* computing, *i.e.* "Keep your applications in the dark."

Of course, there will be times when the user will want more control. For example, the user may want the job to run on a specific machine, perhaps because 20 GBytes of non-replicated data are known to reside there. The Global Computer will provide for this requirement, perhaps automatically as part of its scheduling algorithm.

It is unfortunate that the actual hardware always shows through our beautiful systems software, but any time we want to optimize something, performance for example, we become aware of the components of the system. The Global Computer is no exception. (Even Unix print services occasionally require the user to di-

rect a file with special characteristics to a specific printer.) The point is that a user *may* treat it as a single system; the extra control will be provided for those who need it.

There are other advantages of a common execution environment. Systems people can migrate running jobs to other machines before taking a system down for maintenance. If a machine is lost, only a small part of the aggregate resource becomes unavailable. Also, as machines join the Global Computer the users see more and different resources made available without needing to do anything special to use them. Since there will always be a computer available somewhere in the world, a business with a critical application can rely on the Global Computer to handle most of its disaster recovery.

File Replication and Caching

The hardest part of making the Global Computer interactive response time sufficiently short is dealing with network delay, both latency and transfer time. The best way to hide network delay from the user is to make it occur when the user is doing something else. This is where file replication comes in.

Since the Global computer has a global name space, and there is a unique version of each file, that file can be copied to other sites. The copying can be done either when the file is modified or when it is requested. If copied when needed by a remote user, that user will pay the network delay which can be quite long if the data is being moved a long way. (If the file is cached, the user only waits on the first access.) Thus, the only way to provide fast enough access to data, aside from greatly improving network speeds, is file replication.

File replication implies that a current copy is sent out to other machines each time the file is changed. Since most files are not used interactively for a long time, say several seconds, after they are modified, users will rarely have to wait for the network to deliver the data; it will be there by the time they ask for it. Of course, the operating system is responsible for maintaining data integrity.

It is clear that we don't want a copy of each file on each machine in the Global Computer. Instead, we want to be able to specify a replication list for each file or group of files. Thus, groups working on a project would share a set of files. Other groups working on other projects would share another set of files. Key files like compilers, system commands, text processors, *etc.* would be replicated to all sites.

We might see a structure like that in Figure 1. A person on machine A might be collaborating with someone on machine B; someone else on machine B might be collaborating with people on machines C, D, and E; yet anther person on machine D might be working with a group using machines F, G, and A. This figure shows only the selectively replicated files; there will also be files, like compilers, replicated to all sites. In addition, any user on any machine can access any file simply by naming it; the only difference from accessing a

replicated file will be a delay in getting the data.

File replication involves some trade-offs. It can reduce network traffic by having copies of commonly used files close to the machines using them. For example, each machine in the Global Computer will have a copy of the C compiler even though it will typically be modified only at a single site. On the other hand, replication can increase network traffic. It is not uncommon to modify a single routine many times while upgrading a code. All these changes will be sent across the network even though only the last one need be.

At first glance, disk space is another concern; all those copies of all those applications will take up space. In fact, though, I believe the increase in disk space will be small. Remember our application development scenario? Each collaborator almost certainly has a complete copy of the code and data files. Replicating them simply means that everyone sees the same version, but there will be no net increase in disk space needed.

It is also possible that the Global Computer will lead to a reduction in the amount of disk space used. Since there is a global name space, it is quite easy to access a non-replicated file on another system. If users must pay for disk space, they may choose to keep infrequently accessed files, such as archival output, at a single site. All they will need do is wait a few seconds for the data to come across the network. If the file is then cached locally, they will only have to pay the delay every once in a while. Thus, multiple copies of large files may become unnecessary.

File replication can provide automatic disaster recovery if the replication list is chosen properly. The loss of a single machine will not result in the loss of any replicated data. Nor will access to the replicated data be lost if one machine is down. If replication is handled properly, it can be used to eliminate much conventional backing up; only archival data need be transferred to more permanent storage.

Evolution

Tools available today, particularly those shipped with Unix, have both made long distance computing a reality and led to the current state of affairs. Without these tools, long distance collaboration would not be widely used enough to be a problem; with these tools, we are exposed to all the problems discussed earlier. There is an evolution toward the Global Computer that follows the identification of problems.

TCP/IP

Although there are other examples of network computing, those based on Unix with TCP/IP are the most widely used. TCP/IP provides tools for logging in to a remote machine, `telnet` and `rlogin`, moving files between machines, `ftp` and `rcp`, and executing procedures on remote machines, `rsh` and `rexec`.

These are the tools that made network computing practical. The distinct name space on each machine, the need to explicitly copy files between machines, the

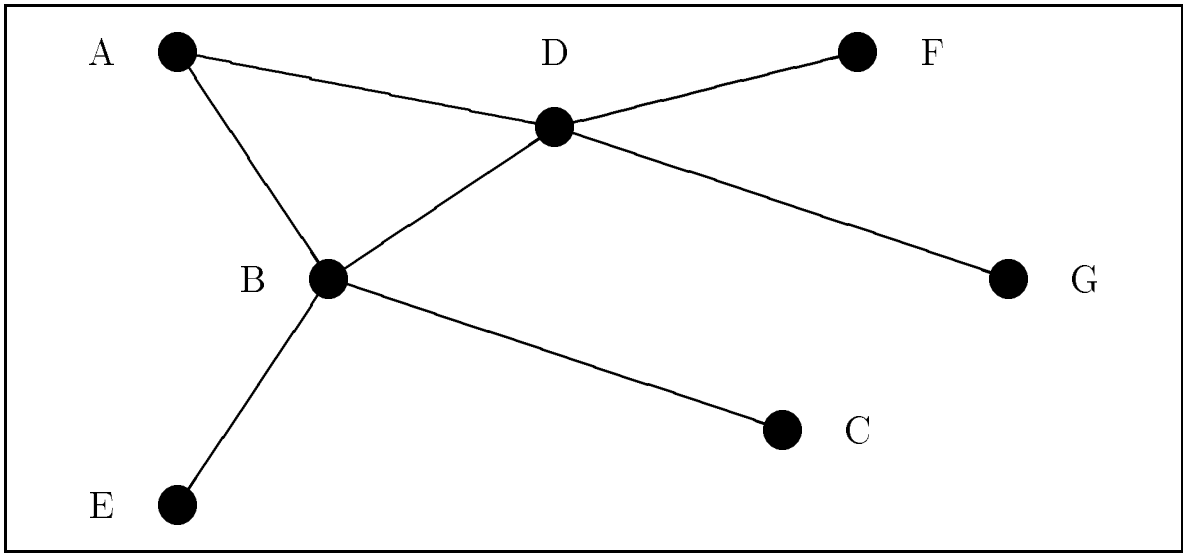

Figure 1: Possible file replication pattern in a large cluster. Individual files can be replicated at the user's convenience. A user on machine B would replicate some files to a collaborator's machine, say D. Another user on B might replicate some other files to collaborator's machines E and D. In this way, a large network of interlocking replications will be built up.

inability to hide network delays, and the lack of version control for application code make collaboration across the network difficult.

NFS

Network File System (NFS) was built to address the problems exposed by doing network computing with TCP/IP. A file system or directory may be mounted at any point in a user's hierarchical file system. Once the mount has been done, all files on the remote system are accessed exactly as if they were local. Programs that exist only on the remote machine can be run (if the binary is suitable for the host), and data files are available to applications.

Although NFS addresses many of the problems of network computing, it is not a complete solution. A global name space can be maintained only by convention since each user can mount a given remote directory at a different point in the hierarchy. This problem makes it hard to write both applications and `makefile`s since the absolute path names are not knowable ahead of time. In addition, the programmer must explicitly take some action to run a command on another computer, *i.e.*, `rexec`. While not a serious problem, it is

just one more thing that gets in the way of applications work.

NFS was not designed to connect systems on different continents with network latency measured in seconds. Every time a remote file is accessed, it is sent across the network. While not often a problem for locally connected workstations, network delay can be a serious problem for geographically dispersed systems. Most users will balk at a 5 or 10 second delay every time they attempt to edit a file. These delays will lead them to copy files to their local systems, defeating the purpose of creating NFS in the first place.

Network delays raise an interesting question for those people using a remote supercomputer. Where do you keep the source files? If you keep them close to the programmer, the supercomputer will pay the network delay on every compile; if you keep the files close to the supercomputer, the programmer pays the network delay on each access. Some attempts are being made to address this issue, such as the file caching used by the Andrew File System.[3]

TCF

Making files easily accessible from many machines led people to try using remote machines to do their work, but they found it inconvenient. The need for more transparency was translated into the LOCUS Distributed System Architecture[5] which became the Transparent Computing Facility (TCF) delivered as part of the IBM AIX product. It is the facilities of TCF that led us to the idea of the Global Computer.

TCF incorporates the three key features needed by the Global Computer – a global name space, a common execution environment, and file replication with caching. As described later, we have implemented a system based on TCF as a prototype for the Global Computer.

There are a number of shortcomings, though. The current design limits the number of machines in the *cluster* to 31 nodes. Also, there is only one superuser password for the entire cluster. While this is not a serious problem if we are clustering a few supercomputers within a single company, it does make life difficult if the cluster includes individually owned workstations or supercomputers from different enterprises.

A more serious problem is the way files are tagged with the resources they need to run. Today, programs only say if they can run on a PS/2 or a S/370 machine. There is no way to indicate that a vector facility or special PS/2 board is needed. This problem will be even worse on a Global Computer that is made up of machines from many vendors.

Increasing the number of nodes in the cluster is simple to fix. It should also be possible to come up with a scheme for having a separate superuser password on each machine. Deciding what information is needed to allow the system to find a suitable machine for every job is more difficult. There are simply too many machines with too many different architectures and configurations. Some sort of extensible defi-

nition will be needed.

Others

There are many projects attempting to make network computing easier.[3][1] Almost any one of them can be used to implement the Global Computer. Unfortunately from our point of view, none of these projects has been used to build a system as large and comprehensive as we would like to see. Further, while each has important pieces, none is complete enough to make the Global Computer a reality. I hope this discussion will encourage the developers of these systems to address the problems of global computing, particularly supercomputing.

Management

There are no technical barriers to building the Global Computer. If there is anything that will prevent it from becoming a reality it is management issues. In fact, the technical part is quite easy when compared to questions of security, chargebacks, and control.

Security

The Global Computer raises some serious security issues. With machines spread out over the world, how will users be authorized and how will their access be terminated? How can we protect proprietary data from accidental or purposeful exposure? How can we prevent unauthorized users from accessing the Global Computer and stealing resources or planting viruses?

In spite of the fact that a geographically dispersed machine is harder to protect than one in a secure environment, I contend that security on the Global Computer will be better than we have today. Look at the Internet worm that crippled a large number of computers[6] or the penetration of U.S. Government computers by German hackers.[7] In the former case, known security holes in Unix were exploited to plant the worm. The latter security problem was uncovered by accident by someone worried about small accounting discrepancies.

The main reason that the security holes exploited by the Internet worm were not plugged is the inability of many sites to provide adequate support. With thousands of machines, many used by only a few people, the majority of installations have no security procedures or only part time security people. While large machines at major installations have security staffs, even they find sharing information difficult. The German hackers exploited another problem that arises when there are lots of machines; small organizations don't want to devote more than minimal resources to accounting.

If there is only one computer for the entire world, it will certainly have full time security and accounting organizations. The Internet worm would not have penetrated the Global Computer because this staff would have fixed any known holes immediately. Furthermore, all systems would have been protected; there would

be no need to communicate the changes to every Unix shop in the world.

The accounting group would have a similar mission. The Global Computer will be able to afford a professional staff that knows how to do computer accounting, how to insure that audit trails exist, and with the mission of discovering any anomalies.

The Global Computer will be a very attractive target for hackers. Not only will the resources available be worth stealing, but the challenge of going after such a large configuration will be a real incentive. Unlike many systems with only minimal thought given to security, the security staff of the Global Computer should be able to put up a good fight. Of course, a system as large as the Global Computer will have its own set of new vulnerabilities. Security work should be quite interesting.

Charging

Most computer centers in the world are run on a cost recovery basis. They provide a system that users are expected to pay for. Often these payments are funny money, accounting tricks used to allocate the costs of the system to the users. Many other installations insist that real money be used to pay for resources.

As long as all the computers you use are owned by your organization, charging is merely difficult. When you start using someone else's computers, the problem becomes intractable. Either you come up with real money or you arrange for funny money.

The whole point of the Global Computer is to make it easy to use someone else's machine. Two tacks can be taken. Users can be limited to use only certain processors and disks in the Global Computer. They will still have access to all the files they need, but their jobs will only run on a subset of the machines. This subset could be as small as their personal workstations or as large as the entire system. As now, users would be responsible for paying real money for the resources they use or arranging, on a case by case basis, access to the processors they want to use.

Another approach is to treat the Global Computer as a utility. Billing will be handled much the way long distance charges are today in the U.S. Users will receive a bill itemizing their costs for the resources they used and would make a single payment to their local system. The local system would then disburse the funds. Each system participating in the Global Computer will receive an allocation that represents its contribution to the total. (A Macintosh might receive one Cray minute per year.)

I hope that a scheme could be devised to encourage the use of lightly loaded machines. For example, if I run a job on a specific machine, I get charged at the full rate; if I run the job on the least heavily loaded machine, I should get a discount. Similarly, I would hope that a priority scheme would be put in place to allow users to get discounts for running at low priority. In fact, a 100% discount for running at the lowest priority would make it possible to do some interesting but un-

fundable work without adversely affecting others.

Control

Problems of security and charging are minor when compared with the question of control. Will Berkeley let Stanford control part of its computer? Will Yale let Harvard add users to its system? How will Israel feel about Syria allocating space on Israeli disks? If there is an issue that will limit the universality of the global computer, control of resources is it.

I have no answers to this question. Clearly, each participating organization will want some method to reserve part of its resources for its own people. Further, as much as we all hate to see it, organizations will allow open access to their systems *except* for certain others. Of course, computers owned by the Computer Utility will be much easier to manage; I hope this paper will stimulate discussion on how best to resolve this issue.

Virgo SuperCluster

It has been said that, "A job worth doing is worth doing badly."[4] which means, get on with it and fix what you don't like later. We have taken this advice and are building a world-wide system within IBM using the available software, AIX with TCF. It is only by doing it wrong that we will find out how to do it right.

Configuration

We currently have three sites on our Virgo SuperCluster, the Palo Alto Scientific Center (PASC), Almaden Research (ARC), both in northern California, and Yorktown Heights Research (YKT) in New York. PASC is running AIX enabled to use 3 processors of its 3090-30E/VF configured with 256 MB of memory; YKT is using 2 processors of a 3090-50S/VF also with 256 MB of memory for AIX; ARC has allocated 16 MB of memory on a 3090-180. Although we could have implemented this cluster using the AIX product currently being shipped by IBM, Virgo is implemented using an experimental version that incorporates several extensions useful for supercomputer users.

Since projects like this are easier to get going if we don't tell management about them, funding is quite limited. A consequence of this decision is the rather strange network configuration shown in Figure 2. It is an example of what you end up with by piggy-backing on someone else's network.

Each 3090 has a channel connected to a PC/AT class machine. The AT is connected to an Ethernet which, in turn is connected to another AT which connects to a T1 line. This configuration is repeated on the other end of the T1. Since we have no direct connection between Palo Alto and Yorktown, we must pass through 8 PC/AT class machines with a delay of about 0.25 seconds per machine. In addition, the T1 line between Almaden and Yorktown is shared with other traf-

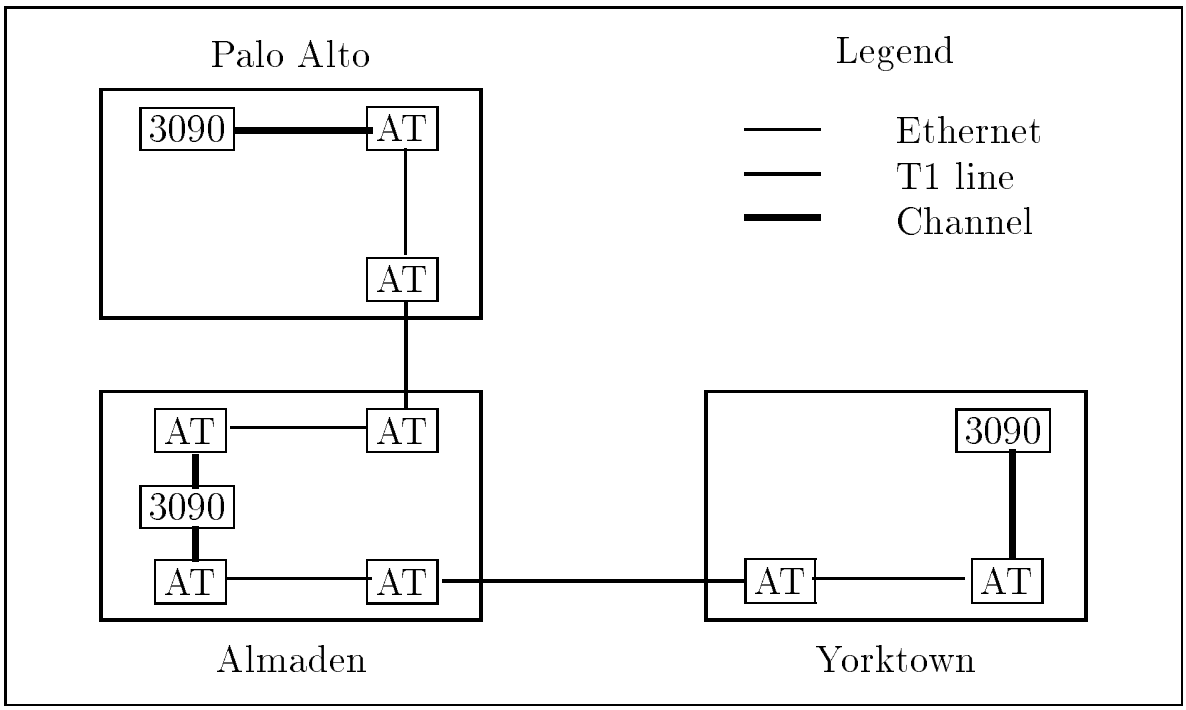

Figure 2: Virgo SuperCluster network configuration. In order to avoid management involvement, we connected the machines by piggy-backing on existing networks. Each PC/AT class machine adds about 0.25 seconds to the latency. Further, the T1 line between Almaden and Yorktown is heavily used for other purposes.

fic. Measurements indicate that our cluster traffic is getting about 10% of the rated bandwidth, about 20 Kbytes per second.

We have structured our file systems as shown in Figure 3. Each machine has a number of local file systems denoted by the short lines. Shared file systems include the root and system commands, `/`, `/usr`, `/bin`, *etc.*. There is also a file system replicated between Palo Alto and Yorktown. We have not yet found it necessary to use file-level replication.

One person in Yorktown is responsible for supporting Virgo. The only other help comes from someone in Palo Alto who helps out on a time available basis; there is no one at Almaden Research supporting this SuperCluster. We have had no problems of security since Virgo is completely within IBM. In addition, problems of charge backs and control have not arisen because we have been able to keep management out of the loop. So far, everything has been handled on a good faith

basis. My only hope is that IBM upper management does not read this Journal.

Experiences

We have had very positive experiences using Virgo. (If we hadn't, this paper would not have been written.) Our successes with Virgo lend support for the viability of the Global Computer. There have been a number of cases in which we found it easier to do our work on the SuperCluster than using more conventional methods.

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

Another occasion where the SuperCluster made our lives easier occurred while preparing for a customer benchmark. Most of the work was being done on a 3090-E in Palo Alto, but the bid was to be for a 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 Virgo, 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 even when the network was down, and it was easy to compare the performance of the two machines. (We won the benchmark.)

One other example of the use of Virgo involved parallel processing. One of the Palo Alto researchers has 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. One case 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.

Lessons Learned

In spite of the poor network we are forced to use, I almost never see any network delays. I now believe that, while faster networks are better than slower ones, almost any network connection will be acceptable. (A 100 baud line may be pushing things just a bit. Is there anybody out there willing to try it?)

In one test, we took a large memory program and put in an immediate exit. Running it locally took less than half a second (measured with `time a.out`). Next, we put the file in a nonreplicated file system and ran it on the Yorktown system, `time on 1 a.out`, and found that the job took 11 seconds, a data transfer rate of 1 MByte per minute. Next, we installed the job in a replicated file system and ran it on the

Yorktown system; it finished in under 3 seconds, most of that time being network latency. The fact that the network transit time is about 10 seconds was confirmed by changing the file and immediately running it on the remote system. We have been working with this code for several weeks now (with the immediate exit removed, of course) and almost never become aware of the network delays.

We have made heavy use of the remote system services. Remember, we have no official support in Palo Alto. When our part time person went on vacation, we were able to continue working using support from Yorktown. In fact, those of us who often show up before normal working hours appreciate being able to get help when we need it. Now, if we can only bring Japan on line, the night owls will be able to get help, too.

The redundancy provided by file replication also proved useful. The day of an important customer demonstration one of our system's programmers inadvertently deleted our AIX system. We did not have to wait for it to be restored; the person running the demo simply used a workstation to log on to the Yorktown machine and ran the demo from there. Not only was the program and its data available, but the single password file meant that a valid login was ready to go.

We have also found some things that need fixing. Each replicated file has a primary site and one or more secondary sites. Users can only update the primary site; the system controls the copies. Unfortunately, writing a file with a remote primary site takes just as long as if the file were not replicated. We have been informed that this delay is a *feature* which we hope to convince the developers to remove. There is a similar delay when examining the directory using `ls` which we have been assured is a bug. The fact that there is a single superuser password for the entire cluster has not been a problem for Virgo but has been one for our other cluster of PS/2s.

The current method for updating files in TCF presents problems. There is one *primary* site that holds the writable version; *secondary* sites hold the replicated copies. If the primary site is unavailable, users can only read their files; they can not update them. A replicated *backbone* file system can be converted by a superuser to be a new primary site if need be, but user replicated files will be read-only until the primary site rejoins the cluster.

Other problems are more interesting. The first time I tried to do a `make` with files from two different file systems with different primary sites, the system kept recompiling files I had not changed. The problem, of course, was that the time stamps were set using local time, a difference of three hours. We now have daemons running to keep the clocks synchronized. Furthermore, each user on the system sees time stamps as local time.

A more difficult problem is vector jobs. TCF was designed to cluster PS/2s with a single mainframe, not to cluster vector processors. Thus, the header information in each program tells only if it is for a 386 machine or a 370 machine. Vector jobs

need to know whether a given 3090 has a vector unit and how long the vector registers are. When the user asks the system to schedule the job to the least heavily loaded system on which the program can run, the system currently assigns vector jobs to any 370 machine regardless of whether it can execute vector instructions. Thus, jobs started this way can fail. A similar problem prevents us from migrating vector jobs between machines with different vector register lengths.

Future

We are pursuing the SuperCluster concept on three fronts, within IBM, outside of IBM, and through standards committees.

We hope to expand Virgo by adding IBM sites as quickly as we can to learn as much about the problems as we can. Candidates include the IBM European Center for Scientific and Engineering Computing in Rome, Italy, the benchmarking center in Gaithersburg, Maryland, and the Dallas code porting center. We also plan to add sites *ad hoc* as Virgo users start collaborations with other IBMers.

If we can build a large cluster we can study such things as how people choose to replicate files, and how much disk space and network traffic all this generates. It is clear that we will not be able to replicate every file to every site, but it will be interesting to see the actual replication patterns. These patterns may be quite different if we require users to determine the replication list or do it automatically based on usage. My best guess is that we will see something like the pattern shown in Figure 1, but we won't know until the cluster is large.

We are also looking for candidates outside of IBM who we can help form SuperClusters. Today, we only know how to cluster machines running IBM AIX. This has limited our search but we have found a multinational oil firm, a major environmental research project, a state university system, and a state government agency.

We also want to make sure that the Open Software Foundation incorporates the parts needed to build the Global Computer. For example, the original proposal for OSF/1 did not provide for a common execution environment. Fortunately, a mini-RFP was put out later and sufficient hooks will be in place so that any vendor that wants to support clustering will be able to do so.

Other work is being done to make working on the Global Computer even more like having everyone in the same place. We have some experimental code that allows multiple machines access to the same X window. Anyone looking at this window can grab the *chalk* and modify the window's contents. We will also be experimenting with voice boards so that people in different locations can talk using the Internet while looking at the same screen image. (We have no idea what packetized speech will sound like.)

Conclusion

Scaling a distributed operating system to support a world-wide community is a difficult problem than has yet to be solved. Providing location transparency and replication transparency are also problems that have not been completely solved for large systems.[1]

I believe that the best way to address these points is to start building the Global Computer and fix the problems as they arise. There are no insurmountable technical barriers, only social ones. If we can overcome these problems, our productivity will improve. By making long distance collaboration significantly easier than it is today, we will see, not just an expansion of the collaborations in place today, but entirely new ways of working. We will also see people addressing problems they would not even have considered in the absence of the Global Computer.

We will have to find the answers to some difficult questions. How much support staff will be needed and how will it be distributed? How will security be managed? What about language differences? Must all machines run the same operating system? Is there a danger of a "Big Brother" coming to power using the Global Computer? There are also some more mundane questions. For example, if everyone in the world is truly on the same computer, what will happen when you list the users on the system, *e.g.*, when you type `who`?

Although the Global Computer will be built to address technical problems, it may also help to alleviate social ones. By removing borders, people will concentrate more on shared interests and less on the national origin of the ideas. Who knows, romance may even bloom as evidenced by a recent marriage between two IBMers who fell in love via electronic mail.

[1] Quoted from the referee's report.

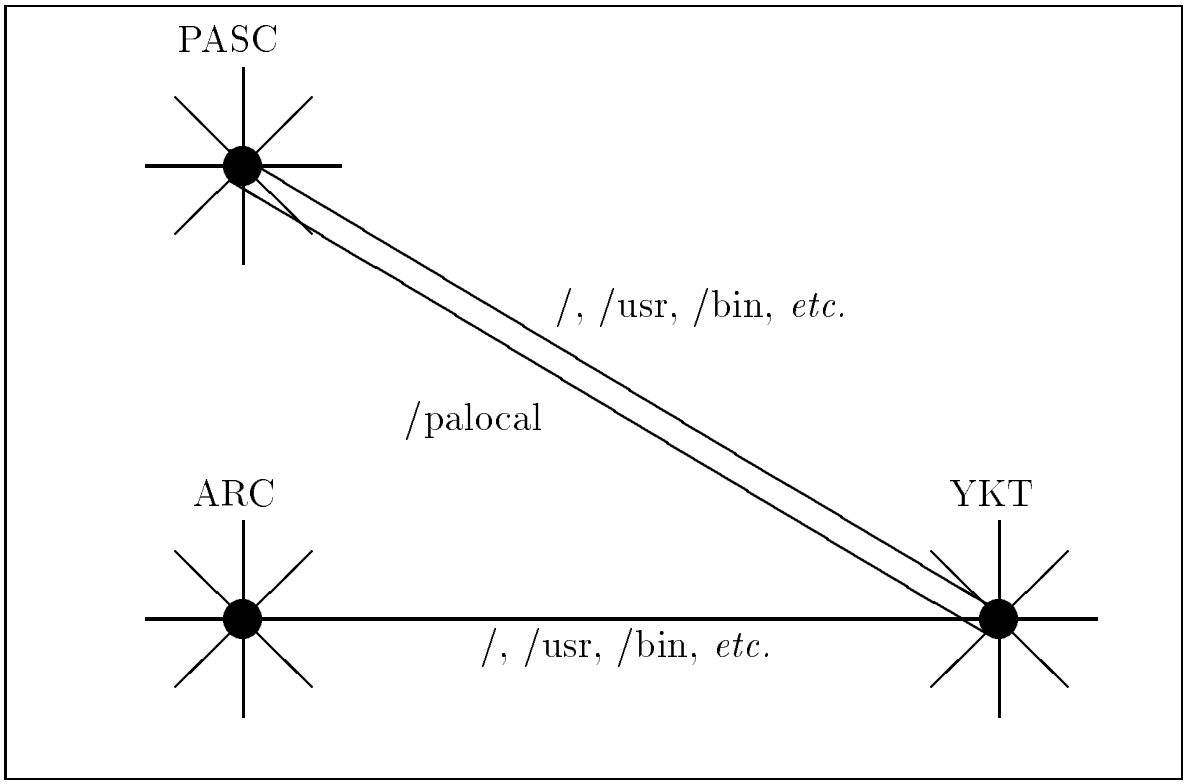

Figure 3: Virgo SuperCluster file systems. All system files contained in directories like `/bin` and `/etc` have their writable copies at Yorktown (YKT) and are replicated to the Palo Alto Scientific Center (PASC) and Almaden Research Center (ARC). There is one file system, `/palocal`, with its writable copy in Palo Alto replicated only to Yorktown.

Acknowledgements: I got the idea for the Global Computer from R. J. Creasy. F. Tung has given the Virgo project management support, but not so much as to impede progress. S. Handelman of Yorktown and B. Lint of Palo Alto have provided system support service for the Virgo SuperCluster. I would also like to thank H. P. Flatt, R. Laprade, R. Nevin, and P. Newman for their help in improving the manuscript.

References

[1] G. A. Champine, D. E. Geer Jr., and W. N. Ruh. Project athena as a distributed computer system. *IEEE Computer*, 23(9), September 1990.

[2] Towards a national collaboratory, report of an invitational workshop. Rockefeller University, 17-18 March 1989.

[3] Special issue on operating systems. IEEE Computer, May 1990.

[4] Thomas Peters. San Jose Mercury News, December 3 1990.

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

[6] E. H. Spafford. The internet worm: Crisis and aftermath. *Comm. ACM*, 32(9), September 1989.

[7] Clifford Stoll. Stalking the wily hacker. *Comm. ACM*, 31(5), May 1988.

Trademarks

A number of trademarks and registered trademarks have been used in this paper.

Unix is a registered trademark of AT&T.

Network File System and NFS are registered trademarks of Sun Microsystems.

Transparent Computing Facility, TCF, PS/2, and AIX are registered trademarks of the International Business Machines Corporation.

OSF is a registered trademark of the Open Software Foundation.