M.I.T. LABORATORY FOR COMPUTER SCIENCE August 25, 1980

Computer Systems Research Division Request for Comments No. 196

SOME THOUGHTS ON DISTRIBUTED SYSTEMS

by David D. Clark

The attached was prepared for an upcoming conference on distributed systems. As the title of this document series might suggest, it is being distributed for comment.

This note is an informal working paper of the M.I.T. Laboratory for Computer Science, Computer Systems Research Division. It should not be reproduced without the author's permission and it should not be cited in other publications.

Research Summary for Workshop on Fundamental Issues in Distributed Computing

June 5, 1980

David D. Clark

I am interested in exploring a particular sort of distributed system, a system composed of a number of autonomous information processing machines dedicated to particular application or user groups. An obvious example of such a system is an interconnection of personal computers performing some task such as office automation. The concern of mine which I wish to explore in this note is the system requirements which are imposed on the individual machines of such a distributed confederacy if the overall goals of the system are to be met.

It is quite clear that the building of distributed applications is a difficult task, no matter the degree of language support. The problems of parallelism, and the desire to survive crashes of individual nodes during the course of a computation, make the construction of a robust and reliable application a substantially greater intellectual task than the completely serial realization on a single machine. I view this as a fundamental disadvantage of distributed systems, an approach which puts me in the opposite camp from many who feel that distributed systems are intrinsically better since the hardware is cheaper. Software costs will inevitably swamp hardware costs, for any legitimate and substantial application. Thus, in the construction of distributed systems we must look for some compensating advantage in the software area, to offset the increased difficulty of constructing applications.

A possible claim for distributed systems is that the operating system of the individual nodes will itself be much simpler than the large and complex operating systems which we build for today's centralized machines. It is often claimed that the operating system on the personal computer will be a very simple program, and I believe that this is the case. Thus, if we can truly recognize the goals of a simple operating system, this may make the software cost of a distributed system lower overall than its centralized equivalent. There is, however, a substantial pitfall in this argument. Many distributed systems, as they are proposed today, contain in addition to the distributed nodes a centralized machine with all of the power of today's systems. If that centralized machine must be there, then we are no better off than we were before we put the distributed nodes in, for we must now build the centralized operating system as well as the operating system on the distributed nodes, and the situation has surely gotten worse. (Let me assure the reader that I am not ignoring many of the fundamental advantages which have been proposed for distributed systems, including incremental growth, and robustness in the face of node failure. These are real and legitimate advantages which may suggest the use of a distributed system even if the initial cost of constructing the application is greater than it would have been in a centralized environment. However, goals such as these are not easy to realize, and to not come automatically just because a system is distributed. In fact, the achievement of these goals makes the distributed implementation even more complicated than it would have been otherwise. I believe it is an inescapable conclusion that the construction of distributed systems that capture the potential advantages of distribution is a very sophisticated task.)

My goal is thus to explore distributed systems which do not contain a centralized shared facility. The ideas which I would particularly like to avoid is a main frame which runs code belonging to several users or several different applications. If a machine is shared among several user or applications, then that machine must have the sophisticated protection architecture required to insure that mutually hostile programs do not interact in a bad way. If the centralized machine is being shared in real time then it must have sophisticated resource allocation strategies guaranteed to insure fairness between the conflicting needs of the various users. Thus, what I am attempting to eliminate from the distributed system I am proposing is a shared multi-application data management facility, since it is that sort of facility which it is extremely difficult to implement reliably on a single mainframe.

Before proceeding to explore the implications of building a distributed system without a centralized data management component, I would like to argue from another basis that this is a desirable goal, a basis not in cost but functionality. I believe that distributed data management systems are better than centralized because of the distributed nature of much of the information to be managed. Let me illustrate this point with a simple concrete example from the domain of office automation, a distributed application which tries to manage appointment calendars so that meetings involving several people can be scheduled.

If such an appointment system is to be genuinely useful, it must at least provide the flexibility and utility of the non-computerized method of making appointments. For example, it must recognize and honor the idiosyncracies which various people have about scheduling appointments. In my case, for example, only certain people can persuade me to schedule an appointment at 9 a.m. I prefer appointments to be clustered so that blocks of time are left free. Certain days of the week, and certain mornings and afternoons, may tentatively be reserved. Certain fixed appointments are known to be moveable if sufficiently stressed.

A distributed system with distributed information management could provide this degree of flexibility very easily. My appointment calendar would reside on my personal computer. I would be permitted to write the programs which manage my calendar, or at least to modify them so that they reflect my personal idiosyncracies. In order to make an appointment, a message would be send over the network to my personal computer.

Let us consider the perils of attempting to produce a version of this program in which the data management function is centralied. There are three possibilities, depending on how much code has been written in the centralized machine.

The first possibility is that the calendar application, as well as code reflecting my idiosyncracies, has been written to run in the centralized machine. Clearly, such a realization has the correct functionality, but at this point it is essentially a centralized application, with the distributed machines serving no function except data formatting and display management. Clearly, this is not really a distributed system. The structure does have the advantage that it is possible to make a successful appointment with me even if my personal computer is not available, which would have been a problem in the distributed scenario.

What happens if we eliminate the requirement that user code run on the centralized machine. It is conceivable that this might make the structure of this centralized computer somewhat simpler, which is the original goal I set out to solve. Now what control do I have over my appointments. I could check with the central computer periodically and cancel those appointments that didn't please me. This seems like a terrible idea since no one would ever be sure that an appointment was really committed. The centralized computer could query my machine on receipt of an appointment request, and if my machine is available it could comment on the suitability of the appointment. This structure will work effectively only if my machine is available most of the time, in which case we have lost one advantage of the centralized implementation. In my opinion, this is not a useful compromise.

A more interesting compromise would be if we could eliminate from the centralized machine the application dependent code as well as the user dependent code, so that what we are left with is nothing but a shared general purpose data management facility. How useful would this be as a component of our distributed calendar system?

I find a glaring defect in this scheme. If my calendar is represented in a central machine as a sequence of data items that can be queried and updated in an application-independent manner, that it is going to be very difficult to give someone the right to schedule an appointment, without also giving him the right to find out what appointments I have scheduled. That sort of constraint is generally available only by tailoring to the particular application. Is it inappropriate for people to examine my appointment calendar? In my opinion, very much so. It is possible to learn a great deal about someone's past and future by examining all the appointments which he has had over an extended period of time. For people in a position of power, even in a benign environment such as a university, this kind of analysis is probably inappropriate. I think that calendars must be protecting in a very particular way, so that one can determine whether a slot is available for an appointment, but one cannot find out what is in that slot if it is not free. In almost every data management system that I know, this kind of restriction can only be done as part of an application package.

To generalize very broadly from this very concrete example, much of the information in this world is private in its raw form. Only mediated access to that data is permitted. This is certainly true in the non-computer world. The distributed system captures this idea very nicely. My information is in my computer, and access to it is only by means of programs which I have written or provided. It is also possible to provide the same functionality in a sophisticated centralized machine which provides the ability to mediate access to databases through application packages. Intermediate positions, which tend to provide some sort of centralized facility, but offload application specificity to the distributed nodes, shoot for two targets and miss both. The correct functionality is achieved either by producing a system which is completely centralized, or completely distributed.

Having presented two arguments which suggest that sophisticated centralized data management facilities should be eliminated from a distributed system, let me consider for a moment the implication on the distributed systems of such an assumption. The most obvious conclusion has already been hinted at in the previous discussion. It must be a fundamental assumption of a distributed system that the personal computers are essentially available all the time. For various pragmatic reasons, this has not always been the case in experimental, distributed environments which have so far been assembled. What could prevent the personal computer from being available most of the time? First, the hardware could be unreliable. I refuse to accept this as a reasonable possibility. Second, the software could be unreliable. While the software is being developed, this is a very distinct possibility, and the problem of program development is one which must be separately dealt with, but operationally, I do not expect the personal computer to crash very often, since we are assuming that the operating system is supposed to be simple and thus presumably reliable. Third, the personal computer could be removed from the network in order to run stand alone applications. I will postulate that this will not occur for a large percentage of the time. Clearly, I am not assuming that the personal computer be unavailable all the time, but only with the same availability as one assumes of a secretary, perhaps 90 percent of the time. A final possibility, and one against which we must guard, is that some feature of the machine or its operating system makes the machine logically available for network services, even though it is physically connected to the network. If, for example, a pattern of programming a personal computer develops in which stand alone applications seize the entire machine while they run, then the running of these applications will preclude the possibility that a handler can run in the background in response to messages arriving over the network. Thus, the operating system which we design for the personal computer must have a rich enough processing environment that tasks can coexist in order to provide background jobs that respond to queries arriving over the network.

If we build a distributed system along these lines, what centralized functions might be appropriate? Is there any function for a shared data management facility? In fact, there is. Big disks are still much cheaper than small disks, and they have much higher performance. Thus, it is possible to imagine a shared disk for the storage of large quantities of information. However, this disk does not provide shared information; information is made available only through the personal computer. Thus, a data transaction now involves three machines: a personal computer issuing a query, a personal computer receiving the query, and a shared disk which has the information relevant to the query. In fact, information may flow directly between the disk and the originator of the query, if this optimization is compatible with the particular application. However, the basic model of the data storage facility is of a strictly partitioned collection of data stored for efficiency on one disk. This is a much easier system to build than one which provides shared access to data.

Another centralized facility which might be appropriate is a message forwarding machine. It is quite possible to believe that, even if machines are available most of the time, there will be occasional appointments which never get consumated unless there is an intermediary who can assist in the case where the two personal computers just never seem to be around at the same time. It would be nice if such a machine could be built in an application independent way, so that the function was simply the forwarding of a low level message traveling from one machine to another. However, the assumption of such a forwarding machine changes the nature of most computer protocols, since applications which previously expected either a confirmation or else a fairly rapid timeout must now be prepared for a message to be delivered hours rather than milliseconds later. This may be a severe distortion to the fabric of many applications. It is not clear whether such a message forwarding machine can be built in an application independent manner.

My basic conclusion is that the full benefit of distributed systems comes from making them truly distributed, rather than adjuncts to centralized facilities, and that the intermediate positions between a centralized machine and a fully decentralized machine have the costs rather than the benefits of both approaches. Thus, construction of decentralized systems as now being practiced by certain manufacturers, via a technique of gradually dispersing the function away from their large mainframe, is not a profitable and effective approach in the long run toward producing a system with interesting functionality. Rather, the decentralized system must be designed from the ground up, with every attempt to utilize fully the power of the decentralized machines to completely replace the complex functions previously performed by the centralized facility.