PROJECT MAC November 14, 1975

Computer Systems Research Division Request for Comments No. 95

# PROPOSED THESIS IN DISTRIBUTED SYSTEM

from Michael D. Schroeder

Attached is the M. S. Thesis proposal of Dan Milch, a VI-A student at IBM Research for whom I am the thesis advisor. It may be of interest to some because it addresses distributed systems, a potential future research topic for CSR.

---

Massachusetts Institute of Technology
Cambridge, Mass.

Proposal for thesis research in partial fulfillment
of the requirements for the combined degree of
Bachelor of Science / Master of Science
in Electrical Engineering and Computer Science.

Title: A Simulation of IBM's Advanced Administrative System in a Distributed Environment.

Submitted by: Daniel Milch
461 Palmer Ave.
Teaneck, N.J. 07666

Daniel Milch
(signature of author)

Date of Submission: October 22, 1975

Expected Date of Completion: December 19, 1975

Area where research is being conducted: IBM
T.J.Watson Research Center
Yorktown Heights, N.Y.

Brief Description of Problem: A model of a computer network which operates with a distributed database will be designed. The model will be based on IBM's Advanced Administrative System. A simulation of the network, using this model, will be performed using actual database reference streams obtained from the AAS support group in White Plains, N.Y. The aim here will be to explore some of the problems, advantages, and disadvantages of a distributed data network.

Supervision Agreement: The program outlined in this proposal is adequate for a Master's Thesis. The supplies and facilities required are available, and I am willing to supervise and evaluate the thesis report.

Michael D. Schroeder
(signature of thesis advisor)

MOTIVATING CONSIDERATIONS

The original motivation for the development of IBM's Advanced Administrative System (AAS) was to speed and make more accurate the task of computer system order processing. The development effort was begun in 1965 and an initial test system was operational in 1967. From the start, the intention was for the system to be used only within the company, and was never intended to be marketed as a "program product". As time passed, more functional capability was added to the system (this growth continues today) so that there are presently twenty different "applications", including order processing, that the system is capable of performing. Today, AAS reaches approximately 400 IBM regional and branch offices, located all across the country, through a network of interactive display terminals. The terminals are tied directly into four regionally distributed concentrators via 4.8 K-Baud lines. The concentrators, in turn, feed directly to two central sites over 50 K-Baud lines. These sites, one located in White Plains, N.Y. and the other in Bethesda, Md., jointly house the heart of the AAS system: the application and support software, and the database. The hardware at each site consists of the computers themselves (currently converting from 360/85's to 370/168's), various communication-controlling devices, and a

set of medium-to-low speed storage devices which hold much of the software and the database. No part of the database appears in duplicate; that is, data stored at one of the central sites is not stored at the other. The White Plains database consists of roughly 170 data files, and anywhere from one to four index files per data file, depending on the type of data file in question. This gives a total of approximately 460 files, or over two billion bytes. The database at Bethesda is slightly smaller in size. There appears to be no interaction between the two central sites in terms of exchange of data: files located at a given site remain at that site and can only be accessed by processes executing wholly at that site. Based on the preceding observation, this study shall only concern itself with that part of AAS which is housed at White Plains. This step is taken primarily as a means of scaling down the problem without the loss of important aspects, and will not be the only simplifying assumption made for this purpose. The size of the White Plains database coupled with its 3.5 million database access requests per day make that facility by itself a very large one to study. Adding the Bethesda site to the study would only complicate matters without adding any new insight.

Today, the concept of a distributed database and distributed systems in general is becoming more and more popular. As engineers strive to design computers which can execute

faster, they find that they are starting to run up against certain physical limitations which tend to thwart such efforts. One way of getting around these limitations is to design a number of fast machines for use in a multi-processor configuration. In this way, the execution rate for the entire system may approach a theoretical upper limit equal to the sum of the execution rates of the individual machines. In order to approach this new limit, however, the system design, both in terms of hardware and software, must minimize the overhead cost of machine-machine interaction.

In the case of AAS, one is tempted to consider the effects of distributing its large database among a number of geographically separated nodes, where each node is essentially a scaled-down version of one of the original central sites. If the only consideration here was increased execution time, geographical distribution would not seem advisable, since such an approach involves a certain overhead on top of that resulting from the transition to multi-processors. However, there are other considerations. To take a simple example, suppose that in a certain section of the country, AAS branch offices needed only to have access to twenty of the files in the database, and no other section of the country ever needed to have access to those files. Then, removing those files, along with a host computer, to that section of the country and allowing it to

run independently might seem like a good idea. Realistically, things are not that simple. In fact, a preliminary study indicates that there is some basis for assuming a degree of regionality of access. That is, some sections of some files in the database appear to be accessed primarily by one or another geographical region of the country. Other sections show no such regionality of access, and in fact are accessed frequently by all regions. Still other sections fall somewhere in between those two extremes. Totally independent operation of distributed processors is therefore ruled out. However, a network of processors which are semi-dependent, but which can still operate if one or two of their members becomes inoperative, appears somewhat attractive. The "payoff" for such a distributed system could come in a number of forms. One possibility, of course, is increased throughput, which was the original motivation. However, another could be a reduction in long-distance communication line usage and, therefore, cost. A third would seem to be that a distributed system is less likely to be totally disabled by a local disaster. That is, a nationwide system which depends totally on the operation of a single, central computing site could be totally incapacitated by a disruption affecting that site such as flood, power failure, earthquake, etc. The question then becomes: is there actually a net payoff in using a distributed system, or will such a system be slower, more costly, and less secure than its "centralized" counterpart.

PROPOSAL OUTLINE

The aim of this study is to try to answer the above question by developing a model of a distributed version of AAS, and then simulating this distributed version as it might run under an actual demand situation. Such a situation will be created by feeding the model data on some typical database access request sequences. The latter data will come from what is refered to as an AAS Trace Tape. In developing this distributed version of AAS, I intend to preserve in the model as many features of the centralized version as possible, adding only such functionality as is necessary to deal effecitvely and efficiently with the change from the centralized "orientation" to the distributed one. In other words, one of the guidelines in this effort will be to model as accurately as possible the various AAS operations implemented in the currently running system, and to model as realistically and consistently as possible new functions and operations which have no exact counterparts in the current system. In this way, comparisons between the operation of the real, centralized AAS system and the new, distributed version of that system will have a measure of validity.

The model will be implemented in the SIMSCRIPT II (SHARE version) simulation language. It will be run either on a 370/168 computer under VM/CMS, or on a 360/91 under OS. The language implementation on both machines is virtually

identical.

## SOME PERTINENT INFORMATION ON AAS OPERATION

AAS is logically divided into a front-end and a back-end system. The front-end, or message processor, deals directly with the user network, interpretting their requests and acting as an interface to the back-end system, known as the data management processor. Using display screen technology, the message processor transmits a screenfull of information to each user, the content of which is based on the history of that user's interaction up to the current point in time. The transmitted information ends in a request for the user to choose a reply from a displayed list of possible replies, and enter it appropriately. This reply, and the subsequent servicing involved, is refered to as an "action". The interpretation process performed by the front end translates each user reply (request) into a series of instructions in the AAS Macro Language. This language, which consists of roughly 140 different macro instructions along with a few standard assembly language instructions, is the one used in actually accessing, manipulating, and updating the AAS database. There is a subset of the AAS Macro Language which is concerned solely with the accessing and updating of the database (i.e. those instructions not concerned with manipulation, adjustment, or testing of data in core, or dealing with the flow of execution). This subset, hereafter refered to as the set of Data Management Requests (DMRs),

comprises just those instructions which are actually used by the message processor to request information (data) from DM. Every time one of the 30 (roughly) DMRs is received by DM, a record is made on the Trace Tape, which record includes a specification of the actual request involved, as well as various bits of information about the execution and completion of that request. It is with this level of execution that the model will concern itself.

DMRs are serviced by DM in a multi-processing mode, just as user requests to the message processor are handled in a multi-processing fashion. From the time a given DMR is received by DM until the time its execution is completed and the requested information sent back to the front end, that request is logically associated with one of a number of Data Management Monitors (DMMs, or simply "monitors"). The DMM, then, defines the process which is servicing the request. During the course of satisfying a request, the DMM is active while being serviced by the CPU or an I/O device, and dormant while it is enqueued for such service. A particular monitor is usually dedicated to a particular user for some portion of that user's action (defined earlier), and thus processes sequentially a portion of the DMRs involved in that action. Subsequent portions of that and other actions on behalf of that user may be assigned to the same or a different monitor, depending on the demand level.

FOCUS OF THE MODEL

Technically speaking, there are two major areas of interest which could be studied in terms of the transition from a centralized to a distributed system configuration. The first area deals with the effect of the transition on aspects and facilities of the currently availabel system, namely resource contention and allocation. How will a move to a distributed system affect the amount of time that DMRs spend waiting in queues? How will disk contention be affected? What will be the affect on buffer utilization: will a buffer containing a newly-read data block be able to satisfy more or less access requests before that buffer has to be reused? The second area of possible interest is the nature of the design and operation of the communication system and database shring protocols which must come into existence as a result of the transition to the distributed configuration. Will the database be partially duplicatd, or will the common data contained at any two nodes be nil? Can the sharing protocols be designed so that a minimum of time is spent in thrashing (a high level of communication back and forth between the various nodes)? It would be nice to consider both of these areas initially. However, due to time limitations, a more conservative approach will be taken: various simplifying assumptions will be made about answers to questions arising with regard to the former area, and the latter area will be explored in a bit more detail. The actual model will be designed to allow for expansion so

that, should the time be available after completion of the task just mentioned, the model can be modified to deal more carefully with aspects which may have not received sufficient attention.

## DEALING WITH CPU CONTENTION

Briefly, then, a first order approximation will be made for the amount of CPU time and I/O time that each DMR requires. At first, the probable approach will be to draw this approximation from a normal distribution whose mean value is determined in some reasonable way. Two methods come to mind for making such a determination: either by considering the individual I/O operations and the number of CPU-executable instructions involved in the execution of each DMR and obtaining time requirements from the device (I/O, CPU) specifications, or by deriving the lump figures from information contained on the Trace Tapes. The latter approach has the advantage that it would constitute a true measure of the actual system performance, which is the thing that is really of interest. There is a problem, however, in deriving even such a first-order approximation from the Tapes; that is, the time stamping for each record on the Tapes is done by refering to a standard machine clock which is incremented every hundredth of a second. Unfortunately (under these circumstances), actual CPU time requirements at the DMR level appear to be considerably less than the hundredth-of-a-second accuracy of the clock. Additionally,

there is a biasing inherent in the time stamping process; the nature of this biasing is not important, but the fact that it exists and cannot be accurately estimated means that the time fields are even less accurate as far as judging CPU usage is concerned. There may be a way of juggling figures around to get a reasonable time estimate; otherwise, the former technique will have to be used.

## "HOLDing" RECORDS FOR UPDATE

As mentioned earlier, database sharing techniques are, for the most part, unique to the distributed version of AAS, and will have to be carefully designed. However, there is one existing facility in the centralized AAS that is concerned with a lockout problem, and will be used in the distributed version. This facility is embodied in the "HOLD" option, which is available on all of the various read-type DMRs. A read with HOLD is used when the intention is to modify certain parts of the record and then write it back on the disk (or delete the record entirely). In fact, a record cannot be updated (or deleted) by a user unless that user was able to read the record initially using the HOLD option (one of the two types of deleting operations contains an automatic HOLD specification, so prior reading is not necessary). Only one user can HOLD a given record at any particular time, in order to prevent simultaneous updating. The fact that one user has a HOLD on a given record does not prevent another user from reading that record, but does

prevent other users from reading the record with a HOLD specification. Normally, a user can have a HOLD on only one record at a time. HOLD on a given record can be released by writing that record to disk, or by a few other operations.

"SN" AND "SS" BLOCKS

Up to this point, analysis has indicated that two different sharing protocols would be useful in a distributed AAS. Blocks containing records which are generally accessed by only one region will be statically resident only at the node which services that region. These will be refered to as Static Non-shared (SN) blocks and records. Of course, occasional access requests for records in such blocks originating from outside of the owning region would have to be dealt with. An outside request would cause a copy of the requested block to be transmitted to the requesting node. Prior to transmission, the block would be fetched in essentially the usual way by a "daemon" monitor specially reserved for such purposes. Depending on the traffic, more than one monitor might be required. Once located, the block could be written to a buffer which, while appearing to the monitor to be just like any other buffer, could actually be the input buffer to the transmission control unit. When the requesting node receives the transmission, the block could be transferred to a buffer which would remain in one of the buffer pools (or perhaps in a new buffer pool whose purpose would be solely to handle such cases) until such time as the

block is no longer needed. If the intent was to update a record within that block, the original request would have contained the HOLD option, and the servicing daemon would have set the appropriate flags. If and when the block is updated remotely, the actual disk update would have to take place at the owning node. In such case, changes are made remotely in the buffer, and the buffer contents are shipped back to the owning node.

Blocks containing records which are frequently accessed by more than one region must be statically resident at the appropriate nodes. This type will be refered to as statically shared (SS) blocks and records. If a given SS block contains no records that are ever updated by any of the owning nodes, no special treatment is required (the validity of such an assumption regarding any block in the AAS database is in doubt). However, if these records are updated, new procedures must be employed in order to insure that the database remains in a consistent state. Nodes owning SS blocks must at the very least be able to identify those blocks as such. Additionally, a node owning an SS block might want to know exactly which other nodes contain copies of that block. When one node wants to update a particular record within an SS block, a HOLD on that record must be broadcast to all the other nodes owning the block containing that record. In order to avoid deadlock problems, broadcasting a HOLD would actually involve

capturing one semaphore (as opposed to capturing many semaphores) located at a particular node having a copy of the block. Each SS block would have associated with it (at each node that has a copy) an ordered list of owners, where the first "operational" node on the list is the one that holds the semaphore. The purpose in having a list of possible owners rather than a single owner identifier is to avoid having one node depend unconditionally on any other specific node, and thus recreate the problems of a centralized implementation. With ordered owner lists, each node operates independently to the extent that it becomes the owner of the key semaphore when all other owining nodes ahead of it on the list are inoperative. Furthermore, this owner characteristic is implicitly and consistently transmitted across the network by virtue of the fact that when the first node on an owner list is deemed inoperative, the second node becomes the owner of the key semaphore, etc. (a "clean" solution would assume that an inoperative node would be operative enough to indicate its condition, and thus avoid having to make a timing-dependent decision as to how long to wait for a response before assuming the inoperative condition). When the semaphore is captured, the update can be performed. Once the update is completed, copies of the new record must be broadcast to all other nodes having copies of that record.

In the two preceding cases, an update at a node (or nodes)

which did not actually instigate that action was involved. The question of precisely how and when the physical update should occur is an important one, and one which has not been definitively answered as of yet. Currently, more information is needed on how updates are performed on the centralized AAS implementation. Indications are that changes to existing records and additions of new records are physically carried out immediately as part of the servicing of those requests; that is, when a DM monitor issues a change/add update request, it cannot service the next DMR until the update has been physically completed. Deletions, on the other hand, seem to be physically performed in a more leisurely manner; all that is immediately involved in servicing this type of request is the flagging of the appropriate index records pointing to the data record in question. The physical deletion actually occurs some time later, though once the index is flagged, the data record is no longer available. In a distributed environment, a large amount of SS update (immediate) interaction among nodes would probably tend to degrade overall system performance. The approach that may very well be taken here is the following: when a broadcasted update is received by a node, 1.) the index record(s) of the updated record are flagged, and 2.) the new copy of the record is held in a special "update pending" pool. The physical update would subsequently be performed when the system resources necessary to complete the update become available, or at

least are not under a heavily loaded condition. If the record which has an "update pending" flag set in its index is requested by a DM monitor, the record is fetched from the update pending pool and not from disk. If the record is locally requested using the HOLD option and then subsequently updated, the update occurs as usual, but, additionally, the flag is reset in the index and the updated record is purged from the update pending pool. In this way, the processing load at each node is more a consequence of the activities of the users in the region which that node serves, and less a consequence of activities and load conditions present at other nodes in the network.

DYNAMIC ALTERATION OF BLOCK STATUS

A third sharing protocol, which actually may be a modificaton of the two preceding protocols, would involve various blocks being "dynamically" resident at one or another of the nodes. Multiple copies of the block might exist at any given time, as with the SS blocks, or the block might exist without copies, as with the SN blocks. The block and its possible copies could be allowed to dynamically change residence, depending on time-wise shifting access patterns. For example, if a block is accessed frequently at one node during a given time period, and then is not accessed at all for some time, and then is requested frequently by a user at another node, it might be wise to change the residency of that block. The algorithm

that would make such a decision, in addition to refering to a time-of-day clock (available on most machines) would also have to know at what time and by whom (local or remote) the block was accessed. Having a block time-stamped each time it is accessed, and indicating whether the access was local or not, would be one approach. Another more costly approach would be to keep track of how many times the block is accessed locally as opposed to remotely. Based on this information, a decision might be made to 1.) change the status of the block from SN to SS (or from SS to SN) and/or 2.) change the residence "mode" from static to dynamic (or vice versa). The cost-effectiveness of the various approaches would have to be considered when deciding which one should be used in the model. Additonally, the assertion that access patterns do indeed shift with time has not been concretely established, although there is a general indication that this might well be the case. Finally, the time scale over which shifting access patterns are to be observed is important. Whether this scale should be on the order of days, hours, or minutes is something that has yet to be decided.

PROPER USE OF THE TRACE TAPES

The Trace Tapes will be used, among other things, to provide a realistic load for the model when a simulation is run. However, the exact order in which the DMRs are recorded on the Tapes is not at all the order in which they were

actually generated by the requesting users, and it is this latter sequence that is really of interest. Much of the ordering of the Tape records is in fact a consequence of the service times involved with those requests, as well as other conditions existing at the time the requests were processed. For example, consider the following two streams of requests generated by two hypothetical users (the requests and the sequences in which they are listed are actually valid and semantically correct, although their exact meaning need not be discussed here):

| USER A | USER B |
|---|---|
| : | : |
| GETSTR | VALID |
| NEXT | GET |
| NEXT | GETDUP |
| ENDSTR | GETDUP |
| REREAD | GETDUP (HOLD) |
| REREAD (HOLD) | ENDDUP |
| PUT | DELETE |
| : | : |

These two streams, of course, would be merged on the Tape. However, the way in which they are merged depends very much on the characteristics of the system at the time they are received and processed; i.e., what blocks currently are resident in the buffer pools, what the CPU contention is like, what the I/O contention is like, etc. There is no reason to assume that these characteristics will be duplicated in the distributed system; in fact, any such assumption tends to defeat the whole pupose of the modelling and simulation. It is precisely the individual user request streams, therefore, and not the merged request stream that appears on the Trace Tapes, that are needed as simulation

input.

Unfortunately, in order to satisfy such a requirement to the letter, all the Trace records used in the simulation would have to be scanned, sorted, and stored before the simulation was begun. Practically speaking, this is not possible. There are ceratin assumptions that can be made, however, which can simplify matters significantly, and at the same time result in only a small deviation from the original requirement. What will be involved is a look-ahead process which scans a finite number of records ahead on the Tape while the simulation is in progress. The aim is, at any given point in the simulation, to always have the next DMR issued by each active user "in hand" so that when the request currently being processed for that user completes, the next one can be supplied immediately. The question is, how far ahead does one look before despairing of finding that next request, and thus assuming that the user has finished and logged off, has gone out for some coffee, or has just stopped to collect his thoughts. The answer is as follows: according to figures supplied by AAS support personnel, 95% of the inputs to the system (actions) are serviced in five seconds or less. Since a number of DMRs are generated for each action, it can be assumed that an even higher percentage of DM requests are serviced within the five second period (or, similarly, that 95% of all DMRs are serviced within a shorter amount of time). Information

on the Trace Tapes indicates, for each DMP, when the request was received and how long it took to service it. Based on this information, one can decide that after looking ahead on the tape for a certain amount of recorded time (and not finding the next request for a supposedly "active" user), no further look-ahead is required in order to provide the model with an accurate picture of all the immediately pending requests from each user active at that time.

These, then, are what I see as some of the motivations for and goals of the modelling / simulation effort. Undoubtedly, specific requirements may change, old ideas may be discarded, and new ideas may be adopted. However, I feel that this proposal has presented a fairly accurate picture of the direction in which these efforts are currently heading.

REFERENCES

Books and Articles

Kiviat, P., R. Villanueva, and H. M. Markowitz, The SIMSCRIPT II Programming Language. Englewood Cliffs, N.J., Prentice-Hall (1968).

Mihram, G., Simulation: Statistical Foundations and Methodology. N.Y., Academic (1972).

Reitman, J., Computer Simulation Applications: Discrete-Event Simulation for Synthesis and Analysis of Complex Systems. N.Y., Wiley-Interscience (1971).

Wimbrow, J. H., "A Large-Scale Interactive Administrative System", IBM System Journal, Vol. 10, No. 4. 260-282 (1971).

People

Conversations and private communications with the following individuals have been of help:

C.R.Attanasio, M.Auslander, G.Bucci, H.Markowitz, P.Markstein of IBM (T.J.Watson Research, Yorktown Heights, N.Y.).

C.Ellis of the University of Colorado.

J.Saltzer, M.Schroeder of MIT.

D.Cretara, S.Kruse, R.Langham, J.Madziola, R.Nardi of IBM (DPD Headquarters, White Plains, N.Y.).

K.Zabriskie of IBM (DPD, Bethesda Md.).