M.I.T. Laboratory for Computer Science

Computer Systems Research Division

March 22, 1978

Request for Comments No. 162

Some Results on File Allocation in a Local Network

By Allen W. Luniewski

This paper considers the allocation of files in a distributed computer system operating in a local network environment so as to improve the response of the system to query and update requests on those files. Although the allocation of files in the most general case is hard [4], the allocation of files in a local network, under suitable models of updates, can be analyzed. This paper presents results for two ways of performing updates that show that under these update strategies an allocation policy exists that is simple, low in overhead, optimal and has no undesirable centralized control of allocation.

1. Single File Allocation

The model of file allocation presented in [4] consisted of a cost model based upon the delay a requestor sees before his request to access (or update) a file is completed. A set of constraints that reflected the physical limits (e.g. storage capacity) of the system and the desired limits on response by the system to requests was developed. These constraints cause the linear programming problem corresponding to the file allocation problem to be "coupled"; that is, the allocation of one file will effect the correct allocation of other files, potentially all of them. This occurs since placing one file at a node may make it physically impossible, due to storage limitations, to place another file at that same node. Such placement may be required to meet the limitations imposed upon the access times that some nodes must have to some files. Also placing a file at some node in the network may effect the access times seen by other nodes to other files due to the queueing delays within the communication network that are induced by traffic to the first file. If we ignore these constraints then it is possible to consider the allocation of one file independently of the allocation of other files. This is the

approach taken in this paper since it causes the file allocation problem to be greatly simplified.[1]

When the file allocation problem is uncoupled by ignoring storage constraints, the linear programming problem that remains contains few constraints. The remaining constraints are:

1. At least one copy of the file must exist.
2. The access (update) times that some (perhaps all) nodes achieve must be bounded.
3. Some nodes must always (never) have a copy.

In the remainder of this paper the second two constraints will not be explicitly considered in the analysis however the first will be considered. This approach will permit analysis of the allocation problem in a straight forward manner. A later section section of this paper will consider the last two constraints.

2. The Local Network Assumption

One of the characteristics of a general communication network is that messages between nodes in the network experience differing delays due to different routing decisions and due to varying queueing delays in delivering the messages. This means that the distance between nodes (in terms of message delay) is potentially different for all pairs of nodes and, moreover, the delay between two nodes may vary over time. Incorporating this variability into a model leads to a model of the file allocation problem that is difficult to analyze. Thus this paper will make an assumption in order to make this variability disappear.

This paper will model the distributed system as a local network and make "the local network" assumption: the delay between nodes in the distributed system is a fixed constant independent of the identity of the communicating nodes and any routing decisions made for the communication. On the surface this assumption may seem unreasonable, but an examination of the technologies used to construct geographically localized networks indicates otherwise. In both the ring net [3] and the

---

1. This approach has also been taken in [2] by Casey.

ethernet [5] the transmission delay between nodes is almost independent of the identity of the communicating nodes since, due to the nearness of the nodes on the communication medium, the transmission delay is very small and almost negligible. Rather, the delay to requests is due almost entirely to the time required to process the request.

The local network model also assumes that every node can respond to requests in approximately the same time each time so requested. If this were not the case variations in message delay, as perceived by the sender before a response is received, would be noticed. Basically, queueing delays in processing messages are ignored in this formulation. This assumption is somewhat stronger than the one in the previous paragraph and may, in fact, not be justified. It is made, however, since it results in a simplified analysis and will hopefully lead to some insight into the problem of file allocation.

The result of these two assumptions is that the delay that a requestor experiences before he receives a response to his request is a constant no matter which other node the request is directed to. The time to reference a local copy can, however, be different and, for the purposes of this paper, will be considered to be zero.

In the file allocation problem the inaccuracies inherent in the local network assumption are probably not overly important. As argued in [4] the optimal allocation of files is a very hard problem and what a practical system should do is to approximate the optimal allocation. The solution that results from the local network assumption will, hopefully, be a close approximation to the optimal allocation although only simulation and experience with a real system will indicate whether or not this is so.

3. The Basic Model

The basic model to be considered in the remainder of this paper consists of an expression that reflects the system wide expected delay in accessing a specific file. Thus this kind of model has to be constructed for every file in the system.

The basic cost model is the following:

$$C(X) = \sum_i \lambda_i \min_{x_k=1} a_{ik}(X) + \mu_i \min_{x_k=1} u_{ik}(X), \quad i \text{ a node.}$$

where $C(X)$ = Cost for the file in question when using the allocation X.

λ_i = The rate at which the i^{th} node accesses the file in question.

μ_i = The rate at which the i^{th} node updates the file in question.

$a_{ik}(X)$ = Cost for the i^{th} node to reference a copy of the file on node k.

$u_{ik}(X)$ = Cost for the i^{th} node to update the file by sending the update request to node k.

X = The allocation of files = $\{x_i | x_i=1$ if the file is on node i and 0 otherwise$\}$.

The local network assumption for query accesses to the file can be expressed as:

$$a_{ik}(X) = \begin{cases} 0 \text{ if } i=k \text{ (i.e. a local copy is referenced)} \\ A \text{ if } i \neq k \text{ (i.e. a non-local copy is referenced)} \end{cases}$$

where A is a fixed, network dependent, constant. This reflects the fact that all non-local references cost the same and that local references are for practical purposes free.

In order to simplify the following discussion we will define N to be the total number of copies of the file kept by the system (i.e. $N = \sum_i x_i$).

The one aspect of this model left unspecified up to this point is the cost of updates. Determining this cost depends upon a particular model of updates. In the remainder of this paper two models of updates will be considered. The first will allow updates to be sent to any copy of the file and the second will only allow updates to be sent to a master copy of the file.

4. Unsynchronized Update Model

Perhaps the simplest way to perform updates in a distributed environment is to allow updates to be sent to any copy of the file.[1] This copy then sends the update to all other copies of the file. After all copies[2] have acknowledged the update, the copy that originally received the update will

---

1. In general the update is actually sent to the node in the distributed system containing the copy, but for this discussion it is easier to abbreviate this by referring to requests as being sent to the copy itself.
2. For this discussion the possibility of copies of the file being inaccessible is not considered. Thus this model addresses the normal case in which all copies of the file are accessible.

acknowledge completion of the update to the originator of the update. It is only at this point that the originator of the update considers it complete. Note that this is a very simple update protocol since there is no attempt to maintain consistency between the various copies of the file. This is not proposed as a reasonable model of updates, rather it is used since it lends itself to analysis and the techniques used will be very similar to those used in the more realistic update model presented later. It should be noted, however, that in SDD-1 [6] updates are performed in a manner that is similar to this model although the details of that procedure are too complicated to include in the simple model presented in this paper.

In this model of updates the update cost, $u_{ik}(X)$, can be represented as:

$$u_{ik}(X) = \min_{x_k=1} a_{ik}(X) + u_1 + (N-1)u_0$$

where u_0 = the cost to send the update to one node containing a copy.

u_1 = the overhead cost for updating the first copy and broadcasting the update to other copies.

The first term reflects the cost to reference the initial copy of the file. The second term reflects the overhead that the first copy experiences in coordinating the update. It may include the cost to set up a process to do the update, the cost of doing access checking and the cost to prepare an update message for sending to the other copies of the file. The third term reflects the per-copy cost that the first node experiences in coordinating the update. It may include the cost to send the update to each copy and it may include the cost to receive the acknowledgement for the update from each copy. In either case, u_0 represents the incremental cost, to a node i, of adding a new copy of the file and is constant for all nodes.

Using this model of updates the optimal strategy for allocating files can be derived. The complete cost function is given by:

$$C(X) = \sum_i \lambda_i \min_{x_k=1} a_{ik}(X) + \mu_i\left(\min_{x_k=1} a_{ik}(X) + (N-1)u_0 + u_1\right)$$

Now let $\Delta_i = \min_{x_k=1} a_{ik}(X) = \begin{cases} 0, & x_i=1 \text{ (node i can reference a local copy)} \\ A, & x_i=0 \text{ (a non-local copy is referenced)} \end{cases}$

$$\therefore C(X) = \sum_i \lambda_i\Delta_i + \mu_i\Delta_i + \mu_i(N-1)u_0 + \mu_i u_1$$

$$= \sum_i \Delta_i(\lambda_i+\mu_i) + U_T(u_1+(N-1)u_0)$$

where $U_T = \sum_i \mu_i$ = the total update rate of the file in question throughout the system.

Now consider giving a copy of the file to a node I that did not previously have a copy. The change in the cost function C(X) is then given by ΔC:[1]

$$\Delta C = C(N+1 \text{ copies}, x_I=1) - C(N \text{ copies}, x_I=0)$$

$$= \{\sum_{i\neq I}(\lambda_i+\mu_i)\Delta_i + (\lambda_I+\mu_I)\Delta_I + U_T(u_1+Nu_0)\}$$
$$-\{\sum_{i\neq I}(\lambda_i+\mu_i)\Delta_i + (\lambda_I+\mu_I)\Delta_I + U_T(u_1+(N-1)u_0)\}$$
$$= (\lambda_I+\mu_I)*(0-A) + u_0U_T$$

The first terms canceled since Δ_i is the same with and without $x_I=1$, the second term produces the first product since Δ_I is 0 when node I has a copy and A when it does not. Thus giving node I a copy of the file improves system performance if and only if $\Delta C \leq 0$. The case of $\Delta C=0$ is included since the cost of the allocation under this cost model is not increased by giving node I a copy, but it does increase the reliability of the system due to the presence of another copy. Thus we add node I if and only if $-A(\lambda_I+\mu_I) + u_0U_T \leq 0$, or if $\lambda_I+\mu_I \geq u_0U_T/A$. Conversely, a node I that has a copy should be deleted from an N+1 node allocation if $\lambda_i+\mu_i < u_0U_T/A$.

This derivation showed how to allocate the second and subsequent copies of the file. It is not valid for the one copy case since it took the form "What is the incremental cost reduction for adding another file". Fortunately the one copy case is simple to analyze:

$$C(1 \text{ copy}, x_I=1) = \sum_i(\lambda_i+\mu_i)\Delta_i + U_T(u_1+(N-1)u_0)$$
$$= \sum_{i\neq I}(\lambda_i+\mu_i)A + U_Tu_1 + (\lambda_I+\mu_I)\Delta_I$$

Thus making node I the one to have the single copy is optimal when $\lambda_I+\mu_I$ is maximal over all nodes since the term $(\lambda_I+\mu_I)\Delta_I$ becomes zero in that case and the term U_Tu_1 has the same value no matter which node is chosen (e.g. a result of the local network assumption). This condition is a relatively easy one to test since all accesses are now done to <u>one</u> copy of the file where appropriate measurements may be taken.

---

1. For the rest of this paper the notation C(text) will be used for the cost function where "text" describes the allocation used to compute the cost.

Putting the results of the previous paragraphs together leads to a distributed file allocation procedure. Once the parameters of the model are known, each node can decide independently whether or not to have a copy of the file. If node i finds that $\lambda_i+\mu_i \geq u_0 U_T/A$ then it should have a copy, otherwise it should not. If no nodes satisfy this condition then only one copy is needed and the node with the largest value of $\lambda_i+\mu_i$ gets the copy.

It is now necessary to show that this distributed allocation algorithm is optimal; that is given the constants $\{\lambda_i\}$, $\{\mu_i\}$, A, u_0 and u_1 the allocation chosen by this algorithm, after <u>all</u> nodes have made their decision, has minimum cost over all possible allocations.

The following lemma will be needed for the proof. Consider the following allocation procedure, called the greedy procedure: Given that the allocation will contain N copies of the file, choose as the N nodes to have a copy those nodes with the largest $(\lambda_i+\mu_i)$.

Lemma: The allocation achieved by the greedy allocation procedure has minimum cost over all possible N node allocations. Conversely, any minimum cost N node allocation can be realized by the greedy allocation procedure.

Proof: Let S1 be the set of nodes chosen by the greedy allocation procedure. Let S2 be any other N node allocation. Now calculate C(S1) and C(S2):

$$C(S1) = \sum_{i \notin S1 \cup S2} (\lambda_i A + \mu_i(A+(N-1)u_0 + u_1))$$
$$+ \sum_{i \in S1-S2} (\lambda_i * 0 + \mu_i(0 + (N-1)u_0 + u_1))$$
$$+ \sum_{i \in S1 \cap S2} (\lambda_i * 0 + \mu_i(0 + (N-1)u_0 + u_1))$$
$$+ \sum_{i \in S2-S1} (\lambda_i A + \mu_i(A + (N-1)u_0 + u_1))$$

$$C(S2) = \sum_{i \notin S1 \cup S2} (\lambda_i A + \mu_i(A + (N-1)u_0 + u_1))$$
$$+ \sum_{i \in S1-S2} (\lambda_i A + \mu_i(A + (N-1)u_0 + u_1))$$
$$+ \sum_{i \in S1 \cap S2} (\lambda_i * 0 + \mu_i(0 + (N-1)u_0 + u_1))$$
$$+ \sum_{i \in S2-S1} (\lambda_i * 0 + \mu_i(0 + (N-1)u_0 + u_1))$$

$$\therefore C(S2)-C(S1) = \sum_{i \in S1-S2} (\lambda_i+\mu_i)A - \sum_{i \in S2-S1} (\lambda_i+\mu_i)A \geq 0$$

The first term is larger since: 1. There are as many elements in S1-S2 as in S2-S1,[1] and 2. As a result of the manner in which the elements in S1 were chosen, each term in the first summation is at least as large as each term in the second summation. Thus $C(S2)-C(S1) \geq 0$ and the allocation that is achieved by the greedy allocation procedure is optimal over all N node allocations; that is, there does not exist a better N node allocation.

To show the converse, let S2 be a minimum cost N node allocation and let S1 be the allocation chosen by the greedy allocation procedure. Now compute C(S2)-C(S1) as above. In order for S2 to be optimal, C(S2)-C(S1) must be ≤ 0. There are two ways in which this can happen. First, $S1-S2=S2-S1=\varnothing$ in which case S1=S2 and the converse is trivially true. Second the sets S1-S2 and S2-S1 may be non-empty (although they still must have the same number of elements). In this case remember that the set of $\lambda_i+\mu_i$ in S1 are maximal over all nodes. Thus $C(S2)-C(S1) \geq 0$ as in the previous part of this proof. Therefore, in order for S2 to be optimal, C(S2)-C(S1) must be identically zero. These two facts in turn imply that each term in the summation over S1-S2 has a term of equal value in the summation over S2-S1 (i.e. the set of values $\lambda_i+\mu_i$ in S1-S2 is the same as in S2-S1). Thus S2 is an allocation that is also achievable by the greedy algorithm.[2] Thus the converse of the lemma is proven. ∎

Now consider the distributed allocation procedure presented earlier in this section. After <u>all</u> nodes have decided whether or not to have a copy of the file, those nodes that have chosen to have a copy will all have the property that $\lambda_i+\mu_i \geq u_0 U_T/A$. Moreover all nodes not chosen have the property that $\lambda_i+\mu_i < u_0 U_T/A$. Therefore the distributed allocation procedure is an optimal N node allocation since it produces an allocation that is the same as that produced by the greedy allocation procedure (where N is the number of nodes chosen by the distributed allocation procedure).

---

1. This is because S1 and S2 have the same number of elements, N.
2. The fact that $S2 \neq S1$ simply means that there is an arbitrary choice to be made in choosing the allocation and that that choice does not effect the cost of the allocation. This can occur if many nodes have the same value of $\lambda_i+\mu_i$.

In order to complete the proof of optimality of the distributed allocation procedure it is necessary to show that it chooses N optimally. Assume that the distributed allocation procedure chooses N nodes. First we show that this N node allocation is better than any allocation of less than N nodes: Any allocation of less than N nodes must have left out nodes which satisfy $\lambda_l+\mu_l \geq u_0 U_T/A$ since there are exactly N such nodes. But from the way in which the distributed allocation procedure was derived adding any nodes so left out must reduce the system cost, thus the optimal allocation contains at least N nodes. Second consider an allocation of greater than N nodes. From the lemma, any such allocation must have the N nodes of the N node allocation as a component. Thus we may consider getting to this larger allocation by adding nodes to the N node allocation achieved by the distributed allocation procedure. However from the derivation of the distributed allocation procedure it is known that adding any more nodes increases the system cost. Thus no allocation of greater than N nodes can have lower cost than the allocation achieved by the distributed allocation procedure. Thus the distributed allocation procedure chooses optimally. Therefore the allocation achieved by the distributed allocation procedure has minimum cost over all possible allocations and optimality of the allocation has been shown.

5. The Master-Slave Model of Updates

Another model of updates is the broadcast model proposed in [1]. Under this model one copy of the file is designated as the master copy and the rest as slave copies. All updates to the file are sent to the master copy. It then broadcasts the update to the other copies. Query requests are sent to any available copy of the file. In this section this model of updates is analyzed and some criteria for the allocation of files within a local network derived.

Let node M be the node that has the master copy of the file. Then the cost for node i to perform an update is just the cost to send the update to node M plus the cost for the master node to broadcast the update to all the slave copies.[1] Thus we have that:

$$u_{ik}(X) = a_{iM}(X) + (N-1)u_0 + u_1$$

1. Again, for the purposes of this discussion the possibility of unavailable copies and their effect on the cost of queries/updates is ignored.

The constants u_0 and u_1 have the same meaning as in the last section although, of course, their actual values may be different. Also note that u_0 may be 0 if the master copy does not require the requestor of the update to wait for completion of the update on all copies.

Thus the cost function C is given by:

$$C(X) = \sum_i \lambda_i \min_{x_k=1} a_{ik}(X) + \mu_i(a_{iM}(X)+(N-1)u_0+u_1)$$
$$= \sum_i (\lambda_i\Delta_i+\mu_i a_{iM}(X)) + ((N-1)u_0+u_1)U_T$$

A node can be in one of three states at any given time. It can have no copy of the file, it can have a slave copy of the file or it can have the master copy of the file. The remainder of this section will consider three transitions: from no copy to a slave copy, from a slave copy to the master copy and from no copy to the master copy.

First consider giving node I a slave copy of the file when it had no copy previously. Proceeding as in the previous section we calculate ΔC as:

ΔC = C(N+1 copies, x_I=1 as slave copy) - C(N copies, x_I=0)

$$= \{\sum_{i\neq I} (\lambda_i\Delta_i+\mu_i a_{iM}(X)) + \lambda_I\Delta_I + \mu_I a_{IM}(X) + U_T(Nu_0+u_1)\}$$
$$-\{\sum_{i\neq I} (\lambda_i\Delta_i+\mu_i a_{iM}(X)) + \lambda_I\Delta_I + \mu_I a_{IM}(X) + U_T((N-1)u_0+u_1)\}$$
$$= \lambda_I(0-A) + \mu_I(A-A) + U_T u_0$$
$$= U_T u_0 - A\lambda_I$$

Giving node I a slave copy of the file improves the system cost if and only if $\Delta C \leq 0$. Thus node I should have a slave copy of the file if $U_T u_0 - A\lambda_I \leq 0$ or if $\lambda_I \geq U_T u_0/A$. This criterion, as was the one in the last section, is good for distributed control of this part of the allocation decision.

Now consider the case of giving node I the master copy when it currently has none. The old master copy will become a slave copy (the possibility of taking away the old master's copy is ignored for the moment). Proceeding as before we calculate ΔC:

ΔC = C(N+1 copies, x_I=1 as the master node, node M as a slave copy)

-C(N copies, x_I=0, M the master copy)

$$= \{\sum_{i\neq I,M} \lambda_i\Delta_i + \lambda_I\Delta_I + \lambda_I\Delta M + \sum_i a_{iI}(X)\mu_i + U_T(Nu_0+u_1)\}$$
$$-\{\sum_{i\neq I,M} \lambda_i\Delta_i + \lambda_I\Delta_I + \lambda_I\Delta M + \sum_i a_{iM}(X)\mu_i + U_T((N-1)u_0+u_1)\}$$

$$= \lambda_l(0-A) + \lambda_M(0-0) + u_0U_T + \sum_i(a_{il}(X)-a_{iM}(X)))\mu_i$$

Now $a_{il}(X)-a_{iM}(X) = \begin{cases} 0 \text{ if } i\neq l \text{ and } i\neq M \\ -A \text{ if } i=l \\ A \text{ if } i=M \end{cases}$

$\therefore \Delta C = -A\lambda_l + u_0U_T - \mu_lA + \mu_MA$

Thus giving node I the master copy reduces the system cost if and only if $\Delta C \leq 0$. In other words if $-A\lambda_l+u_0U_T-\lambda_lA+\mu_MA\leq 0$. Or if $\lambda_l+\mu_l\geq\mu_M+U_Tu_0/A$. This condition is again a simple condition for deciding when to give a node the master copy. Unfortunately it does not have the distributed control property that the previous results had since knowledge of the master copy's update rate is needed[1] at each node. The next paragraph of this paper will show that this is not really a problem as the allocation problem can be broken into two parts; the first the decision as to whether or not to give a node a copy and the second to decide which of the nodes having a copy should be the master copy; the first can be made in a distributed manner while the second can be made by the master copy.

Finally consider changing the status of node I from that of a slave copy to that of master copy and at the same time make the old master copy a slave copy. Once again ΔC is calculated:

$\therefore \Delta C$ = C(N, node I has a slave copy, node M has the master copy)

- C(N copies, I has the master copy, M has a slave copy)

$$= \{\sum_i\lambda_i\Delta_i + \sum_i\mu_ia_{il}(X) + ((N-1)u_0+u_1)U_T\}$$
$$-\{\sum_i\lambda_i\Delta_i + \sum_i\mu_ia_{iM}(X) + ((N-1)u_0+u_1)U_T\}$$
$$= \sum_i\mu_i(a_{il}(X)-a_{iM}(X))$$

But $a_{il}(X)-a_{iM}(X) = \begin{cases} 0 \text{ if } i\neq l,\ i\neq M \\ -A \text{ if } i=l \\ A \text{ if } i=M \end{cases}$

$\therefore \Delta C = -\mu_lA + \mu_MA$

Making node I's copy the master copy and the current master copy a slave copy reduces the overall system cost if and only if $\Delta C<0$. Or if $-\mu_lA + \mu_MA<0$, or if $\mu_l>\mu_M$. (The case $\Delta C=0$ is ignored since there is no reliability issue here).

---

1. A strategy whereby the master node periodically broadcasts its update rate to other nodes in order to allow them to say "Now I should be the master copy" is, of course, possible. However the next few paragraphs will show that such a strategy is not needed.

The one copy case remains to be investigated. As in the last section it is easily analyzed, noting that the one copy case is also the master copy:

$$C(\bar{x}_l=1) = \sum_{i\neq l}(\lambda_i\Delta_i+\mu_i a_{iM}(X)) + ((N-1)u_0+u_1)U_T + (\lambda_l\Delta_l+\mu_l a_{lM}(X))$$
$$= \sum_{i\neq l}(\lambda_i+\mu_i)A + (\lambda_l+\mu_l)\Delta_l + u_1U_T$$

Thus, by the same reasoning as in the previous section, we make node l the one copy (and hence also the master copy) if it has the maximum $\lambda_l+\mu_l$ over all nodes.

The optimality of this allocation follows from the local network assumption in a manner similar to that given in the previous section. For that reason it is not repeated here and, rather, is left as an exercise for the reader.

Now the complete allocation strategy for the local network case under the master-slave model of updates can be described. A node should have a copy of the file if it updates the file frequently enough; e.g. when $\lambda_l \geq u_0U_T/A$. The master copy is chosen from those nodes that have a copy of the file by making it the node that updates the file at the greatest rate. Both criteria have a nice intuitive feeling to them which is reassuring in validating the model. Although this strategy employs centralized control to determine the master copy, this decision is always easily made by the current master copy since, by definition, it handles <u>all</u> update traffic.

6. The Constraints

At the beginning of this paper three constraints were presented that a valid solution to the allocation problem must satisfy. The first constraint has already been dealt with - there must always be at least one copy every file.

A second constraint is that some nodes must experience bounded access times. Under the local network assumption there are only two possible access times, 0 and A. If the bound is less than A for some node then that node must have a copy of the file. If the bound is greater than or equal to A then no special action is needed since <u>any</u> allocation will meet that bound.

Another constraint is that the time to perform an update must be bounded. From the cost of updates this immediately implies an upper bound on the number of copies of the file that can be in the system since update cost is a linear function of the number of copies. The effect of this on allocating copies is that if too many nodes are eligible to receive copies then only the most eligible (those that produce the most reduction in system wide cost) should be allowed to receive copies. Unfortunately this means that centralized control is needed to make this decision.

The last constraint is that some nodes must always (never) have a copy. If some node must never have a copy this presents no problem - it simply never considers making a copy locally. If some node must always have a copy this also creates no problem since it is given a copy and the remaining nodes can still make their decision in the manner outlined earlier in this paper. This comes about since the decision algorithm has been derived in the form "Given N copies, when are N+1 copies better?" and this derivation still is valid.

7. Conclusions

This paper has considered the issue of file allocation in a local network environment. Under two models of updates - unsynchronized updates and the master-slave model - some simple criteria for deciding whether or not a node should have a copy of the file have been derived. These criteria all have the very desirable property of not requiring centralized control.

The local network assumption is a very strong one. In a local network this assumption is probably very close to exact. In more general networks the assumption begins to break down. The hope, to be verified by further research, is that the allocation in general networks resulting from this assumption is not far from optimal.

This paper is not the end-all of the local network file allocation problem either. The issues of transactions and inter-file consistency have been ignored. Since transactions access sequences of files, can this information be used to produce a good allocation? How do methods to maintain mutual consistency effect the cost of queries/updates? Must these considerations inevitably lead to the coupling of one files allocation to another?

Acknowledgment

Special thanks to Warren Montgomery for suggesting the local network assumption at a time when the file allocation problem seemed insurmountable.

Bibliography

[1] Alsberg, P.A., et. al., "Multi-Copy Resiliency Techniques", University of Illinois, CAC Document #202, May, 1976.

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

[3] Farber, D.J., Larson, K., "The Structure of a Distributed Computer System - Communications", Proceedings of the Symposium on Computer-Communications Networks and Teletraffic, Microwave Research Institute of Polytechnic Institute of Brooklyn, 1972.

[4] Luniewski, Allen W., "File Allocation in a Distributed System", MIT Laboratory for Computer Science, Computer Systems Research Division RFC 152, December 19, 1977.

[5] Metcalfe, R.M., et al., "Ethernet: Distributed Packet Switching for Local Computer Networks," CACM19, No. 7, pp. 395-404, July, 1976.

[6] Rothnie, J.B., Bernstein, P.A., Goodman, N., and Papadimitriou, C.A., "The Redundant Update Methodology of SDD-1: A System for Distributed Databases," Computer Corporation of America Technical Report, February, 1977.