1985

Performance Analysis of Network File Systems

Walter F. Tichy

Zuwang Ruan

Report Number:

85-511

Performance Analysis of Network File Systems

Walter F. Tichy
Zuwang Ruan

Tilde Report CSD-TR-511
April 3, 1985
Revised April 17, 1985

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

ABSTRACT

In a local area network, the efficiency of the network file system is a key performance issue. This paper compares several different classes of network file systems with respect to number of file transfers and auxiliary messages processed by each node.

The file system classes compared are: (a) *remote file access*, (b) *polling* (each node may create a read-only replicate of a file, but must poll the primary copy to determine whether it is up-to-date), (c) *broadcast staling* (each update broadcasts a message to mark replicates as stale), (d) *multicast staling* (each update sends an outdating message to the sites with replicates).

The results show that, under realistic assumptions, both polling and staling require 30-50% fewer file transfers than remote file access, while simultaneously increasing reliability because of replication. However, polling and staling generate auxiliary messages. Broadcast staling performs poorly in large nets because every node must process an auxiliary message for every update in the net. Polling generates numerous useless messages if the read ratio is high. Multicast staling performs best. It generates one half to one fifth of the auxiliary messages produced by polling. Furthermore, the load on each workstation is small and independent of the network size.

This work was supported in part by grants from the National Science Foundation (MCS-8219178), SUN Microsystems Incorporated, and Digital Equipment Corporation.

Performance Analysis of Network File Systems

Walter F. Tichy

Zuwang Ruan

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

1. INTRODUCTION

As the technology for local area networks matures, we are witnessing a paradigm shift in our view of computing resources. Before the advent of high-speed, low-cost networks, communication among machines was slow and expensive. Consequently, each computer was an isolated, autonomous unit, engendering the *centralized view* of computing. With recent advances in network technology, we have grown accustomed to working with networks of dozens of autonomous computers, linked with high-speed communication facilities. In such networks, it is easy to exchange mail and other data, but each user still runs his programs and stores his data primarily on a single, fixed node in the network. We call this extension of the centralized view the *distributed view*. However, this view is only a temporary stage in the current paradigm shift. The drawbacks of the distributed view will lead to the notion of a *computing engine*. A computing engine is a virtual computing environment that provides a wide range of services and hides the topology of the underlying network. The user of a computing engine deals only with the logical services; the location of objects, the identities of the processors, and the required communication channels are of no interest. The computing engine view is a prerequisite if we wish to succeed in (1) building location independent programs that execute on any node in the net and can be transported to other environments, (2) sharing objects by their logical names instead of by their incidental locations, and (3) expanding our local networks without massive reprogramming.

A network file system is an important component of a computing engine. It provides a long-term store for objects and makes them accessible from all machines in the engine. Efficiency of the file system is crucial, because it determines whether the illusion of a computing engine is credible. If access to remote files is inefficient, or if adding new nodes to the engine degrades file system performance, users will revert to the distributed view with all its shortcomings.

This paper presents and compares several different classes of network file systems. Each class is analyzed in two different scenarios: In the symmetric scenario, all network nodes have identical secondary storage capabilities, and the file system data is evenly distributed over all nodes. In the asymmetric case, the engine contains one or more special nodes that act as shared file servers, whereas the remaining nodes have little or no storage capacity. The performance measure chosen is not the load on the network per se, but rather the workload that each individual node must handle in order to support a network file system.

The next section outlines the network file system classes. Section 3 presents the performance analysis of the file systems. Section 4 strengthens the results by developing a more general model that allows the analysis of file reference locality.

2. NETWORK FILE SYSTEM CLASSES

We distinguish the following five general classes of network file systems: remote file access, polling, broadcast staling, multicast staling, and weighted voting.

2.1. Remote File Access

With remote file access, any file is accessible from any node in the computing engine, no matter where in the engine the file is stored. Remote file access must provide *access transparency* and should provide *location transparency*. Access transparency is achieved if system calls operate on any file in the network, regardless of the file's location. Typical implementations use remote procedure calls to reach non-local files.

File location transparency means that it is impossible for the user to derive the location of a file from its name. File location transparency is required for writing truly portable programs. It also allows the operating system to pick the location of a file, to replicate files, and to migrate files in order to improve efficiency.

A fairly large number of remote access file systems have been implemented. Example systems with access transparency, but no location transparency are COCANET [1], UNIX United [2], and IBIS [3]. The following remote access systems provide varying degrees of location transparency: LOCUS [4], the Unix Network File System [5], and the Apollo File System [6]. LOCUS also provides some file replication, but no migration.

An important aspect of a network file system is its resiliency with respect to node failures. In a file system with remote access and no replication, all files on a failed node become inaccessible. In addition, files on nodes that have not failed may also become inaccessible, if parts of the directory paths leading to them are stored on failed nodes. Only replication can guard against inaccessibility.

2.2. Polling

In a polling system, each file consists of a *primary copy* at some site, and zero or more read-only, *secondary copies* at other sites. All write access is to the primary copy. A read operation at a site other than the primary site first attempts to read the local secondary copy. If none is present, the operation creates the copy with a file transfer. If it is present, the read operation polls the copy's primary site and compares time stamps, in order to determine whether the copy is up to date. If not, a file transfer is started. An example system incorporating this approach is the ITC Distributed File System [7].

Polling has two major advantages over remote access file systems. First, polling improves read-availability, because replicated files (and directories) remain available even if their primary copies are on a failed node. Second, performance improves provided the frequency of reads is higher than that of writes. The reason is that read operations at a given node transfer each remote file once; all but the first read operation will be local until the next update. By contrast, remote file access transfers essentially the entire file on every operation.

2.3. Broadcast Staling

Broadcast staling is the mirror image of polling. It uses primary/secondary copies like polling, except that the responsibility of outdating secondary copies rests with the write operation. Each write operation (which must update the primary copy) broadcasts a message that marks corresponding secondary copies as stale. Each read operation uses the local secondary copy if it is not stale, or performs a file transfer otherwise.

To prevent useless broadcast messages, the primary copy is in one of two states: U or M. State U means that the file is a unique copy; there are no replicates anywhere. State M means that there may be replicates in the network. If a primary copy in state M is updated, a staling message is broadcast and the file reverts to state U. Write operations in state U do not broadcast a staling message. A primary copy's state changes from U to M when the first replicate is created. A secondary copy is either in state F (fresh) or S (stale). The complete state transition diagram for both primary and secondary copies is given in Figure 1.

While staling has the same properties with regard to read-availability as polling, it requires a special protocol to restart a failed node. A failed node misses all staling messages until it is restarted. Thus, a conservative approach would be to mark all secondary copies at that node as stale during startup. However, depending on the duration of the outage, many of those files may actually still be fresh. They can be salvaged if the restarted node enters a polling mode until all its secondary copies have been verified as being either fresh or stale. In this manner, start-up time is minimized.

2.4. Multicast Staling

Multicast staling is identical to broadcast staling, except that each primary copy maintains a list of sites which have a replicates. If this list is empty when a write occurs, no message is sent (similar to state U above). If the list is non-empty, a write operation sends a staling message to all sites on the list, and then erases the list. This arrangement significantly reduces the number of auxiliary messages compared to broadcast staling, especially if the network is large.

The multicast staling protocol is self-adapting in the following sense. If a file is written, only the sites that are interested in it will be notified. If a file is written very rarely, it will eventually be replicated everywhere and allow for efficient reads. Few staling messages will be sent; in particular, the staling messages should be less frequent than polling messages. If a file is written frequently (for instance, temporary files are written about as often as they are read), replicates will rarely develop and no staling messages will be necessary. A network file system with these properties is being developed at Purdue [3,8].

If a node fails and is restarted, it enters a polling mode for secondary copies, just as in broadcast staling. Polling mode terminates once all secondary copies have been touched. Since the lists of sites which have replicates may be corrupted, the node also enters broadcast mode for primary copies. Broadcast mode reverts to multicast mode once all primary copies have been touched.

2.5. Weighted Voting

In weighted voting [9], each write operation writes W copies, and each read operation inspects R copies. W and R are chosen such that $W+R > N$, where N is the total number of copies of each file in the network. In this fashion, the read operation is guaranteed to find the latest version of a file. If the read and write operations have a sufficiently large overlap, node failures can be tolerated. A degenerate case of weighted voting is full replication: Write operations duplicate their data on all nodes. Then read operations need to access only the local copy.

Weighted voting, and full replication in particular, are highly reliable, but also extremely expensive. Full replication, for instance, has the effect of burdening every node with all updates in the entire network. Weighted voting may be acceptable for a small number of nodes, but its cost becomes impractical as the number of nodes increases. We shall therefore exclude it form further consideration.

3. MODELLING AND ANALYSIS

For modelling file access costs, we consider only the number of file transfers and the number of polling and staling messages being processed by each site. Thus, we ignore the cost of local file operations. Our goal is to measure the overhead placed on each processor for supporting a network file system*.

File transfers generate a significant load for each node involved in the transfer. By contrast, polling and staling messages are lightweight and require only the checking of time stamps or the change of state attributes, respectively. We therefore regard the number of file transfers as the major criterion, and the auxiliary messages as the minor criterion. Note also that a staling message is one–way, whereas a polling message requires a reply. We ignore the cost of the reply, because the boolean return value can easily be piggybacked onto the acknowledgement required in a reliable system.

We assume furthermore that every read or write operation accesses an entire file. Thus, a remote access has the same cost as a file transfer. This assumption is reasonable, since sequential file I/O predominates, at least for UNIX-like operating systems [10], and sequential I/O is a strong indication that the entire file is being

* From our measures, it is possible to derive the load on the interconnection network, but since the local area network is not normally the bottleneck, we leave this computation as an exercise to the reader.

processed. However, it should be noted that the notion of a file in our model is an abstraction, and our results apply to individual blocks or records of files as well.

The analysis proceeds in two steps. First, we compute the cost of accessing a single file, and then use the results to compute the cost for the entire file system. In the second step, we consider the following two cases. The symmetric scenario is a net in which all nodes have identical storage capabilities, and the file system is evenly distributed over all nodes. The asymmetric scenario is a net in which a shared file server stores all files, whereas the other nodes have little or no storage capacity.

3.1. Access Cost for a Single File

The access sequence to the file is characterized by two independent random variables *OP* and *SITE*. The value of *OP* is either *read* or *write*, specifying the operation. If n is the number of sites in the network, *SITE* takes on a value in $\{1, 2, \cdots n\}$, indicating the requesting site. We assume the following probabilities:

$P\{\,OP = read\,\} = r,\ \ P\{\,OP = write\,\} = w,\quad r + w = 1$, and

$P\{\,SITE = i\,\} = \lambda_i,\ \ 1 \le i \le n,\quad \sum_{i=1}^{n} \lambda_i = 1.$

Among the n sites, site p is distinguished. In case of the remote access scheme we assume the file resides at site p; for the other protocols, the primary copy is assumed to be located there.

3.1.1. File Transfers

In a remote file access system, all operations requested from sites other than p require file transfers, because of our assumption about full-file access. Therefore, the probability of transferring a file for reading or writing is

$$t_{ra} = 1 - \lambda_p .$$

In the polling and staling schemes, a secondary copy at some site i $(i \neq p)$ alternates between two states: it is fresh (F), i.e., identical to the primary copy, or not fresh (S). If it is not fresh, it is either stale or not existent. Polling and both variants of staling cause the same number of file transfers, because in all three cases, a secondary copy is created or refreshed during a read operation, and the copy becomes stale during a write operation. The only difference is that in the staling schemes, it is possible to determine locally whether a file is stale, whereas the polling protocol interrogates the primary copy.

The above assumptions about *OP* and *SITE* imply that the state of the copy at site i is a Markov chain with the following transition matrix (see also Fig. 1):

$$\left[\begin{array}{cc} r & w \\ r\lambda_i & 1 - r\lambda_i \end{array}\right]$$

The probabilities of a secondary copy at site i being fresh ($P_{F\,i}$) or not ($P_{S\,i}$) are thus as follows:

$$P_{F\,i} = \frac{r\lambda_i}{w + r\lambda_i}, \quad P_{S\,i} = \frac{w}{w + r\lambda_i}.$$

Since a read operation from a site with a stale copy results in reading the primary copy and refreshing the local one, it requires a file transfer. In addition, all write operations requested from sites other than p need remote accesses. The probability of transferring a file (for reading or writing) in the polling and staling schemes is thus:

$$t_{ps} = \sum_{i \neq p} r\lambda_i \frac{w}{w + r\lambda_i} + w\,(1 - \lambda_p).$$

3.1.2. Auxiliary Messages

Remote file access does not require any auxiliary messages for maintaining a network file system.

In the polling scheme, all read operations from sites other than p need to poll the primary copy. The expected number of polling messages received at site p for a primary is therefore

$$a_{po} = r\,(1 - \lambda_p).$$

In the broadcast staling scheme, the primary copy sends a staling message to all nodes when its state changes from multiple (M) to unique (U). Similar to the above analysis for the secondary copy, the state of the primary copy is a Markov chain with the following transition matrix (see also Fig. 1):

$$\left[\begin{array}{cc} r & w \\ r(1 - \lambda_p) & w + r\lambda_p \end{array}\right]$$

Hence the probabilities of the primary copy at site p being in state M or U are

$$P_M = \frac{r(1 - \lambda_p)}{1 - r\lambda_p}, \quad P_U = \frac{w}{1 - r\lambda_p}.$$

The primary copy generates a staling message whenever it is in state M and there is a write operation. If we assume that the network provides a broadcasting facility, the staling message is sent to all $n-1$ other sites. Consequently, the expected number of staling messages received per file operation anywhere in the net is

$$a_{bst} = w\ (n-1)\frac{r(1-\lambda_p)}{1-r\lambda_p}.$$

In the multicast staling scheme, messages are sent only to those nodes that have a secondary copy. Since a site receives a staling message for a particular file if and only if there is a fresh secondary copy in it and a write operation occurs, the expected number of staling messages received per file operation anywhere in the net is

$$a_{mst} = w \sum_{i \neq p} \frac{r\lambda_i}{w + r\lambda_i}.$$

3.1.3. Number of Replicates

The number of replicates created per file on nodes other than p in the polling and staling schemes is

$$r_{ps} = \sum_{i \neq p} \frac{r\lambda_i}{w + r\lambda_i}.$$

3.2. Performance of the Network File System

Based on the analysis in the preceding section, we can now determine the performance of the entire network file system. Let the number of files be q. We assume that each file f_j $(1 \le j \le q)$ satisfies the assumptions we made in the last section. The access sequence to f_j is parameterized by $r^{(j)}$, $w^{(j)}$ and $\lambda_i^{(j)}$, $i = 1, ..., n$. To simplify our discussion further, we assume the requests for all files have the same rate and the same distribution. That is, for all $1 \le j \le q$ we have

$r^{(j)} = r,\ \ w^{(j)} = w,$

$\lambda_i^{(j)} = \lambda_i,\ 1 \le i \le n.$

We study two different scenarios of network file systems. In the symmetric scenario, all sites have identical storage capabilities, and the files (or the primary copies) are evenly distributed across all sites. In the asymmetric scenario, one distinguished site, the file server, issues no requests but provides a file store shared by all other sites.

3.2.1. The Symmetric Scenario

We assume that for each file the request rate from its storage site (for remote access) or primary site (for the other three cases) is λ_p, and all other sites have the same request rate, namely, $\frac{1-\lambda_p}{n-1}$.

Let μ be the total arrival rate of file requests from a site, which is assumed identical for all sites. Recall that the results derived in the previous section reflect the cost (the number of file transfers and the number of auxiliary messages to be processed by each site) caused by one request. We can easily compute the total file transfer rate in the entire network, and then divide by n to get the file transfer rate T of each site. Recall that polling and staling cause the same number of transfers.

$$\boxed{T_{ra} = \frac{n \; \mu \; t_{ra}}{n} = \mu \, (1-\lambda_p)}$$

$$\boxed{T_{ps} = \frac{n \; \mu \; t_{ps}}{n} = \mu \, (1-\lambda_p) \left[1 - \frac{r^2 \, (1-\lambda_p)}{w \, (n-1) + r \, (1-\lambda_p)} \right]}$$

The rate A of receiving auxiliary messages (polling, broadcast staling or multicast staling) at a node is:

$$\boxed{A_{po} = \frac{n \; \mu \; a_{po}}{n} = \mu \; r \; (1-\lambda_p)}$$

$$\boxed{A_{bst} = \frac{n \; \mu \; a_{bst}}{n} = \mu \; r \; w \; \frac{(n-1)\,(1-\lambda_p)}{(1-r\,\lambda_p)}}$$

$$\boxed{A_{mst} = \frac{n \; \mu \; a_{mst}}{n} = \mu \; r \; w \; \frac{(n-1)\,(1-\lambda_p)}{w\,(n-1) + r\,(1-\lambda_p)}}$$

We assume that for each broadcast and multicast, a node sends out exactly one message. We also assume a broadcast scheme with efficient acknowledgement. The network would easily clog if each broadcast would result in n replies. In reliable networks, a node does not need to acknowledge until it has received n messages from the broadcasting node. This arrangement results in one acknowledgement per broadcast on the average. A similar consideration applies to multicast.

Figure 2 shows the rates of file transfers and auxiliary messages to be handled by each node as the size of the network increases. The parameters were chosen as follows: $\lambda_p = 1/n$ (hence $\lambda_i = 1/n$ for all $1 \le i \le n$), i.e., all sites have identical arrival rates for each file. The read ratio was set to $r = 0.8$ and $w = 0.2$, according to the file access rates given in reference [10]. The solid curves stand for the file transfer rates at each site, while the dashed curves represent the arrival rate of polling or staling messages. Even under the assumption that there is no locality of reference, polling and staling require about 30% fewer file transfers. At the same time, read availability increases, because replicates are created. The number of replicates created under the present assumptions approaches $r/w = 4$ as n increases.

Note that the number of file transfers rises gradually in all cases and is limited by $\mu(1-\lambda_p)$. The number of broadcast staling messages grows linearly with the number of nodes, making this method impractical in large, symmetric networks. Clearly, multicast staling is the method of choice, since it produces the least amount of auxiliary messages.

3.2.2. The Asymmetric Scenario

For the remote access file system, we assume that all files are in site 1, the file server. For the polling and staling schemes, the primary copies of all files are in the file server, and some of them are replicated to other sites on demand. There are no file accesses from the file server, while all workstations generate the same arrival rate to each file. Thus, $p = 1$, $\lambda_p = 0$, $\lambda_i = 1/(n-1)$, $i = 2, \ldots, n$.

In this scenario, the cost to the file server and that to a workstation are different. Let μ be the total arrival rate from a workstation. Clearly, the total arrival rate for the computing engine is $\mu \cdot (n-1)$. The file transfers per time unit for the file server are:

$$\boxed{T_{ra,1} = \mu\,(n-1)\,t_{ra} = \mu\,(n-1);}$$

$$\boxed{T_{ps,1} = \mu\,(n-1)\,t_{po} = \mu\,(n-1)\left[1 - \frac{r^2}{w(n-1)+r}\right]}$$

The file transfer rates for the workstations are the above rates divided by $(n-1)$.

The polling scheme adds more load to the file server. The arrival rate of polling messages at the file server is:

$$\boxed{A_{po,1} = \mu\ (n-1)\ a_{po} = \mu\ r\ (n-1)}$$

The generation rate of polling messages at the workstations is the above rate divided by $(n-1)$.

In broadcast staling, the arrival rate of auxiliary messages at a workstation equals the generation rate of staling messages at the file server (provided each broadcast sends out a single message.) These rates are as follows:

$$\boxed{A_{bst,i} = \frac{1}{n-1}\ \mu\ (n-1)\ a_{bst} = \mu\ r\ w\ (n-1),\ 1 \le i \le n.}$$

In multicast staling, the generation rate of multicast messages at the file server equals the one for broadcast staling. The reason is that in both cases, a staling message must be sent if there is a write operation to a file which has replicates (state M or non-empty list of sites, resp.).

$$\boxed{A_{mst,1} = A_{bst,1} = \mu\ r\ w\ (n-1)}$$

In contrast, the arrival rate of multicast messages at a workstation is:

$$\boxed{A_{mst,i} = \frac{1}{n-1}\ \mu\ (n-1)a_{mst} = \mu\ r\ w\ \frac{(n-1)}{w(n-1)+r},\ 2 \le i \le n}$$

Figure 3 shows the rates of file transfers and auxiliary messages to be handled by the file server as the size of the network increases. The parameters are $r = 0.8$, $w = 0.2$. Clearly, the number of file transfers and auxiliary messages to be handled by the file server increases linearly with the size of the network. Thus, the file server will be the bottleneck in any asymmetric network sooner or later. Compared to remote access, both polling and staling reduce the number of file transfers by almost 30%. In general, the reduction depends on the read ratio r, and is, of course, greatest if r is close to 1. Staling and polling may therefore allow the network to expand farther than with remote access, before the file server becomes a serious bottleneck. At the same time, reliability increases, as the number of replicates created approaches $r/w = 4$, as in the symmetric case.

Unfortunately, polling generates a large number of auxiliary messages, and that number increases rather than decreases as r approaches unity. Thus, the polling messages might easily overwhelm the gain in reduced file transfers, such that polling might be slower than remote access in a large net. In contrast, the number of multicast staling messages decreases as r approaches unity. Furthermore, the number of staling messages generated by the files server is significantly smaller than the number of polling messages received at the file server. More precisely, $A_{po,1} / A_{mst,1} = 1 / w$. For the parameters in Fig. 3, this means that the file server must handle five (!) times as many polling messages as multicast messages.

Finally, the bottom line in Figure 3 shows the number of multicast staling messages received by each workstation. The number of these messages is bound by μr, a small constant independent of the size of the network. This means that the overhead on a workstation does not increase as the network expands. (The same is true for polling, but not for broadcast staling.)

4. LOCALITY OF REFERENCE

This section studies the influence of file access locality on the performance of network file systems. File access locality means that if a file is accessed by a site, this site has higher probability of issuing the next request for the file than any other site.

It is obvious that locality does not affect the performance of the remote file access scheme, since files are not cached in the requesting sites at all. However, locality influences the performance of the polling and staling schemes. This section generalizes the model of the previous section and strengthens the results.

4.1. Access Cost for a Single File

Consider a particular file in a computing engine with n sites $\{1, 2, ..., n\}$. Assume that the file's primary copy is at site p. The access sequence to the file is characterized by two discrete-time random processes OP_t and $SITE_t$. For example, the t-th request for this file is a read operation from site i if $OP_t = read$ and $SITE_t = i$. We assume that $SITE_t$ is a homogeneous Markov chain with transition matrix

$$Q = \left[q_{ij} \right]_{n \times n}, \quad \sum_{j=1}^{n} q_{ij} = 1, \ 1 \leq i \leq n.$$

That is, if the current request is from site i, the probability of the next request from site j is q_{ij}. We assume that the Markov chain $SITE_t$ is irreducible and aperiodic, and all states are positive recurrent, hence there is a unique stationary distribution satisfying:

$$\bar{\lambda} = \bar{\lambda} Q, \quad \left| \bar{\lambda} \right| = 1,$$

where $\bar{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_n)$ and the notation $\left| \cdot \right|$ stands for the sum of all elements in a vector.

Assume that OP_t is an independent sequence with identical distribution for all t:

$P\{ OP_t = read \} = r$, $P\{ OP_t = write \} = w$, $r + w = 1$.

Define random processes $STATE_t^{(i)}$, $1 \le i \le n$. $STATE_t^{(i)}$ takes its value in $\{ M, U \} \times \{ 1, 2, ..., n \}$ if $i = p$, or in $\{ F, S \} \times \{ 1, 2, ..., n \}$ otherwise. $STATE_t^{(i)} = (F, j)$, for example, means that the (secondary) copy in site i is fresh at time t and the current request for this file is from site j. Based on the assumptions made earlier, each $STATE_t^{(i)}$, $(1 \le i \le n)$ is a Markov chain with $2n$ states.

We study secondary copies first. If the secondary copy at site i is fresh, i.e., in one of the (F, j) states, $1 \le j \le n$, then it remains fresh if and only if the next operation is a read operation, no matter where the request comes from. On the other hand, if the secondary copy is stale or absent, i.e., in one of the (S, j) states, $1 \le j \le n$, then it becomes fresh if and only if the next operation is a read operation requested from the site i itself. Consequently, the transition matrix of $STATE_t^{(i)}$, $i \ne p$ is as follows (see also Fig. 4):

$$\begin{bmatrix} r\,\mathrm{Q} & w\,\mathrm{Q} \\ r\,\mathrm{Q}_i & \mathrm{Q} - r\,\mathrm{Q}_i \end{bmatrix}_{2n \times 2n},$$

where Q_i is the matrix with all 0 elements except its i-th column is the same as the i-th column of Q. Let $\overline{f}^{(i)} = (f_1^{(i)}, ..., f_n^{(i)})$, $\overline{s}^{(i)} = (s_1^{(i)}, ..., s_n^{(i)})$, where $f_j^{(i)}$, $s_j^{(i)}$ $(1 \le j \le n)$ are the stationary distribution of $STATE_t^{(i)}$. They satisfy

$$\overline{f}^{(i)} = \overline{f}^{(i)}\, r\, \mathrm{Q} + \overline{s}^{(i)}\, r\, \mathrm{Q}_i,$$
$$\overline{s}^{(i)} = \overline{f}^{(i)}\, w\, \mathrm{Q} + \overline{s}^{(i)}\, \mathrm{Q} - \overline{s}^{(i)}\, r\, \mathrm{Q}_i,$$
$$\overline{f}^{(i)} + \overline{s}^{(i)} = \overline{\lambda}.$$

Solving for $\overline{f}^{(i)}$ and $\overline{s}^{(i)}$, we have:

$$\overline{f}^{(i)} = r\overline{\lambda}\, \mathrm{Q}_i\, (\mathrm{I} - r\,\mathrm{Q} + r\,\mathrm{Q}_i)^{-1},$$
$$\overline{s}^{(i)} = w\overline{\lambda}\, (\mathrm{I} - r\,\mathrm{Q} + r\,\mathrm{Q}_i)^{-1}.$$

The probabilities of a secondary copy at site i being fresh $(P_{F,i})$ or not $(P_{S,i})$ are thus as follows:

$$P_{F,i} = |\overline{f}^{(i)}|, \quad P_{S,i} = |\overline{s}^{(i)}|.$$

Using this result, we can compute the expected number of file transfers (for reading or writing) per file operation in the polling and staling schemes. Namely,

$$t_{ps} = \sum_{i \neq p} \left| \bar{s}^{(i)} \, r \, Q_i \right| + w \; (1 - \lambda_p) = r \sum_{i \neq p} \sum_{j=1}^{n} s_j^{(i)} q_{ji} + w \; (1 - \lambda_p).$$

Furthermore, the expected number of multicast staling messages received per file operation anywhere in the net is:

$$a_{mst} = \sum_{i \neq p} \left| \bar{f}^{(i)} \, w \; Q \right| = w \sum_{i \neq p} \sum_{j=1}^{n} f_j^{(i)}.$$

The expected number of replicates in the polling or staling schemes is:

$$r_{ps} = \sum_{i \neq p} \left| \bar{f}^{(i)} \right| = \sum_{i \neq p} \sum_{j=1}^{n} f_j^{(i)}.$$

Similarly, the primary copy at site p changes its states among $\{(M, 1), ..., (M, n), (U, 1), ..., (U, n)\}$. The transition matrix of $STATE_t^{(p)}$ is (see also Fig. 4):

$$\begin{bmatrix} r\,Q & w\,Q \\ r\,Q - r\,Q_p & w\,Q + r\,Q_p \end{bmatrix}_{2n \times 2n}.$$

Hence its stationary distribution satisfies:

$$\bar{m} = \bar{m} \; r \; Q + \bar{u} \; r \; Q - \bar{u} \; r \; Q_p,$$

$$\bar{u} = \bar{m} \; w \; Q + \bar{u} \; w \; Q - \bar{u} \; r \; Q_p,$$

$$\bar{m} + \bar{u} = \bar{\lambda},$$

where $\bar{m} = (m_1, ..., m_n)$, $\bar{u} = (u_1, ..., u_n)$.

By solving the above equation, we get

$$\bar{u} = \bar{\lambda} \; w \; (I - r\,Q_p)^{-1} = \bar{\lambda} \; w \left(I + \frac{r}{1 - r q_{pp}} Q_p \right),$$

$$\bar{m} = \bar{\lambda} - \bar{u} = \bar{\lambda} r \left(I - \frac{w}{1 - r q_{pp}} Q_p \right).$$

Therefore, the probabilities of the primary copy being multiple and unique are:

$$P_M = \left| \bar{m} \right| = r \cdot \frac{1 - (w \lambda_p + r q_{pp})}{1 - r q_{pp}},$$

$$P_U = \left| \bar{u} \right| = w \cdot \frac{1 - r(\lambda_p - q_{pp})}{1 - rq_{pp}}.$$

Using this result, we can compute the expected number of broadcast staling messages received anywhere in the net per file operation

$$a_{bst} = (n - 1) \left| \bar{m} w \mathrm{Q} \right| = rw(n - 1) \frac{1 - (w\lambda_p + rq_{pp})}{1 - rq_{pp}}.$$

Note that in both broadcast and multicast staling schemes, the number of messages generated at the primary site is $\frac{a_{bst}}{n - 1}$.

4.2. Performance of Network File Systems with Locality

To simplify the modelling of locality, we consider a special case known as Easton's model. Easton's model was originally introduced for modelling the locality of page references in database systems. We adopt it to model our random process $SITE_t$.

In this model, the transition matrix Q is given as

$$q_{ii} = l + (1 - l) z_i,$$

$$q_{ij} = (1 - l) z_j, \quad i \neq j,$$

where l $(0 \le l < 1)$ and $\sum_{i=1}^{n} z_i = 1$. Clearly $\sum_{j=1}^{n} q_{ij} = 1$, $1 \le i \le n$ and $SITE_t$ is an irreducible aperiodic Markov chain with all states positive recurrent. Hence it has a unique stationary distribution $\bar{\lambda} = (\lambda_1, ..., \lambda_n)$. It is easy to verify that $\bar{\lambda} = \bar{z} = (z_1, ..., z_n)$. Therefore, Easton's model requires $n + 1$ parameters rather than n^2 parameters to specify its transition: the stationary distribution $\bar{\lambda}$ and the locality parameter l. If l is 0, the process is in fact an independent sequence. With l near 1, the next operation has high probability for coming from the same site as the current operation. By tuning l, we can model file operations with various degrees of locality.

Figure 5 illustrates the performance of a symmetric scenario which has the same parameters as in Figure 2 ($r = 0.8$, $w = 0.2$, $\lambda_i = 1/n$, $1 \le i \le n$), except the locality is 0.5 (in Figure 2, the locality is 0). It shows that the demand updating requires 40% fewer file transfers than the remote file access. The multicast staling messages are only about half of the polling messages. Figure 6 shows the effect of the locality parameter l in a symmetric scenario of eight sites. Note that locality does not affect the number of polling messages, and hardly affects the number of broadcast messages. However, the number of file transfers and multicast staling messages drops with increasing locality, such that their sum becomes less than the number of file transfers of a remote access system.

Figure 7 illustrates the performance of an asymmetric scenario which has the same parameters as in Figure 3, except that locality is 0.5 instead of 0. Replication saves about 40% of file transfers compared with remote file access. Figure 8 shows the effect of the locality parameter in an asymmetric scenario of eight sites (i.e., one file server and seven workstations). Again, the polling messages are unaffected. The number of multicast messages generated at the file server are, of course, also unaffected, but already quite low. However, locality significantly reduces the number of file transfers, and somewhat reduces the number of multicast messages received at each workstation.

5. CONCLUSIONS

Both polling and staling replicate files on demand. Our study shows that these schemes save a significant number of file transfers compared to remote file access, provided the frequency of read operations is higher than that of write operations. File access locality reduces the number of file transfers in the polling and staling schemes further, while remote file access cannot take advantage of locality. Remote access may require up to twice as many file transfers as the other schemes.

The polling and staling schemes require auxiliary messages, while remote access requires none. Surprisingly, the sum of file transfers and auxiliary messages in the multicast staling protocol is less than the number of file transfers in a remote access system. No such claim can be made about the polling and broadcast staling protocols. However, auxiliary messages are much cheaper than file transfers.

Polling performs poorly if the read ratio is high, because it generates numerous auxiliary messages to check whether files are up-to-date. Furthermore, the number of polling messages is unaffected by locality.

Broadcast staling performs poorly for large networks, because it loads each node with an auxiliary message for every write operation in the net. Broadcast staling performs worse than polling in the symmetric case, but better in the asymmetric case. The number of auxiliary messages is only insignificantly affected by locality.

Multicast staling clearly generates the smallest number of auxiliary messages, and this number decreases with increasing locality of reference. Polling requires twice the number of auxiliary messages in the symmetric case, and five times in the asymmetric case. Compared to broadcast staling in the asymmetric case, multicast staling has the advantage of generating a fixed load for each workstation independent of the network size, whereas broadcast staling generates a load proportional to the number of nodes.

An area for further study is automatic file migration: If a file is accessed frequently from a particular node, the primary copy should move to that node. File migration exploits the locality of write operations.

References

1. Rowe, Lawrence A. and Birman, Kenneth P., "A Local Network Based on the UNIX Operating system," *IEEE Transactions on Software Engineering* **SE-8**(2) (March 1982).
2. Brownbridge, D. R., Marshall, L. F., and Randell, B., "The Newcastle Connection or UNIXes of the World Unite!," *Software -- Practice and Experience* **12** p. 1147-1162 (1982).
3. Tichy, Walter F. and Ruan, Zuwang, "Towards a Distributed File System," *Proceedings of the Summer USENIX Conf.*, p. 87-97 (June 1984).
4. Popek, Gerald, Walker, Bruce, English, Robert, Kline, Charles, and Thiel, Greg, "The LOCUS Distributed Operating System," *Operating Systems Review* **17**(5) p. 49-70 ACM, (Oct. 1983).
5. Weinberger, P. J., "The UNIX Eighth Edition Network File System," pp. 299-301 in *Proc. of the 1985 ACM Computer Science Conference*, (March 1985).
6. Leach, Paul J., Levine, Paul H., Hamilton, James A., and Stumpf, Bernard L., "The File System of an Integrated Local Network," pp. 309-324 in *Proc. of the 1985 ACM Computer Science Conference*, (March 1985).
7. Satyanarayanan, M., "The ITC Distributed File System," in *Proc. of the Workshop on Operating Systems in Computer Networks*, ACM and IBM, Zurich, Switzerland (January 1985).
8. Comer, Douglas E., Korb, John T., Murtagh, Thomas, and Tichy, Walter F., "The TILDE Project," in *Proceedings of the Workshop on Operating Systems in Computer Networks*, ACM and IBM, Zurich, Switzerland (January 1985). 13 pages
9. Gifford, David K., "Weighted Voting for Replicated Data," pp. 150-162 in *Proceedings of the Seventh Symposium on Operating Systems Principles*, ACM, Pacific Grove, CA (December 1979).
10. Cheriton, David R. and Roy, Paul J., "Performance of the V Storage Server: A Preliminary Report," pp. 302-308 in *Proc. of the 1985 ACM Computer Science Conference*, (March 1985).

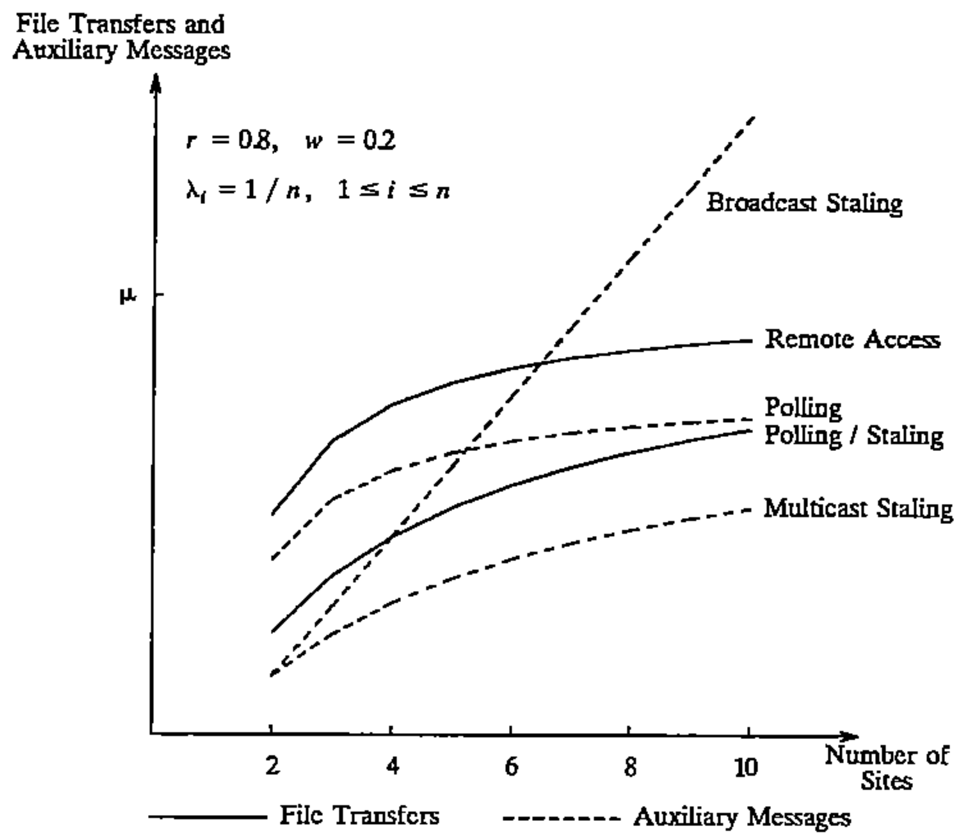


Fig. 2. A Symmetric Scenario (No Locality)

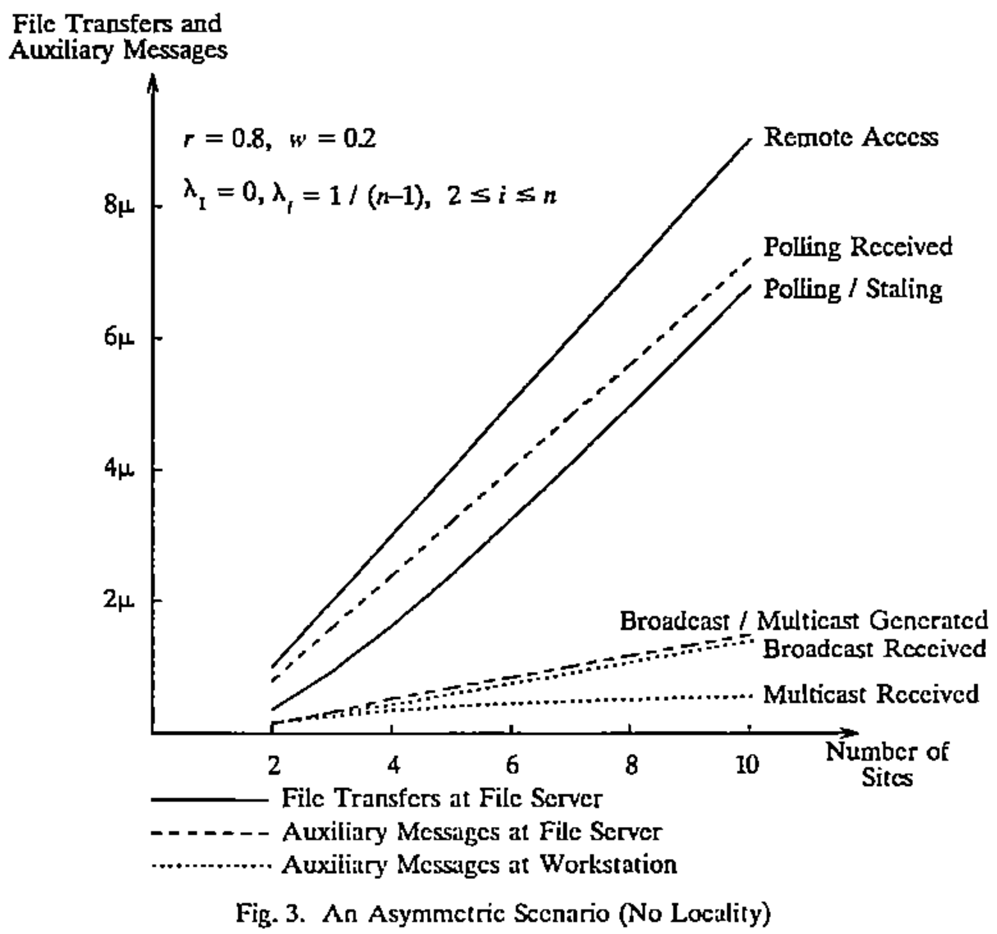
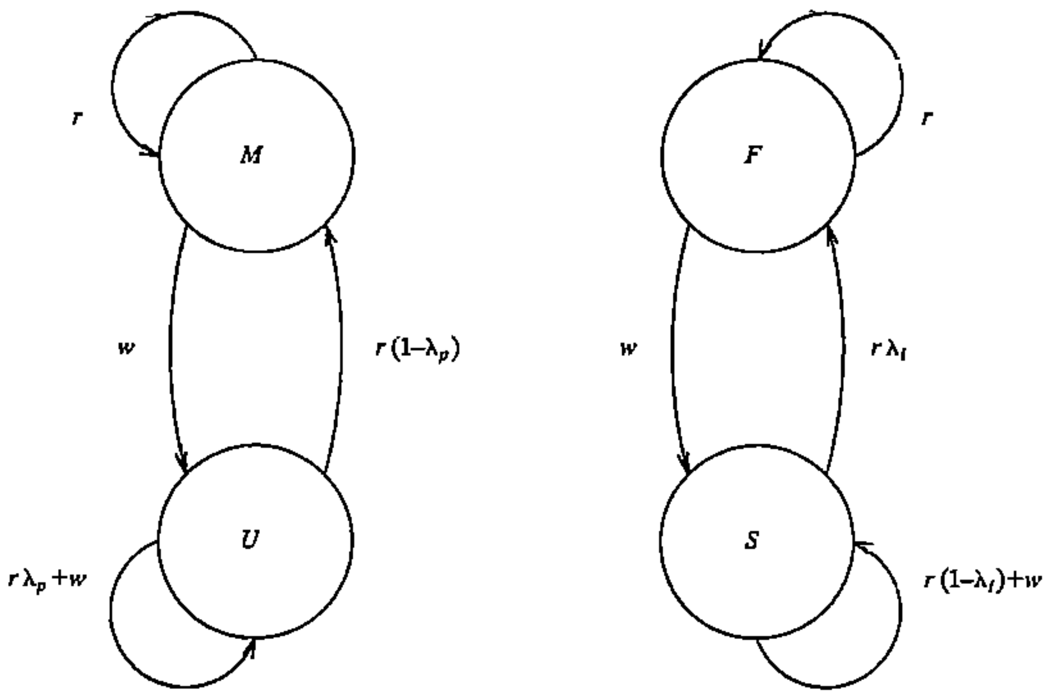


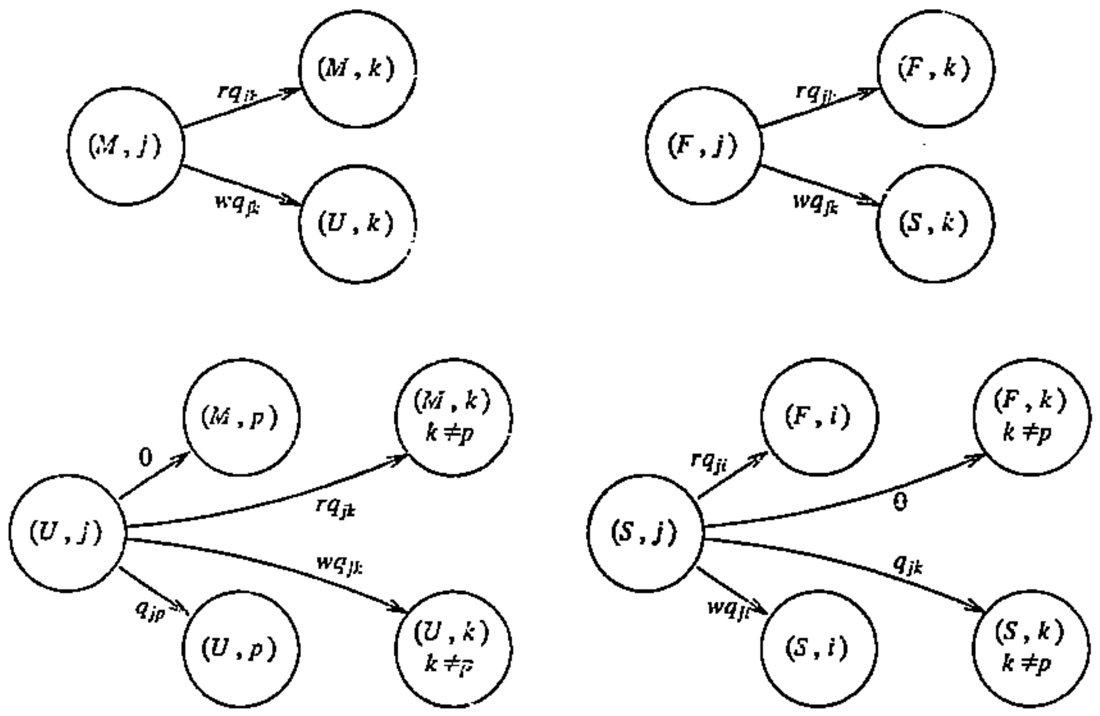
Fig. 3. An Asymmetric Scenario (No Locality)



Primary Copy at Site p

Secondary Copy at Site i

Fig. 1. State Transition Diagram



Primary Copy at Site *p* Secondary Copy at Site *i*

Fig. 4. State Transition Diagram

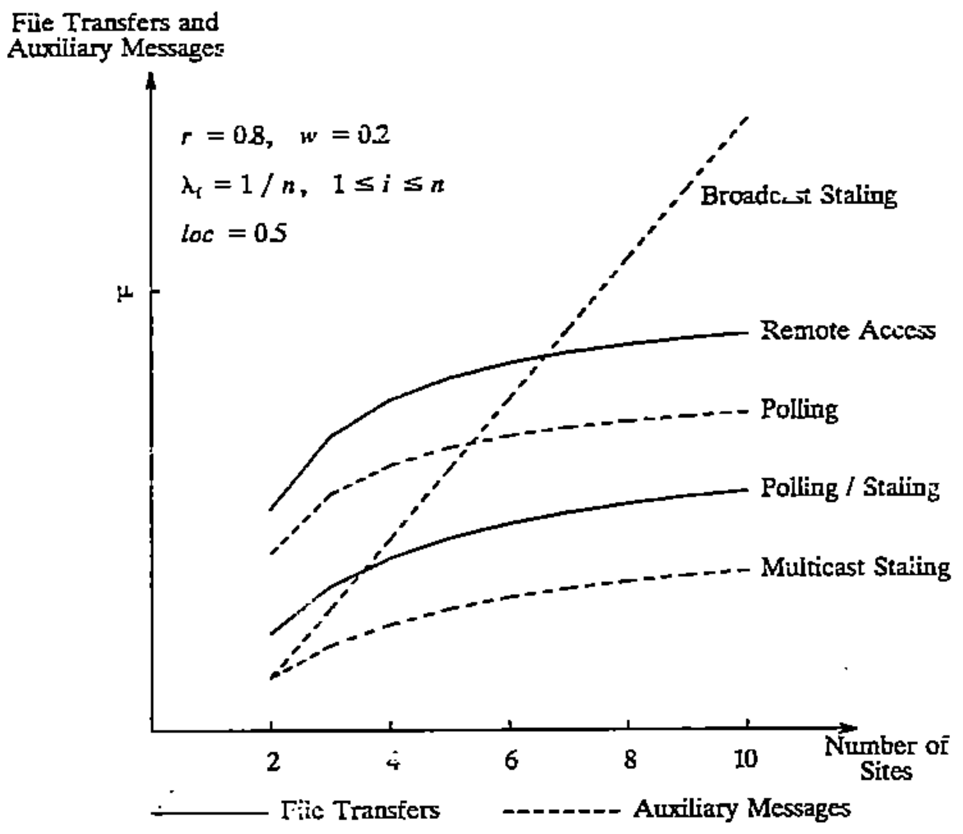


Fig. 5. A Symmetric Scenario with Locality 0.5

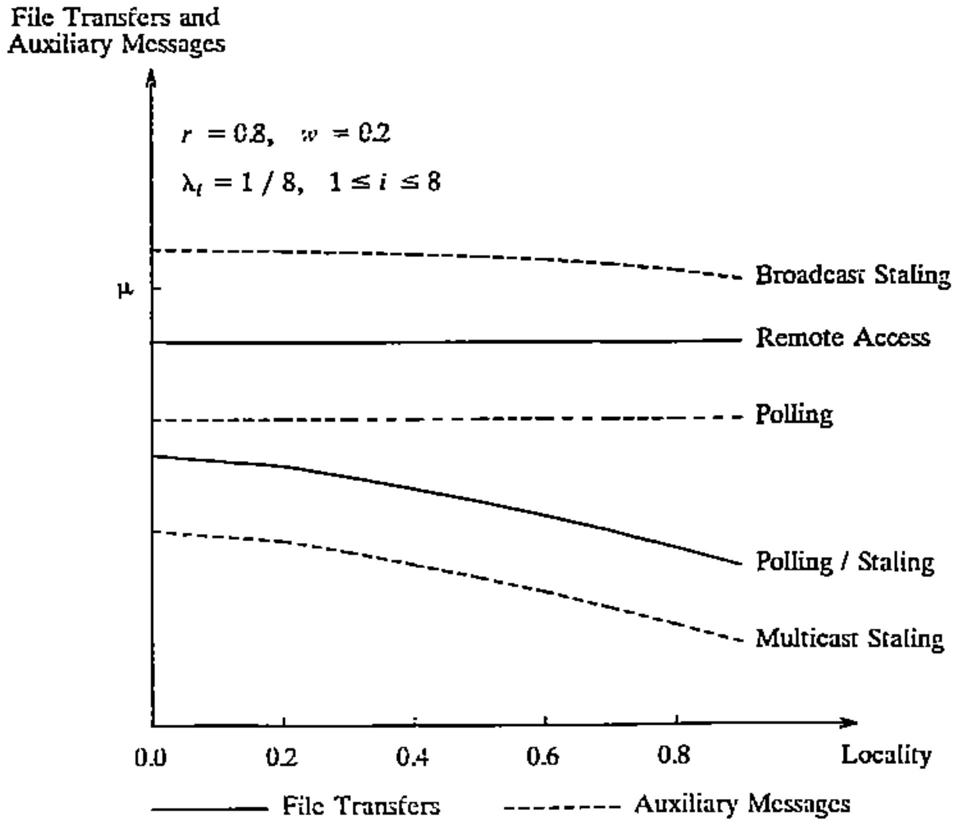


Fig. 6. A Symmetric Scenario of 3 Sites with Varying Locality

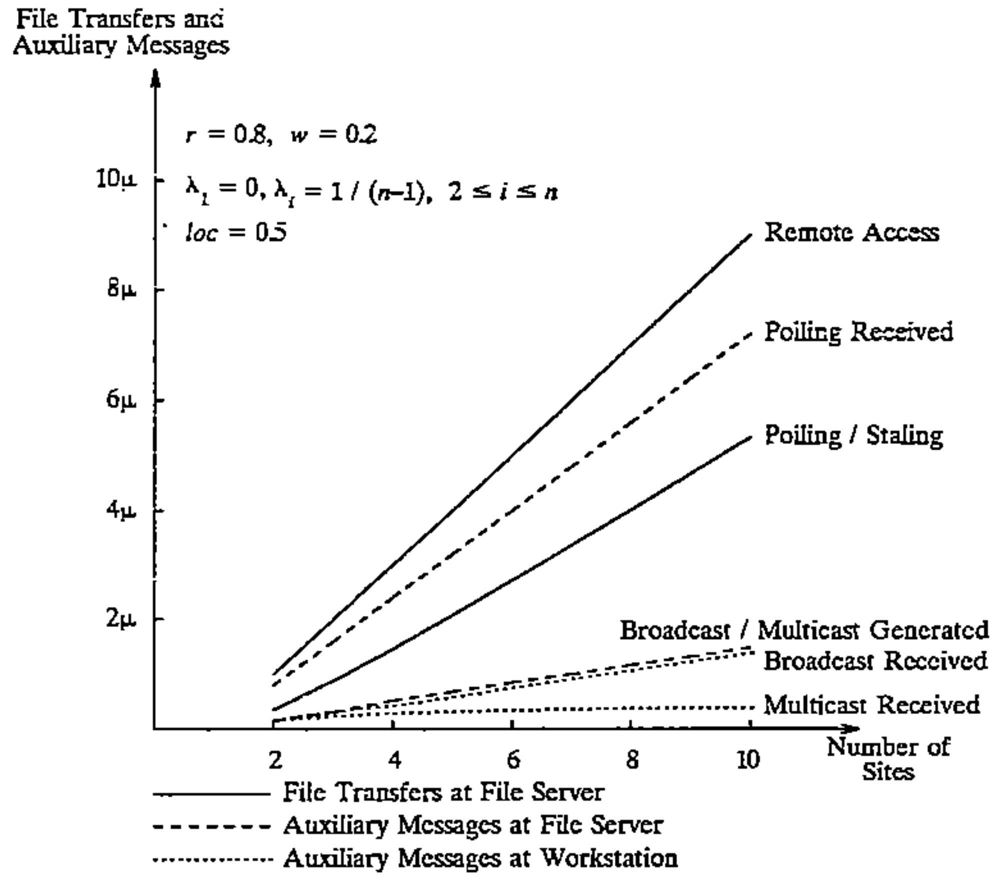


Fig. 7. An Asymmetric Scenario with Locality 0.5

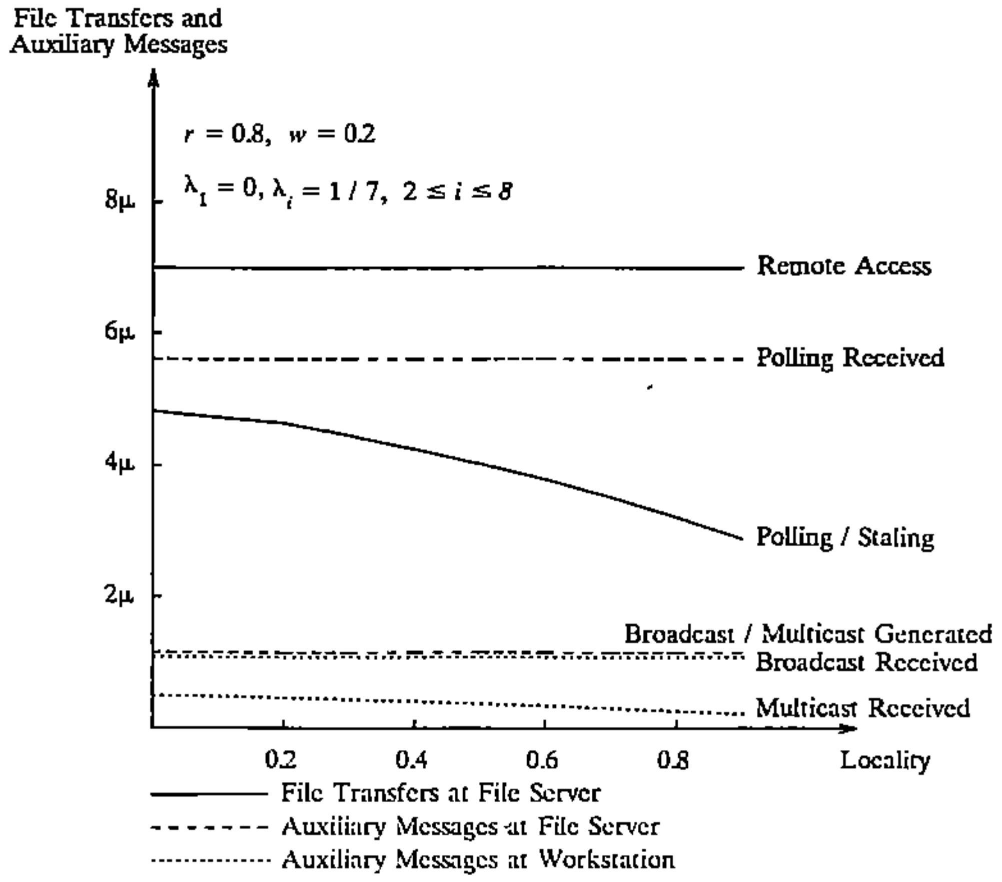


Fig. 8 An Asymmetric Scenario of 8 Sites with Varying Locality