NAME SERVICE LOCALITY AND CACHE DESIGN IN A DISTRIBUTED OPERATING SYSTEM*

Alan B. Sheltzer
Robert Lindell
Gerald J. Popek

University of California, Los Angeles

ABSTRACT

Name service is an important component in the overall behavior of distributed operating systems. The use of a distributed name cache to improve system performance and reduce the elapsed time to access remote resources is explored. A classical cache design evaluation, applied to the problem of distributed name management, is presented. Reference strings, collected from a production distributed system, are used as input to a trace-driven simulation to determine the degree of locality exhibited in name to object translation, evaluate cache parameters, and justify the utility of a distributed name cache.

1. Introduction

Operating systems which provide a flexible, hierarchical name space often spend a substantial portion of their time supporting the mapping of user specified names into the low level descriptors necessary to access resources; 40% of Berkeley Unix overhead is reportedly consumed in this manner[1]. Nevertheless, the independence between a resource's name and its location on disk or elsewhere is very valuable. In a distributed operating system, this independence between resource name and location, called *network transparency*, is especially important. Unfortunately, the cost is also potentially greater, since interaction with remote name service may be needed in addition to any relevant disk lookup and processor activity.

In this paper, we investigate the costs of name service in a distributed operating system environment, and find that it is an important component of the system's overall behavior. We then consider whether a distributed cache system, with its underlying assumptions of locality and requirement for multicache consistency, is a suitable solution. A classical cache design evaluation is performed by collecting extensive reference strings from a production distributed system environment and simulating various cache sizes, invalidation algorithms, and multicache consistency strategies.

A relatively simple cache design is found to be quite effective. The resulting improvement in overall system performance and reduction in elapsed time to access remote resources helps extend the feasibility of transparent distributed systems to much larger scale systems, including those with network links of lesser quality than typical LANs.

1.1 Caches

Caches are typically placed between a large, relatively slow and inexpensive source of information and a much faster consumer of that information. The cache capacity is relatively small and expensive, but quickly accessible. The goal is for the cache behavior to dominate performance but the large storage facility to dominate costs, thus giving the illusion of a large, fast, inexpensive storage system. Successful operation depends both on a substantial level of locality being exhibited by the consumer, and careful strategies being chosen for cache operation - disciplines for replacement of contents, update synchronization, etc. The value of successful caches often is enormous, representing the difference between satisfactory cost/performance and failure.

Caches have been used for many years between main memory and the central processor. Recently, distributed caches have appeared in integrated hardware systems constructed of multiple processors. In these multiprocessor architectures, each processor has a private cache, and a mechanism exists to prevent the simultaneous existence of different versions of the same data block in different caches. The considerations in the design of multiprocessor hardware caches are present, in analogous ways, in a distributed operating system, and their proper resolution is of

*This research has been supported by the Defense Advanced Research Projects Agency under contract DSS-MDA-903-82-C-0189.

similar importance.

1.2 Name Service in a Distributed Environment

In a system with a *location transparent* name space, it is the system's responsibility to find the resource given its name, and set up all necessary bindings for subsequent access. The cost of this distributed name lookup, and the associated update of relevant (perhaps partially replicated) tables as resources are created, destroyed, and moved, can be a major cost of the distributed system's operation. In a production Locus installation[2] for example, it is not uncommon for half of total network traffic to be in support of name service.

An immediate question is whether this key function is susceptible to cache based speedup methods. If so, the performance improvement can be remarkable. However, one must first determine whether name lookup exhibits the requisite degree of locality, and whether the necessity to invalidate other systems' cache entries when a name service update occurs represents significant cost, complexity and delay. If these characteristics are satisfactory, one can then proceed with the determination of the other relevant cache design parameters.

In order to address these issues, over 15 million "name-service" reference string entries were collected during normal operation of a production Locus system. Each such entry represents a path name element in the distributed Unix directory hierarchy. These measurements serve as a basis for much of the discussion in this paper. The reference strings were input to a trace-driven simulation which was used to determine the degree of locality in directory referencing, determine directory cache parameters, and evaluate the overhead of maintaining multicache consistency.

1.3 Organization of this Paper

We first briefly review the Locus operating system, including the procedure followed in name service. Cache design issues are then raised, and the data which was collected is summarized; some of it is interesting in its own right. We then lay out the design of a distributed directory cache for Locus, and evaluate its effectiveness. The results are discussed, their applicability to other environments is postulated, and future work is outlined.

2. The Locus Distributed Operating System

Locus is a distributed version of Unix that runs on high speed, low delay local area networks. Locus provides a fully transparent, distributed file system as well as transparent support for distributed processes. Networks of heterogeneous cpu types are also handled transparently.

The file system appears to the user and applications software as a single, tree structured name space with one root, across the entire collection of machines. Changing the storage site of a file does not require changing its name. Important files are replicated below the user visible level, and the system is responsible for keeping the copies mutually consistent.

Process execution in Locus is similarly transparent. It is possible to create (i.e. fork) processes locally or remotely, with exactly the same semantics. Processes may migrate to a similar cpu type while in the midst of execution without effect on continued correct execution. Processes interact with one another across machine boundaries in the same manner as if they were co-located (e.g. Unix signals and ipc operate transparently networkwide). Besides providing access to remote resources through the same interfaces as local ones, Locus also achieves a substantial degree of *performance transparency*. The delay in access to remote resources typically is little different than the delay in access to local resources.

Locus has been extended to operate in an internetwork environment consisting of LANs interconnected by long haul networks. Full transparency is maintained with remote performance for typical interactive activity approaching or equaling local performance even when resources are located across a long haul link.

2.1 Single site pathname expansion

In a hierarchically structured file system such as Unix or Locus, each file in the system is either a data file or a directory. Directories contain entries of the form <string name, file descriptor pointer> where the file descriptor pointer is an index into a table of file descriptors (called "inodes" in Unix) and each inode contains the device addresses of the actual data pages for the target file, plus status information. An object is referred to by a pathname which is a sequence of directory names separated by slashes and ending in a file name. A pathname starts either at the root directory or the current working directory.

The system maps a name to an object by reading the first directory component in the pathname and then sequentially searching the entries within the directory for a match on the string name of the next pathname component. If an entry is found, the file descriptor pointer of the entry is used to locate the device addresses for the next pathname component. If the next component is the last component in the pathname, the target object has been found. If not, the pathname component is a directory and search-

ing continues.

2.2 Distributed pathname expansion

In a distributed system, the directories and target object that are referred to by a pathname may be stored at sites other than the site that requests the target object. There are two approaches to pathname expansion in a distributed environment. In the first approach, called *transparent pathname expansion*, each directory is brought across the network from the storage site and searched at the using site (the site that requests the pathname expansion). This approach has the advantage that distributed operation is identical to single site operation as long as there is a transparent mechanism for reading remote pages. However, a substantial amount of network traffic may be generated during pathname expansion as each directory component is opened, read, and closed.

In the second approach called *remote pathname expansion*, the pathname is expanded at each site that stores a pathname component. When the using site finds a pathname component that is stored remotely, it packages the remainder of the pathname into a network request message and sends it to the site that stores that component. The storage site services the request by continuing pathname expansion, opening and searching the directories locally. If all of the remaining components are stored at the storage site, then the result of the pathname expansion is returned to the using site. If the storage site finds that a component is stored remotely it sends the remainder of the pathname to the next storage site and pathname expansion continues there. This approach reduces network traffic for pathnames that contain many directories all stored at a single remote site. However, if the pathname contains directories that are stored at several sites, network traffic is not reduced. More importantly, if there is a significant level of user program access directly to directory pages, substantial directory page traffic results in addition to remote pathname expansion messages. Measurements of the UCLA Locus network indicate that approximately 20% of all commands issued require user program access to directories. Common tasks such as listing the contents of a directory, copying or removing all entries in a directory, expanding wildcard arguments, and accessing parent directories, all use information found in directory pages. With remote pathname expansion, directory pages remain at the storage site, so this approach does not effectively reduce message traffic in many important cases.

The Locus distributed operating system supports transparent pathname expansion. References to a remotely stored directory that is currently open at a using site do not generate network traffic if the directory pages are found in the local buffer cache. However, when a remotely stored directory is closed, all of the associated file pages are flushed from the buffer cache to insure that only one version of the directory exists in the system. Thus, the next open of a remote directory involves rereading all the directory pages to be searched.

An example of the network traffic generated to expand a pathname (*/th/foo/file1*) under Locus where the directories and target file reside at a site different than the using site is shown in figure 1.

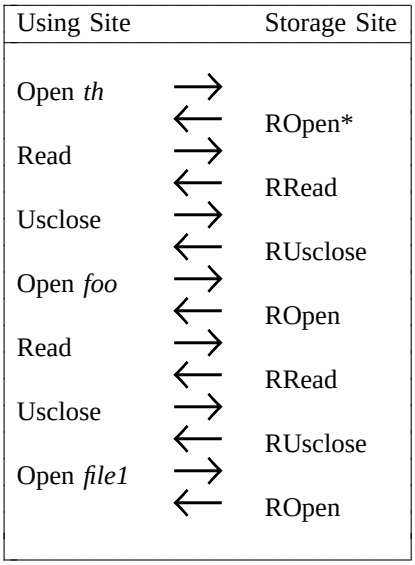

Figure 1: Message Traffic for Pathname Expansion of */th/foo/file1*

The *th* directory is opened and read across the network. The directory is searched for an entry that contains the string name *foo* and when such an entry is found, the *th* directory is closed. Next, the "foo" directory is opened and read across the network and the string name *file1* is searched for. When a match is found, the *foo* directory is closed and the target data file (*file 1*) is opened.

Fortunately, the activity of most users is usually confined to a small, slowly changing subset of the entire name hierarchy. Furthermore, most directories have a high read to modify ratio. This behavior implies that the number of messages due to pathname expansion can be reduced by caching directory pages and related file descriptors at each site even after the directory has been closed. If significant directory reference locality exists, then current references to directory pages are likely to be found in the directory cache and will not generate network traffic.

---

*Responses are indicated as Rmessage-type, ie. the Open Response is shown as ROpen. The explicit ACK messages sent for each Locus message (except for Read and RRead) are not shown.

Total network traffic will be decreased and more importantly, the elapsed time to reference remotely stored objects will be reduced.

3. Issues in distributed cache design

In order to properly evaluate the potential effectiveness of a cache, it is important first to understand the necessary design goals and issues. Then measurements and simulations can be used to determine specific parameters. The goals in the design of a hardware or software cache[3] are:

1. Maximize the probability of finding a reference in the cache (hit ratio)

2. Minimize the time to access the information that is in the cache (access time)

3. Minimize the delay due to a miss

4. Minimize the frequency and overhead of invalidating a cache entry.

A careful selection of cache parameters, including cache size and replacement algorithm, is necessary to achieve these goals. A distributed cache presents the additional problem of guaranteeing multicache consistency. If caches at several sites store a copy of the same information (e.g. a directory page) and one of the sites modifies the information without notifying the others, an inconsistent state results with possible disastrous consequences.

The hit ratio for the cache is improved by increasing the cache size, given a significant amount of directory locality. However, the memory requirements for the system and the time to access a given cache element also increase as the cache size increases. Furthermore, some directory references are unique (never rereferenced), so increasing the cache size beyond a certain point will not improve the hit ratio. In a distributed cache, the overhead of maintaining multicache consistency increases as cache size increases because a larger cache increases the probability that a site is caching a page that must be invalidated.

There are several approaches to maintaining consistency for a multisite directory cache. In the simplest approach, directory pages are flushed from the cache whenever the directory is closed. A request to open for modification is blocked until all sites have closed the requested directory. This policy insures multisite cache consistency since a site that has a directory open for modification knows that no other valid copies of the directory page(s) exist at other sites. However, the cache is only useful during the period that a directory is open.

In another approach, directory pages remain in the cache after the directory has been closed. Whenever a storage site receives a request for modification for a directory page, it broadcasts a message to all other sites identifying the directory page. Each site examines its cache for the page and invalidates it if present. Although this scheme may work well for a few sites, as the number of sites increases, the network traffic generated becomes prohibitive.

A better solution is for the storage site to record the sites that have requested a given directory page. When the storage site receives a request to modify the directory page, only those sites in the list are notified to flush the page from their caches. The overhead for cache invalidation is acceptable if there is a low rate of directory modification to directories that are shared among sites and if only a small number of sites share a directory when that directory is modified.

When the directory cache is full and a cache miss occurs, some existing cache entry must be replaced with the target reference. A good replacement algorithm is necessary to achieve a satisfactory hit ratio. Choices include a global LRU algorithm, LRU per process or user, or first in first out. The delay due to removing the selected entry from the cache and bringing in the target reference must also be minimized.

4. Directory reference measurements

A trace-driven simulation was used to investigate directory reference locality and evaluate different design choices for the distributed directory cache. Directory reference strings were collected on each site of the 15 site UCLA Vax Locus network for 10 hour periods for 6 days. That network commonly supports more than 150 active users during the five hour busy period of each day. There are various organizations administrating subsets of the machines; as a result, certain communities of users regularly cross many machine boundaries, while other users' activity is primarily local.

An event trace was recorded each time the operating system referenced a directory component during pathname expansion. Each event contained the name of the directory reference, its file descriptor, an indication of whether the directory was stored at the site that issued the reference or at some remote site, an indication of whether the directory reference was found in the existing buffer cache of the using site, the type of reference (for read or for modification), a networkwide timestamp, plus other in-

formation. References to directories in which the directory was the target file (e.g. directory listing commands) were not recorded. Approximately 2 to 3 million entries were collected per day.

The reference strings for each site were combined and sorted by time. Locus contains an intersite time synchronization facility that keeps the clocks on all sites within a few milliseconds of one another. Cache simulations were run on the combined, multisite reference string so the caches on all sites were simulated simultaneously. The extent of directory sharing between sites and the effect of multisite cache invalidation could thus be evaluated.

4.1 Measurement results

The data which was collected showed differences in certain measurements among sites, but for any given site, the measurements were quite stable.

For a given site, the percent of references for remotely stored directories, the percent of references for modification, and the percent of remote references not found locally, showed little variance during the 6 days of measurements. For all sites, the percent of directory references for modification also varied little, with an average of 2.5% and a standard deviation of 0.5. The variance among sites for the local versus remote values was more pronounced because different sites support different mixes of local and remote service. Although local directory references were dominant on all sites, the percent of references for remotely stored directories varied from 2.3% to 14.6%. The percent of remote references not found at the using site varied from 20.2% to 83.2%. On sites where remote traffic was mostly due to the activity of background processes that kept directories open, this percentage was low, but on sites where remote traffic supported normal interactive computing, the value was much higher.

Directory reference traces collected for 10 hours for a single example site are shown in figure 2. As expected, most of the references are local (89%). However, a significant portion (71%) of the remotely stored directory pages must be brought across the network during pathname expansion. The remaining references to remotely stored directories are references to directories that are currently open locally and are therefore found in the buffer cache ("incore") of the using site. The goal of a directory cache is to increase the number of remotely stored directory pages that are found incore at the using site. The number of directory references that are requests for modification is quite low (1.3%) but single site measurements are not enough to determine the amount of directory sharing when modification is requested. The results from the trace-driven simulation using the combined data from all sites is needed to determine the overhead of multicache invalidation.

	References		
	Total	Local	Remote
# of references	354491	317199	37292
% of references	100.0	89.48	10.52
% for mod	1.27	1.07	2.99
% not incore	17.41	11.12	70.87

Figure 2: Results of Directory Reference Event Collection

4.2 Locality in Directory Referencing

The property of "locality of reference" has been observed in program execution[4,5], file access[6], as well as database access[7]. We are interested in the degree to which locality is also exhibited by operating system functions and in name to object translation in particular. The operating system constantly searches for information (e.g. the translation from string name to object) and then discards this information when in fact, the information may be used again in the near future. If locality exists, caching recently used information is likely to reduce the overhead of operating system management.

A measure of the locality present in a reference string is given by Rodrigiuez-Rosell[8] and used by Kearns[7] to demonstrate locality in database reference strings:

$$L(t,\tau) = w(t,\tau)/\tau$$

where $L(t,\tau)$ is interpreted as the instantaneous locality measure at time t. This measure can be applied to directory reference strings by setting:

τ = window size in # of directory references
$w(t,\tau)$ = working set size at time t for window τ
= the # of distinct directories referenced among the τ most recently referenced directories in the reference string

Averaging $L(t,\tau)$ over the number of references gives the average locality $\overline{L(\tau)}$ for a given window size. A reference string that exhibits little rereferencing and thus has poor locality will have a value of $\overline{L(\tau)}$ near 1 over a wide range of window sizes. If there is substantial rereferencing in a

reference string, then as the window size increases we expect $\overline{L(\tau)}$ to decrease rapidly as references are likely to be found in the current working set so the average working set size does not increase.

A plot of the average locality $\overline{L(\tau)}$ versus window size for the example single site directory reference string is shown in figure 3.

Figure 3: Average locality $\overline{L(\tau)}$ as a function of window size τ for directory references

The sharp decrease in $\overline{L(\tau)}$ implies a significant level of locality in directory referencing and suggests that a directory cache of reasonable size would be very beneficial. At a window size of about 40, the curve flattens out, indicating that some small number of directories are never rereferenced.

4.3 Trace-driven simulation results

A trace-driven simulation was used to evaluate directory cache size, replacement algorithms, and the overhead of maintaining multicache consistency. A plot of the miss ratio versus cache size for a directory cache using a global LRU replacement algorithm is shown in figure 4. The miss ratio is the probability of not finding a remotely stored directory during pathname expansion at the using site. The cache size is given in number of directory pages with the assumption that each directory is stored on a single page. The cache size would be slightly larger to support the small number of directories stored on more than one page. A cache size of just 15 directory pages for the Locus site considered in figure 2 reduces the miss ratio from about 71% without a directory cache to just 11%. Almost 95% of the remotely stored directory references will be found at the using site with a cache of 40 pages.

Figure 4: Miss ratio versus window size for a directory cache

Therefore, in most cases, the elapsed time to access remote objects will be greatly reduced since 95% of all accesses to remote objects will not include the overhead of pathname expansion. The hit ratio for all sites with a cache of 40 pages varies from about 87% to 96%. Increasing the cache size further does not appreciably improve the miss ratio as some small number of directory pages are never rereferenced.

Instead of using a single, global cache at each site, a cache could be dedicated to each user. Cache entries would be replaced as each user cache filled up. Our measurements show that the number of active user ids per site varied from a few to over twenty. Users frequently execute processes remotely, especially on server sites, thus increasing the number of user ids per site. Results from the trace-driven simulation show that although a cache per user (with a per user LRU replacement algorithm) uses a smaller cache size per user to achieve the miss ratio of the global cache, the total number of pages dedicated to the directory cache is substantially greater.

When a request for modification is received by the storage site of a directory, the storage site must send a cache invalidation message to each site that currently has the page in its cache. Both the number of directory references that require cache invalidation messages and the number of sites that must receive cache invalidation messages must be very low for the distributed cache to be effective.

Data from the directory reference traces for a single day indicate that over a 10 hour period there were 2,474,407 total directory references issued by all sites. Results from the trace-driven simulation show that even if each site provides a cache of 60 directory pages, only 1265 or 0.051% of the references require cache invalidation. The distribution of the number of sites that need to be sent cache invalidation messages for each reference that requires cache invalidation is shown in figure 5 for a cache of 60 pages. Only 93 references or .0038% of the total number of references require more then a single cache invalidation message. Thus, the overhead of maintaining multicache consistency is quite low and should not be costly even as the number of sites in the system grows substantially.

Figure 5: Distribution of # of sites that require cache invalidation messages

A plot of the miss ratio versus cache size for a single site cache (without cache invalidation) and the miss ratio versus cache size for a distributed cache where cache invalidation is generated by the simulation, is shown in figure 6. The slightly higher miss ratio of the cache invalidation plot is mostly due to counting all references for modification as misses (since they must be serviced by the storage site) rather than from the removal of invalidated entries from the cache.

5. Operation of the Distributed Directory Cache

Pathname expansion with a directory cache works as follows. When a using site searches a remotely stored directory during pathname expansion and the directory pages are not found in the directory cache at the using site,

Figure 6: Plot of miss ratio with and without cache invalidation

a "no open read" (NOR) request message is sent to the site that stores the directory. The storage site returns the file descriptor (inode) and the first directory page to the using site, which enters the items into the local directory cache. The storage site adds the using site to a list of all sites that currently have that file open for NOR. Additional directory pages are read by the using site as usual and stored in the cache. Remotely stored directory pages are removed from the using site directory cache on an LRU basis.

When a cache miss forces a directory page to be removed from the cache, the storage site must remove the using site from the NOR list for that directory. To reduce the delay due to a cache miss, an explicit "removed from cache" message of an NOR directory is not sent to the storage site, but is instead piggybacked on the next open message to that storage site. A table of the last few NOR "removed from cache" messages are kept for each storage site to be sent with the next open message. If an overflow of this table occurs before the next open message is sent, a "removed from cache" message is dropped. Given the low rate of cache invalidation, the overhead of sending an invalidation request to a site that doesn't actually have the directory in its cache should be minimal.

All opens for modification are sent to the site that stores the directory. When a request for modification is received for a directory which has a non-null NOR list, the storage site sends a message to each site in the list to invalidate the file descriptor and pages associated with that directory. After acknowledgement is received from all sites, the storage site opens the directory for modification. After the directory has been closed for modification, it can

be reopened NOR by other using sites.

6. Discussion and Future Work

The primary goal of the distributed name service cache discussed in this paper is the reduction in system response time to users' remote requests. A secondary goal is to reduce overall network traffic, and the corresponding system overhead that message sending and receiving generates. The use of a distributed cache for Locus substantially improves remote performance and reduces network traffic, as expected. The degree to which overall system performance is improved varies, depending on the level of remote behavior actually taking place. Caching is operational in a Locus testbed used for systems research; at the time this paper was written, the implementation was being prepared for production use. Available experience in its use corresponds well to the results of reference string analysis reported in this paper.

Elsewhere[9], the utility and feasibility of full transparency, even in networks with much higher delay and bandwidth limits than LANs, is discussed. It was concluded that transparency is quite attractive and feasible. The distributed caching discussed in this paper is especially promising in those internet systems, because of the observation that a substantial portion of network traffic is naming related, and since great reduction of such traffic is especially beneficial in internet environments.

While it is believed that the reference strings which have been collected adequately represent typical use patterns, nevertheless there are other environments which need to be examined. One wishes to investigate a system on which there is a different application mix; commercial rather than engineering and scientific activities, for example. It would also be instructive to conduct similar studies for systems other than Locus.

A workstation oriented network presents a different situation, also. The network we considered consisted of multiuser systems, with an average of approximately ten users being served per machine. An effective cache size was of the order of 30 pages. In a single user node, we expect that the desired cache size will be correspondingly smaller, perhaps just a few pages. The reference strings will be reanalyzed to indicate what cache sizes may be appropriate in this case.

In summary, name service in the distributed environment which was studied exhibited a high level of locality, with an exceptionally low level of conflict between modification to directories at one site and their concurrent use elsewhere in the network. Further, the implementation effort to support the cache disciplines described in this paper is not complex. Thus we conclude that a distributed name service cache can be implemented in a reasonable manner, and it can have a substantial positive effect on distributed system performance.

References

[1] M. Karels, private communication, 1984.

[2] B. Walker, G. Popek, B. English, C. Kline, G. Thiel, "The LOCUS Distributed Operating System", Proceedings of the 9th ACM Symposium on Operating System Principles, Oct., 1983.

[3] A. Smith, "Cache Memories", Computing Surveys, Vol. 14, No. 3, September 1982.

[4] P. Denning, "On modeling program behavior" in Proc. Spring Joint Computer Conference, vol. 40, AFIPS Press, 1972, pp. 937-944.

[5] A. Madison and A. Batson, "Characteristics of Program Localities", CACM, Vol. 19, No. 5, May 1976.

[6] S. Majumdar, "Locality and File Referencing Behavior: Principles and Applications", M.S. thesis, University of Saskatchewan, August 1984.

[7] J. Kearns and S. DeFazio, "Locality of Reference in Hierarchical Database Systems", IEEE Transactions on Software Engineering, March 1983.

[8] J. Rodriguez-Rosell, "Empirical Data Reference Behavior in Data Base Systems", IEEE Computer, November 1976, pp. 9-13.

[9] A. Sheltzer, "Network Transparency in an Internetwork Environment", Ph.D. Dissertation, Computer Science Department, University of California, Los Angeles, 1985.