1988

Locating Distributed Information

Craig E. Wills

Report Number:
88-770

Wills, Craig E., "Locating Distributed Information" (1988). *Department of Computer Science Technical Reports*. Paper 659.
<https://docs.lib.purdue.edu/cstech/659>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries.
Please contact epubs@purdue.edu for additional information.

LOCATING DISTRIBUTED INFORMATION

Craig E. Wills

CSD-TR-770
May 1988

Locating Distributed Information*

Craig E. Wills

Department of Computer Science
Purdue University
West Lafayette, IN 47907

CSD-TR-770

May, 1988

Abstract

As part of research into transparent execution of computational services in a distributed environment, two problems we have encountered are how to manage and locate information about service availability and the status of server machines. The key observation about these two problems is that they are not isolated problems, but each is an instance from a class of distributed information problems. The class of problems is characterized by the inherent distributed nature of the information where each server machine supplies a portion of the total information.

In this paper we first examine general techniques for managing distributed information and how various evaluation criteria, when applied to the techniques, are affected by the parameters and scale of a specific problem. We then illustrate the use of the techniques and criteria for the problems of locating service and server status information, as well as determining the current status of a user in a distributed environment. In each case we find the best technique to use for the problem based on its specific parameters.

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

1 Introduction

Our research [Wil88] has been concerned with understanding the issues and problems of transparently executing computations in a *computing engine*, a distributed environment of machines loosely coupled with high-speed local area networks. The central idea of our approach is that named computations available to the user in a distributed system should be treated as *services*, where a service is offered by one or more machines. Our approach is to separate the identification of a service from its execution. As part of our work we have designed and built a prototype of a *service execution mechanism* that provides the user consistent and uniform access to computations from multiple machines in a distributed environment.

Two issues in the design of the service execution mechanism are how to manage and locate information about service availability and the status of server machines in the computing engine. Location of services available from the server machines is a central problem in determining if a service is available in the computing engine and if so, which machines provide the service. Retrieval of status information about the server machines, specifically their load, is needed in selecting which machine to use for invoking the service.

The key observation about these two problems is that they are not isolated problems, but each is an instance from a class of distributed information problems. This class of problems deals with status or availability information that is supplied on a per-machine basis. This class of problems arises not only for locating service and machine status information, but also for such problems as locating the current status of a user interacting with machines in a distributed environment or locating the machine that contains a particular resource. The collective information available from all machines defines a distributed database. Traditional distributed database designs maintain replicated, consistent copies of a database in the face of machine failures and multiple sources of updates. Our

class of problems is distinguished from the traditional problem by the following characteristics.

- The information is inherently distributed with each server machine supplying a portion of the total information. Although information may be replicated on other machines, each piece of information has only one source of updates.
- Slightly out-of-date copies of the information can be used but should eventually converge to the most current information. Weaker consistency requirements for copies of the information allow updates to be made without using commit protocols to ensure consistency of all copies.
- Performance in locating information is important. Depending on the desired consistency constraints, accuracy of the obtained information may be traded off for quicker access to the information.
- Information is volatile and may go away. Because all information is particular to a machine, failure of a machine makes its information unavailable and copies of the information to be invalid.

The last characteristic leads to the need to detect and possibly measure *negative* information. We define negative information to be information about the unavailability or negative status of a resource. Examples of detecting or measuring negative information are determining if a service or file is not available from a server machine, how long a user has been logged out, or if logged in then how long he has been idle. Techniques used for location of information may not be appropriate, or have to be modified, for detection or measurement of negative information. For example, querying a machine directly can be used to obtain its current status, but if the machine fails the same technique cannot be used to find out how long it has been down.

Thus, the problems we wish to investigate are actually just instances of a class of distributed information location problems. Rather than investigate the problems directly, we examine different techniques for storing, updating, and accessing information for the class of problems. Our examination of the appropriate technique for a problem is similar to Mullender and Vitányi's [MV85] study of

distributed match-making between a process asking for a service and a process giving that service. However, our problem space is more general, and our approach is an analytical rather than a theoretical comparison of the techniques. In other related work, Ammar et al. [AAA88] address the specific issue of how system parameters affect the performance of location operations when hint tables (caching) are used.

Our objective is to understand the general issue of how to choose a technique based on the parameters of a problem. The parameters include how often the information is accessed, how often the information is updated, if the location of the resource can be determined from its name or indirectly through a database, and the communication costs to access information on other machines. Communication costs may vary widely depending on whether the communication medium is a local area or long haul network.

Choosing a technique for a problem implies evaluating each technique to determine which minimizes potential costs. A number of criteria can be used to measure costs such as the response time to access information, number of network packets, and accuracy of the information. In this paper we first examine general techniques for managing information and how various evaluation criteria, when applied to the techniques, are affected by the parameters and scale of a specific problem. We then illustrate the use of the techniques and criteria for the problems of locating service and server status information, as well as determining the current status of a user in a distributed environment. In each case we find the best technique to use for the problem based on its specific parameters.

2 Information Management Techniques

In this section we describe techniques for managing information that originates on a per-machine basis. We denote each machine that supplies a portion of the total information as a *server* of the information, and each machine that makes

a request for the information as a *client* of the information. Machines may be both clients and servers.

Despite the many types of information location problems that exist in distributed systems and the seemingly large number of solutions to these problems, the solutions can be reduced to a few basic techniques. The basic techniques are derived from the observation that information can be stored in three places within the distributed environment:

1. Information can be stored at its origin and not replicated in the distributed environment. Thus, information for each server machine is stored only by that machine.
2. Information can be stored close to where it is accessed. Thus, information is collected from the servers and stored on each client machine.
3. Information can be stored by machines well-known to both clients and servers. Thus, information is collected by intermediate machines, known as *masters*, that either know all information or cooperate to share all information among themselves.

Three of the basic techniques for managing and accessing information are direct consequences of these methods for storing information. A fourth technique is available if the specific server machine that contains the information is known. In addition, variations of these techniques exist such as caching information or giving preference to information on the client machine. In the following we name, describe, and illustrate each of these techniques. We also discuss the appropriateness of each for detection and measurement of negative information.

2.1 Direct Access

Direct access is used when the location of the information is given or can be derived from the name of the information requested. For example, in the IBIS [TR84] and Newcastle Connection [BMR82] file systems, the location of the file is explicitly included as part of the file name. In Sun's NFS [SGK*85] and the Tilde file naming mechanism [Dro86], the location of the file is not explicitly given but

can be determined by consulting a local table. When direct access is used, each server maintains its own information and only corresponds with other machines when information is requested. Negative information such as the absence of a requested file is immediately available if the server machine is up, but negative information concerning if the machine is down can only be detected by using a timeout period in waiting for the response.

2.2 Multicast

In situations where the location of the information cannot be determined directly, one possible technique is to let each server store its own information and let clients multicast requests to the server machines for information as needed. Multicast allows a message to be sent to a number of machines using a single address [CD85]. It is typically supported by local area network hardware, but its support by network software is currently limited, with the V-System [CZ85] as one exception; broadcast or multiple messages are typically used in place of multicast.

We use the term multicast for this technique because information is retrieved using multicast from the server machines on demand. The multicast technique is useful if the rate at which information is accessed is slow in comparison to the rate at which the information changes. Two problems exist with this technique. First, all servers get information requests even though only a small proportion may respond to the request. Second, the amount of time the client should wait for responses to the request is unknown if the number of responses is not known *a priori*. In this case, the client is reduced to waiting "long enough."

This technique also makes detection of negative information difficult because the lack of response from a machine may mean it is down, the response was lost, the machine does not supply any requested information, or it did not respond soon enough. Consequently, this technique works best when a small, fixed

number of responses is expected. The Sun RPC *rstat* and *rusers* programs use this technique to collect information about machine and user status in a network [SMI86]. From experience, these programs take an exceedingly long time to timeout and require multiple broadcasts to guarantee that responses from all active machines are received.

2.3 Local Store

In contrast, if the client machines collect and store all information from the servers in the network then each client machine has a complete store of information. A server process on each client machine maintains the store of information available in the network and all requests for information can be satisfied on the client machine. To access information a request is sent to the server process on the client machine; when information changes on a server machine a multicast message must be sent to all client machines informing them of the change. The local store technique is useful when the rate of requests is much greater than the rate of change for the information, or when the time to access information must be minimal. It also requires that it should be feasible for each client machine to store the total amount of information.

Negative information is easy to detect and measure only if each client machine can expect periodic updates of information from the server machines. In this case the client is guaranteed to know of changes to the information and can assume the information is unavailable if it does not receive an update over a period of time. This technique is used by the *rwhod* protocol in UNIX to manage user and machine status information with the file system as the communication medium for requests [UNI86]. From experience in our computing environment, this protocol is not used by workstations in an environment consisting of many machines to avoid overloading the workstations with update packets.

2.4 Network Master

As opposed to storing all information at each client, another technique is to designate a well-known machine in the network as a master machine that knows all information. Whenever information changes the server machines send updates to the network master, and whenever information is requested the client machines send requests to the master machine. This approach is appropriate when the total amount of information is large, and the use of multicast or broadcast messages to all server machines is not satisfactory. An obvious problem with this approach is vulnerability to failure if one master server is used. To combat this problem, multiple masters can be used that duplicate or share the information between them. Requests and updates then can either be multicast to all masters or sent to one with masters propagating messages among themselves as necessary. As with the previous technique, the master servers must be able to expect periodic updates to supply negative information. This technique has been used in Grapevine [BLNS82] and the Domain Name System [Moc87] to manage information in large, distributed environments.

2.5 Variations

Many variations of these four basic techniques exist. Two common ones that we will discuss are the use of a local cache, and giving preference to information on the client machine.

Caching is a common technique for reducing the cost of accessing data in a distributed system [Ter87]. As opposed to the local store technique described above in which each client machine is informed of updates to the information, a cache is only used to retain information that was discovered by a previous request. Thus a cache can only be treated as a set of hints that is not necessarily complete nor accurate. However, if the cache miss rate is low, or the cost to recover from inaccurate hints is small then a cache is a valuable technique in conjunction with

one of the information location techniques. The Address Resolution Protocol (ARP) for mapping between machine internet addresses and physical hardware addresses maintains a local cache on each machine to significantly reduce the number of broadcasts that must be performed [Plu82]. Cheriton and Mann [CM86] have successfully combined caching along with multicasting in the study of a decentralized naming facility.

Another practice is to associate quality with information. Some systems give preference to information that is “good enough.” For example, Eager, Lazowska, and Zahorjan [ELZ86] propose the idea of a threshold for load sharing. If the load of the local machine is below the threshold then the local machine is used to invoke a computation, otherwise direct requests are made to one or more other machines to check their load. In Maitre d’, a load-balancing system, a threshold is also used, but if the load of the local host is not satisfactory then a server is selected using locally stored load information [Ber86]. Another example is outlined in a paper related to our work concerning services. Comer [Com85] proposes that each client machine use a local version of a service if it is available, otherwise broadcast a message to locate the service. This scheme is applicable when local services should be used whenever available.

3 Evaluation Criteria

Given these different techniques for managing and accessing information we need a means for evaluating which is the “best” for a particular problem. We use an analytical analysis to derive cost functions for various evaluation criteria based on the parameters and scale of a problem. This approach provides an objective basis for comparing different techniques for a specific problem over multiple evaluation criteria. In this section we concentrate on evaluation of the four basic techniques and then discuss how the variations we described affect the cost of these techniques.

The cost function for each criterion is expressed in terms of two types of values—problem and scale parameters. Problem parameters are dependent upon the problem being analyzed. They include the rate at which information is accessed and updated. Scale parameters are dependent on the size and composition of the distributed environment. They include the number of client and server machines in the distributed environment. By changing the scale while keeping the values of the parameters constant, techniques can be evaluated for environments of different scale.

For purposes of our research we choose to consider the following criteria for comparison of the different techniques: the number of messages sent and received per machine, the total number of messages sent on the network, the storage required at each machine, the response time to obtain information, and the accuracy of the obtained information. The importance of each criterion varies for each problem so we do not place any relative weights on the criteria.

In deriving formulas for each of the criteria and techniques, we make a number of assumptions about the distributed environment to simplify the analysis.

- All requests for information are initiated from a client process on a client machine. Hence all information requests cause at least one message to be sent and received on the client machine.
- All messages are reliably received. Although this assumption is not valid for an unreliable network, we assume the number of extra messages needed to correct for unreliability is low in comparison to the total number sent.
- All messages can be encapsulated in a single network packet so the number of messages and packets is the same. Again this assumption may not always be valid, but we assume the number of additional packets is small.
- A message sent to a multicast address requires only one network packet, but is delivered to all machines in the multicast group.
- Network master machines each contain a copy of all information and no messages need to be exchanged among the master machines.

Given these assumptions, we can derive cost function formulas for each of the techniques. Two good measures for the cost of a technique for a given problem

are the number of messages that are handled (sent and received) by each machine and the total number of messages sent on the network. Formulas for these costs are given in Tables 1 and 2.

Table 1: Rate of Packets Handled for Each Machine

Technique	per client	per server	per master
Direct Request	$2 \cdot A$	$2 \cdot A \cdot c / s$	—
Multicast	$A \cdot (1 + r) \quad 1 \leq r \leq s$	$2 \cdot A \cdot c$	—
Local Store	$4 \cdot A + U \cdot s$	U	—
Network Master	$2 \cdot A$	U	$2 \cdot A \cdot c + U \cdot s$

Table 2: Rate of Packets Handled on the Network

Technique	Network Packets
Direct Request	$2 \cdot A \cdot c$
Multicast	$A \cdot (1 + s) \cdot c$
Local Store	$U \cdot s$
Network Master	$A \cdot (1 + m) \cdot c + U \cdot s$

The parameter A is the access rate for information per client machine. U is the update rate for information per server machine. The scale parameters c , s , and m are the number of client, server, and master machines, respectively. The parameter r is the expected number of responses from servers when a multicast message is sent. Even though r messages are received by a client machine, we assume that all s server machines may respond to the request. For the network master technique a message is multicast to all master machines and the first response is used. Again we assume that all master machines will respond to the request, but the client machine will ignore all responses after the first. Directed requests are assumed to be equally distributed between the server machines. For the local store technique, each access generates two packets on the client machine, which are each handled by both the client and server process.

A criterion that is crucial for time critical problems is the response time that can be expected to retrieve information. Unfortunately, the response time is a difficult value to express because of variability in machine and network response time. We express the communication cost in terms of two parameters: E_{local} , the expected round trip delay to echo a packet from the client process to a server process on the same machine; and E_{remote} , the expected round trip delay to echo a packet to a server process on a remote machine. In addition to the expected round trip time and processing time, I , to lookup information, the actual response time depends on indirect factors such as network and machine load, and direct factors such as the time to load executable code for the server process into memory. We represent the additional time used because of these factors as V due to its variability. The response time for retrieving information for each of the techniques is given in Table 3. Remote requests require a timeout period τ in case no response is received. When a multicast message is used, as done with the multicast and network master techniques, the client must wait for the timeout period or until the correct number of responses is received. The notation $\max_k$ denotes the maximum response time of the first k responses.

Table 3: Response Time to Retrieve Information

Technique	per access
Direct Request	$\min(\tau, E_{\text{remote}} + I + V)$
Multicast	$\min(\tau, \max_r(E_{\text{remote}} + I + V))$
Local Store	$E_{\text{local}} + I + V$
Network Master	$\min(\tau, \max_1(E_{\text{remote}} + I + V))$

Two other criteria are important for evaluation, but are more difficult to express in analytic form. The first, storage requirements, depends on the size of each piece of information and the total amount of information. Each of these values is extremely dependent on the problem in question. For each technique, each server machine must store its own information and the local store and network

master techniques store all information at the clients, and masters respectively.

The second criterion is the timeliness of the information received. The “older” the information, the more likely that the information is out-of-date and inaccurate or the server machine that supplies the information is down. The direct request and multicast techniques respond with up-to-date information directly from its source, but the other two techniques depend on how closely updates generated by the server machine correspond to actual changes in the information.

The formulas in these tables express the costs of the four basic information management techniques. To incorporate variations on these techniques, such as a cache on the client machine, we assume that the probability of a “hit” in the cache is p . If the cost for an operation without the cache is C_{normal} and the cost for a cache lookup is C_{lookup} then the total cost using a cache is

$$C_{\text{cache}} = C_{\text{lookup}} + (1 - p) \cdot C_{\text{normal}}$$

For example, using caching with the multicast technique and assuming negligible cost lookup for the cache, the rate of packets handled by each client is

$$C_{\text{cache}} = (1 - p) \cdot A \cdot (1 + r)$$

The formula analytically states what is intuitively known; if the cost of maintaining and looking up information in the cache is low and the hit ratio is high then a cache can significantly reduce the overall costs. The costs for the cache are the amount of storage that must be used on the client machine and the potential staleness of the cached information that is used. If local information is given preference, a similar cost results except C_{local} (the cost for local information lookup) is substituted for C_{lookup} and p represents the probability of local information being used.

4 Application of the Techniques

In this section we shift from a discussion of techniques and evaluation criteria for the general class of problems to application of these techniques to three specific problems. The various techniques can be objectively evaluated for each problem once its parameters are known. The principal parameters for each of the problems is how often the information changes in comparison to how often the information is accessed. The values for these parameters for each problem were derived from data gathered at peak usage times in our department computing environment. All analysis was done for a local area network (Ethernet) environment with round trip times for a packet (e.g. UDP) on the order of 10 ms.

For each problem we present the basic parameters and discuss the applicability of the techniques from Section 2. We then apply the evaluation criteria to these techniques and explicitly show how the number of packets on the network and for a client and server machine vary with the scale. The criteria are used to select the most appropriate technique for each problem. Further details of how the data were gathered, and how the parameters were used in the evaluation analysis can be found in [Wil88].

4.1 Locating Service Information

Locating service information is a problem of determining which machines in the computing engine provide a particular service. Using our gathered data to determine the rate at which a user at a client machine would be expected to initiate services we obtained an expected value of 20 services initiated per hour, with a high (not necessarily maximum) value of 80. The expected rate of change (additions or deletions) of services was 0.1 updates/hour at each server machine. These numbers agree with our intuition that services are used much more often than they change.

Given the high access/update rate ratio and that we expect most “small”

services to be provided by the client machine, the techniques used for managing the information should keep the service information readily available for each client machine. In considering the management techniques we eliminate the direct access technique because the location of the service information is not known. The local store technique appears a good choice because it keeps all of the service information at the client machines for the quickest response.

In their basic form, neither the multicast nor the network master technique appear useful for managing information because each technique requires at least one message be sent to a server or master machine to get non-local service information. However, if we consider combining each of these techniques with caching so that most service information can be found on the client machine then each is a possible approach. A problem with caching service information is that new instances (from new or existing server machines) of cached services may become available, but because the cache is used these new instances are never considered. To avoid this problem we postulate hybrid techniques that cache services as they are used, but also cause servers to multicast updates to the clients. A client machine then only needs to store the updates for services that are in its cache. Information not in the cache is retrieved through a multicast message, either to all servers or just the masters depending on the technique.

Figure 1 shows how number of packets on the network and for a client and server machine vary for each of the evaluation techniques according to the scale. For the analysis we assumed 10% of the machines in the network were server machines and that 75% of client machines were active at any time. The analysis was done using a high cache hit ratio (0.98) which is likely to be achieved after a few hours of steady work.

As shown there is little difference in the cost of the three techniques in terms of number of packets if the cache hit ratio is high. The difference is whether all service information, or only the information that has been used, should be stored on the client machine at the cost of retrieving information as needed. The use of

Figure 1: Criteria for Service Information Management Techniques

a cache causes more packets to be generated, but more important, the expected response time increases and is more variable if the cache hit ratio is not high, which can be expected at startup time. Consequently for our service execution mechanism design we choose the local store technique because it provides the most consistent response time and the number of network packets is not dependent on the hit ratio or the number of client machines in the network. This technique requires the most storage for each client machine, but from experience with our prototype the storage requirements are feasible for even a relatively slow workstation (Sun-2) with relatively small amount of memory (4MB).

4.2 Locating Server Status Information

Having chosen the local store technique to locate service information used in service resolution and location, we needed to determine how to locate server status information for service selection. By observing the type and mix of computing used in the data we collected, we determined that most services are small and that only about 10% of services will be invoked on a machine other than the client. For these remotely invoked services, server status information, specifically

load information, is needed by each client to select from among multiple server machines that provide the services. From our data we have an expected value of 2 accesses per hour from each client machine for server load information with a high value of 10 per hour. To determine the rate of change for load information we assume that each server machine samples its load at periodic intervals, but the load is defined to change only if the difference between the current load and the last reported load differs by a "significant" amount. Letting this amount vary as 25% of the load itself with a minimum of 0.50 we determined an expected update rate for the load on a server machine as 12 updates per hour based on empirical evidence.

Any of the four basic techniques discussed in Section 2 could be used for obtaining server status information because the server machines that are under consideration for selection are known from the location step. The multicast, local store, and network master techniques allow for the load from all machines to be gathered and the machine with the best processing power/load ratio to be selected. However, in the case of the direct request technique, many servers may offer a service and a request would have to be sent to each to gather all information. Instead we modify the technique using a threshold as outlined by Eager, Lazowska, and Zahorjan [ELZ86]. The idea is to send probes (direct requests) to servers until a server with a load less than a pre-determined threshold is found (up to a fixed probe limit). This technique does not necessarily choose the best server, but seeks to minimize the cost of information retrieval in a large network.

Figure 2 shows the evaluation criteria that are the most dependent on the scale for this problem—the count of packets. The costs for gathering service information using the local store technique are included as a base cost, so the plots indicate the total cost for service location and selection.

In evaluating the techniques, their strengths and weaknesses for management of server status information are much more disparate than for the service infor-

Figure 2: Criteria for Server Status Information Management Techniques

mation problem. Because of its sensitivity to the size of the network and the number of requests, the multicast technique does not appear appropriate for this problem, especially if the rate of information requests increases. The local store technique is good because it is impervious to the request rate and provides quick response, but its dependence on the size of the network and the rate of updates is not attractive. Although our analysis for update rate is worst case, it is not clear that the load behavior of a time-shared machine (used for load data analysis) will be the same as a server machine in a distributed environment. The network master technique decreases the load on the client machines from receiving updates, but places the load on the master machines as well as requiring a network access and increased response time for each request. In addition, the local store and network master techniques provide information that is not the most up-to-date.

With these factors in mind, the best technique for the problem appears to be the direct request approach with a threshold. This technique holds down the number of messages sent and received by each machine while finding up-to-date information about possible server machines. The technique also detects possibly down machines when a reply is not received for a request. The chief

drawback to this approach is determining a suitable threshold. One idea is to use a hybrid technique where a couple of master machines monitor the load on the network machines, and then the client machines can occasionally gather this information and set the threshold at the median value. Another problem is that the retrieval of the status information takes additional time. This problem appears less serious because remote invocation of a service also takes more time so the additional information location time is negligible.

4.3 Locating User Status Information

Another problem in the same class, but not part of the service execution mechanism, is determining the status of a specific user in a distributed environment. A user may have login sessions established on his personal workstation and a few additional hosts. Determination of his status means finding out on which machines he is active, and if he is not active how long he has been idle or how long it has been since he logged out of a session. Tools exist to determine this information, but require the requestor of information to know the specific machine(s) to check or require the requestor to filter the desired information from information about all active users. For our problem we assume that only a user id, which is unique for all users, is known and the requestor would like to know the current status of that user without knowing the machine(s) he usually uses. Observing the use of similar tools we determine that the average access rate is small, about 0.25 accesses per hour with a high of one per hour, and the update rate is 60 per hour (for information accurate to the nearest minute).

In examining possible techniques, the local store and network master techniques can both be used by storing all information, either at client or master machines and then extracting specific user information as needed. The multicast technique appears desirable because it only requests information as it is needed. Another technique is motivated by the observation that users tend to maintain

sessions on just a few machines. It is a hybrid of the direct request approach and divides the problem of locating a user's status into two subproblems. If we refer to the machines on which he maintains a session as a user's "work" machines then the first sub-problem is to locate the set of work machines for a specific user. This information can be stored at a few network master machines and be periodically updated from each server machine at non-peak times because the information changes slowly. For client requests, the work machines for the requested user are obtained from the network masters, and then direct request queries are sent to these machines to obtain the status of that user.

Figure 3 shows the effect of scale on the number of packets for each of the techniques. As opposed to the previous problems, all machines, even workstations, are both clients and servers of information. Thus the number of packets for a single machine is the number it handles as a client and as a server.

Figure 3: Criteria for User Status Information Management Techniques

In evaluating the techniques, the local store and network master techniques cause the network and either the clients or masters to handle a large number of packets. Each new machine causes another update packet to be handled by

the network every minute. Reducing the rate of update packets (at reduced accuracy for the information) only slows the packet rate as the scale increases. In contrast, the multicast technique is much better suited to the problem and the average number of packets handled by each machine is small regardless of the scale. In addition, this technique obtains up-to-date information regarding the current status and idle time of the specific user. A problem with this technique is that if a user is logged out, the last logout time is not practical to obtain. Although each machine could send a "last logged out" reply, the client machine would be inundated with replies from every machine when the user never logs in to a majority of these machines. Even restricting replies to those machines on which a user has an account may result in a number of unwanted replies because of centralized password files that allow a user to login to multiple workstations when he only uses one. Another problem with the multicast technique is determining how long to wait. Because the number of replies is not known, the best that can be done is to wait a "long time" since receiving the last reply and then assume that all replies have been received. This approach may not work if one machine is slow.

The hybrid direct request approach appears to be the best technique because it solves the problems with the multicast approach without creating significantly more overhead. The expected response time is longer because a message first needs to be sent to a master machine to obtain the user's work machines before sending the multiple directed requests to these work machines. However, the advantage is that each server machine always sends a reply regardless whether the user is active, idle, or logged out. Therefore, the client machine knows exactly how many replies it is expecting and can deterministically stop waiting. The client machine can also detect when packets have been lost.

5 Conclusions

In this paper we have examined techniques for managing and locating information for a general class of distributed information problems. Rather than viewing each problem within the class in isolation, we were able to identify common properties for the problems. The class of problems is characterized by the inherent distributed nature of the information where each server machine supplies a portion of the total information.

Four basic techniques for managing and locating information for this class of problems were identified and cost formulas were derived for each technique based on a set of evaluation criteria. By determining the parameters and scale of a specific problem the evaluation criteria allow us to make objective comparisons of different techniques for the problem. It is important to realize that there is no one technique that is satisfactory for all problems, but rather we need to understand the parameters and scale of a problem before deciding which technique to employ. Even when parameter values are not known or have to be estimated, the evaluation criteria can be used to evaluate the techniques over a range of parameter and scale values.

We used our results to evaluate management techniques for two specific problems of our service execution mechanism—locating service and server status information. We also used the results to examine the related problem of obtaining user status information. Although each problem is in the same class, different parameters led to a different solution for each problem. We found that hybrid techniques such as the use of caching or reduction of an information location problem to smaller, simpler information location problems may be necessary to obtain a satisfactory solution.

References

- [AAA88] Mostafa H. Ammar, José M. Bernabéu Aubán, and Mustaque Ahamad. Using hint tables to locate resources in distributed systems. In *Proceedings IEEE Infocom '88*, March 1988.
- [Ber86] Brian Bershad. Load balancing with Maitre d'. *login*, 32–43, January/February 1986.
- [BLNS82] A. Birrell, R. Levin, R. Needham, and M. Schroeder. Grapevine: an exercise in distributed computing. *Communications of the ACM*, 25(4):260–274, April 1982.
- [BMR82] D. R. Brownbridge, L. F. Marshall, and B. Randell. The Newcastle connection or UNIXes of the world unite! *Software-Practice and Experience*, 12(12):1147–1162, December 1982.
- [CD85] David R. Cheriton and Stephen E. Deering. Host groups: a multicast extension for datagram internetworks. In *SIGCOMM '85 Symposium*, pages 172–179, ACM, September 1985.
- [CM86] David R. Cheriton and Timothy P. Mann. *A Decentralized Naming Facility*. Technical Report STAN-CS-86-1098, Department of Computer Science, Stanford University, February 1986. Revised version to appear in *ACM Transactions on Computer Systems*.
- [Com85] Douglas Comer. Automatic location and invocation of computational service. In *National Communications Forum*, pages 748–750, October 1985.
- [CZ85] David R. Cheriton and Willy Zwaenepoel. Distributed process groups in the V kernel. *ACM Transactions on Computer Systems*, 3(2):77–107, May 1985.
- [Dro86] Ralph E. Droms. *Naming of Files in Distributed Systems*. PhD thesis, Department of Computer Science, Purdue University, August 1986.
- [ELZ86] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering*, SE-12(5):662–675, May 1986.
- [Moc87] P. Mockapetris. Domain names—concepts and facilities. November 1987. RFC 1034.
- [MV85] Sape J. Mullender and Paul M. B. Vitányi. Distributed match-making for processes in computer networks. In *Proceedings of the 4th ACM*

Principles of Distributed Computing, pages 261–271, ACM, August 1985.

- [Plu82] David C. Plummer. An Ethernet Address Resolution Protocol. November 1982. RFC 826.
- [SGK*85] R. Sandberg, D. Goldberg, S. Kleiman, Dan Walsh, and Bob Lyon. Design and implementation of the Sun Network File System. In *Proceedings of the Summer USENIX Conference*, pages 119–130, USENIX Association, June 1985.
- [SMI86] *Remote Procedure Call Programming Guide*. Sun Microsystems, Inc., February 1986.
- [Ter87] Douglas B. Terry. Caching hints in distributed systems. *IEEE Transactions on Software Engineering*, SE-13(1):48–54, January 1987.
- [TR84] Walter F. Tichy and Zuwang Ruan. Towards a distributed file system. In *Proceedings of the Summer USENIX Conference*, pages 87–97, USENIX Association, June 1984.
- [UNI86] *UNIX Programmer's Manual, 4.3 Berkeley Software Distribution, Virtual VAX-11 Version*. Computer Science Division, University of California, Berkeley, April 1986.
- [Wil88] Craig E. Wills. *Service Execution in a Distributed Environment*. PhD thesis, Department of Computer Science, Purdue University, May 1988.