M.I.T. Laboratory for Computer Science
Computer Systems Research Division

December 19, 1977
Request for Comments No. 152

File Allocation in a Distributed System

by Allen W. Luniewski

As distributed systems come into being, so will distributed information repositories. Such repositories will consist of multiple computers, perhaps every computer in the distributed system, geographically distributed, communicating through a multiplexed communication network and serving their customers through the same network. This paper addresses the issue of allocating physical copies of the information within the repository to the various computers (nodes) of the repository. A model is developed that describes the allocation problem. Some problems with using this model to solve the allocation problem are pointed out and requirements for a useful allocation algorithm presented.

The repository consists of information collected into objects called files. A file is a collection of information that the user of the repository has declared to be related. The repository assumes nothing else about the information in files. Examples of files are traditional files, Multics segments and ORSLA areas[Bishop]. The purpose of the information repository

is to provide both query and update access to the information within it.[1]

1. Why Not Complete Distribution

To be successful the repository must provide fast, reliable access to its data for its customers. Effectively, this means that the data should be close, in terms of delay through the network, to the requestor of the data. This would seem to indicate that a copy of a file should be placed at every node of the repository that uses it - complete distribution. Although a simple allocation policy, this solution is not necessarily optimal or even feasible. There are at least two objections to complete distribution - one economic and one performance related.

Keeping a copy of each file at every node that uses it requires enormous amounts of storage at each node - enough to hold a copy of all of the information in the repository that the node uses. Such a policy might be feasible for a repository consisting of only a few nodes or a repository with a limited amount of information, however for larger numbers of nodes, or repositories with more information, this is not economically feasible (memory is cheap but not free). Thus, from an economic point of view, the number of copies of files should be minimized.

---

[1] The term "database system" is not used here since the repository attaches no intrinsic meaning to the information content of what it stores. An analogy to the repository is the file system in more traditional, centralized computer systems (thus the term "file").

The performance question concerns updates to the information in the repository. Complete distribution leads to minimal access time (since each node has a local copy to reference), but update times may become unacceptable. Since the repository is responsible for keeping all copies of a file consistent,[1] an update incurs all of the problems associated with multiple copy file systems (see for instance [Thomas]). All solutions to the multi-copy problem incur some overhead, related to the number of copies, thus degrading the responsiveness of the repository to updates. Thus update performance considerations lead to the minimization of the number of copies while access time constraints lead to maximizing the number of copies.

Considering all three of these factors the conclusion is that allocating a copy of every file at every node, complete distribution, is not the most desirable policy,[2] and, rather, what is wanted is an allocation with a minimal number of copies that provides adequate response to accesses.

2. A Mathematical Formulation

In [Chu] Wesley Chu presented a model of file allocation in a distributed computer system. His model attempts to minimize storage and transmission costs subject to delay and storage constraints. This section presents a model, inspired by Chu's work and similar to Casey's [Casey], that attempts to

---

[1] The possibility of having files that are not kept up to date but, rather, are only periodically updated is not considered here. Such actions are envisioned as happening at a higher level in the system where the semantic content of files can be of help in doing this.

[2] This is not to say that for certain files such an allocation policy might not be most desirable.

minimize expected delay subject to the storage capacity constraint of nodes and constraints on maximum acceptable delay. This model differs from Chu's in that delay is the factor of concern and not money and differs from Casey's in that all files rather than just one are examined.

The cost (objective) function will be taken to be the system wide expected delay in accessing or updating a file. The cost function, which is to be minimized, is given by:

$$C = \sum_{i,j} (u_{ij} * \min_{x_{kj}=1} (a_{ijk}(X) * x_{kj}) + \min_{x_{kj}=1} (c_{ij}(X) * v_{ij})), \text{ i a node, j a file.}$$

where u_{ij} is the usage rate of the j'th file by the i'th node, $a_{ijk}(X)$ is the expected delay in accessing file j, which is at node k, by node i given the allocation $\{X\}$,[1] x_{ij} is 1 if file j is allocated at node i and 0 otherwise, v_{ij} is the update rate of the j'th file by the i'th node and $c_{ij}(X)$ is the cost, in delay, of node i updating the j'th file given the file allocation $\{X\}$. The goal is to find an allocation $\{X\}$ to minimize C given the other parameters, subject to given constraints (below).

The parameters a_{ijk} and c_{ij} reflect delays due to the queueing of requests at nodes in the network and due to transmission delays within the network (e.g. delays at packet switches in a packet switched net or delays due to collisions in an ethernet). In addition, $c_{ij}(X)$ reflects the cost to perform an update given that there are N_j copies of the j'th file (e.g. $N_j = \sum_i x_{ij}$). These two parameters are hard to evaluate, or express simply[2] (and, in fact,

---

[1] $\{X\}$ = the set of all x_{ij}.

[2] A model that more closely approximates the real system would treat them as

they represent the complexity of real systems). In his paper Chu analyzed the term $a_{ijk}(X)*x_{kj}$ under the assumption, among others related to queuing delays, that there is only one copy of the file in the network and found an expression for it in terms of sums of products of the x_{ij}. A similar expression should be derivable for $c_{ij}(X)*v_{ij}$.

There are three constraints on choosing the x_{ij}. First, there must be at least one copy of every file:

$$\sum_i x_{ij} \geq 1 \qquad \text{for all } j.$$

Second, the storage capacity of each node is bounded by b_i bits:

$$\sum_j x_{ij}*L_j \leq b_i \qquad \text{for all } i.$$

where L_j is the length, in bits, of the j'th file. Finally the delays, for accesses and updates, that each node experiences should be bounded in a node dependent fashion (by A_{ij} and U_{ij} respectively):

$$\max_k a_{ijk}(X)*x_{kj} \leq A_{ij} \qquad \text{for all } i, j.$$

$$c_{ij}(X)*v_{ij} \leq U_{ij} \qquad \text{for all } i, j.$$

The first constraint reflects access time constraints and the second reflects update time constraints. These four constraints restrict the ways in which {X} can be chosen while trying to minimize C.

Modifications to this model are possible. For instance a constraint bounding the number of copies allowed for a file may be useful when it is known in advance that multiple copy updates are extremely expensive.[1] A more

---

time varying random variables. Such a model would be a stochastic integer programming problem.

[1] This information is implicit in the magnitude of the c_{ij}'s, however specifically including these constraints serves to reduce, perhaps

realistic model than the one presented here can be obtained by using a probabilistic programming model. In one such case, the access time and update time bound constraints would not be regarded as absolute bounds, but, rather, would be constrained to hold with given probabilities (the idea being that processes should usually experience low delay but an occasional long delay is acceptable for most users). One could also envision constraining files to always (never) lie on certain nodes (such constraints might arise as a result of security requirements on the files). Another possibility is to ignore storage constraints of nodes. In this case the problem becomes easier since it is possible to allocate the copies of one file without worrying about the allocation of other files - the linear program becomes uncoupled with respect to the files.[1] Many such modifications are possible; this model has been chosen as it reflects the essence of the allocation problem.

3. Problems with a Mathematical Approach

There are four problems that make the use of the model in the previous section to solve the file allocation problem impractical. This section will point out these problems. Solutions are not presented as the solutions represent research problems of varying difficulty.

---

dramatically, the collection of {X} that must be examined. (In the jargon of the trade, this constraint serves as a cutting-plane).

[1] This is true only if one assumes that the delays to access files located at a particular node are independent of the overall allocation of files. In effect this assumes that queuing delays in the network are independent of the allocation of files, at least to a first approximation.

The model presented in the previous section is an integer 0-1 programming problem; that is, the unknowns, x_{ij}, are constrained to take on only the values 0 or 1. In [Eswaran] it is shown that the linear 0-1 integer programming problem is NP-complete in the size of {X}. The model presented in the last section is at least this hard since it is non-linear (i.e. the cost function and/or the constraints are not linear in the x_{ij}). A consequence of being NP-complete is that no algorithm, except for enumeration of all possible solutions, is known for solving the model for the x_{ij}. As the model contains a very large number of variables (= #_of_nodes*#_of_files), solving it for the optimal solution is impractical in all but trivial cases.[1]

Even if one assumes that solving the model is practical in a computational sense, one is still faced with the distributed, on-going nature of the computation. It is an on-going computation since files are constantly being created and destroyed. In general this means that the optimal allocation of files changes every time that a file is created (or destroyed) and, as a result, the "correct" location of all files must constantly be reevaluated and files constantly moved around in order to maintain optimum response. As this calculation is probably difficult, and the movement of files potentially expensive, it is impractical to always have the optimal allocation. It is a distributed computation since various parts of the repository know only some of the model's parameters. For instance, a node probably only knows its own access/update rates and not those of other nodes. In addition the allocation decisions of one node (e.g. picking some of the

[1] An interesting possibility is that the subclass of 0-1 problems described by this model is not NP-complete and that, in fact, good algorithms for solving them may exist.

x_{ij}) effect the "optimal" decision of other nodes. For instance node A may decide to allocate file 1 at node B and this may prevent node C from allocating file 2 at node B due to storage limitations at B. Thus in order to perform the calculation, either (1) all of the parameters must be collected on one node in order to allow that node to perform the calculation or, (2) a method of computing the solution must be devised that allows each node to work on the part of the problem that it knows about or, (3) an algorithm that allows collaboration among the nodes must be devised. The first possibility introduces centralized control, the second may be hard to find due to the intertwined nature of the computation (e.g. inter-file and inter-node dependency, principally through the $a_{ijk}(X)$ and $c_{ij}(X)$) while the third may be difficult to find since it must produce an optimal solution even though nodes involved in the computation may fail.

The last two questions concern the parameters of the model. The preceding discussion has assumed that all of the parameters of the model are known. However, in general, they are not known - only estimates of them can be known. Thus error is introduced in the estimation of {X} that may result in not only a sub-optimal allocation but also a very poor one! Unfortunately the situation is even worse than this - not only are the parameters unknown but they are also time varying and it is the time varying nature of the parameters that is important. For instance as $\{u_{ij}\}$ and $\{v_{ij}\}$ change they reflect the time varying demand for files. If the parameters were static it would be easy to monitor file usage for a few days, accepting the potentially large overhead to do so accurately, and then place files in their correct place in the repository. In practice, however, the pattern of requests for a

file ($\{u_{ij}\}$ and $\{v_{ij}\}$) varies over time. There are short term variations due to high demand from a user and more long term variations due to changes in the overall patterns of access. Ideally, both types of variation should be handled by the allocation process.

4. The Desired Algorithm

The issues raised in the last section lead to one common conclusion - the explicit solution of the integer programming model is not practical. Instead what is wanted is an algorithm that does "well" in some (as of now undefined) sense. This algorithm should be distributable - individual nodes must be able to decide on the "proper" location of files in a manner that is as independent as possible of other nodes. It must provide a low overhead solution in terms of storage, delay and CPU time (thus recording the entire access history of a file is not a reasonable algorithm even though such an algorithm should permit very good allocation decisions to be made). Lastly the algorithm must respond to the time varying nature of accesses. Response to long term variations is essential, but response to short term variations should probably be considered a secondary goal since it may be hard to provide and higher level encachement of files may alleviate the problem.

In addition a decision algorithm is needed to decide whether or not to use the new allocation calculated by the allocation algorithm. This decision algorithm must weigh the cost, under some model, of moving files to arrive at the better allocation verses the potential savings of the new allocation. This decision is hard since there is a risk involved - the data that caused

the new predicted optimal allocation may have been of a transient nature and did not reflect future demand where as the current allocation does so.

5. Conclusion

This paper has discussed the problem of allocating files in a distributed information repository. Such a repository should provide fast, reliable service to its users. A 0-1 integer programming model for allocating files within the repository was presented. It has been argued that explicit solution of the model should not be undertaken. Rather, the model should be used as a guide in evaluating and discovering algorithms to do the allocation. Finally, the requirement for a distributable, low-overhead algorithm for file allocation that results in adequate response to user requests was presented. The search for such an algorithm should prove to be a useful and interesting research problem.

BIBLIOGRAPHY

[Bishop] Bishop, Peter B., "Computer Systems with a Very large Address Space and Garbage Collection", M.I.T. Laboratory for Computer Science Technical Report 178, May 1977.

[Casey] Casey, R.G., "Allocation of Copies of a File in an Information Network", Proceedings AFIPS 1972 SJCC, Vol. 40.

[Chu] Chu, Wesley W., "Optimal File Allocation in a Multicomputer Information Center", IFIPS Conference Proceedings, 1968.

[Eswaran] Eswaran, Kapali P., "Placement of Records in a File and File Allocation in a Computer Network", IFIPS Conference Proceedings, 1974.

[Saltzer] Saltzer, J.H., "Comments on Murphy's Left Handed Least Unlikely Last Optimal File Allocation Algorithm with Examples", Datamatics, 13(1 April 1977), pp.7-11.

[Thomas] Thomas, R.H., "A Solution to the Update Problem for Multiple Copy Data Bases Which Use Distributed Control", BBN Report #3340, July, 1976.