Designing Directories in Distributed Systems: A Systematic Framework

K. Mani Chandy and Eve M. Schooler
Computer Science Department, 256-80
California Institute of Technology
Pasadena, California 91125
{*mani, schooler*}*@cs.caltech.edu*

Abstract

This paper proposes a framework for the systematic design of directory-based distributed applications. We evaluate a space of directory designs using our framework. We present a case study consisting of design, implementation and analysis of directories for a multicast application.

Our framework is based on a model that extends the formal concept of process knowledge *in distributed systems. This concept is used informally in phrases such as "process p knows when it is in state s that process q is active." We show that this definition of knowledge is too strong for many distributed applications, including directory design. We propose a weaker concept:* estimation. *We define the meaning of phrases of the form: "process p in state s estimates with probability 0.9 that process q is active." We specify directory design as an optimization problem with the objective function of maximizing estimation probabilities, and with constraints on the amount of bandwidth, computation and storage used. We show how this specification helps in a systematic analysis of alternative directory designs.*

Key words: Distributed systems, directories, design frameworks, theory of process knowledge, estimation, performance models, multicast.

1. Introduction

This paper presents a framework for designing directories in distributed systems, and describes a case study of a design, implementation and analysis of a directory for a multicast application.

For our purposes, a directory is a process in a distributed system that contains information about other processes. For instance, a directory process p may contain information that another process q is deadlocked. The problem that we are concerned with is as follows: For a given application, what sort of information should be stored in directories and how should this information be organized and maintained?

The outline for this section is as follows. The point of departure for our framework is the concept of *process knowledge* in distributed systems. We review the definition and theorems about process knowledge, and then show why this concept is inappropriate for directory design. We then show how global clocks and time impact knowledge, and analyze systems with bounded message delays. Next, we suggest a weaker concept: *estimation*. Estimation theory is a well-established branch of communications engineering; our contribution is to use it to formulate a framework for directory design.

1.1. Knowledge

A Distributed System. A distributed system is a set of processes and a set of communication channels. Different processes do not share state directly. Processes can interact with each other only by sending and receiving messages. The state of a distributed system is given by the sets of states of its component processes and channels. A distributed system has no global clock.

The Concept of Process Knowledge. Informal discussions about distributed systems may use phrases of the form: "process p in state s *knows* that process q is active." This means that process q is active in all reachable (global) states of the system in which process p is in state s [1, 4]. A global state is a state of the entire system, as opposed to a process' local state.

In general, "p knows x" is a predicate on p's local state, where p is a process and x is a predicate on global states of the system. The predicate "p knows x" holds for a local state s of process p if and only if x holds for all global states in which p's component is s. From the definition, if p knows x at a point in a computation then x holds at that point. For instance, if p knows q is active at some point in a computation, then q is indeed active at that point. In more philosophical terms, this statement says that a process only knows truths; a process cannot know falsehood. Early philosophical works on knowledge defined the concept in terms of a set of axioms [5].

Knowledge Gain and Loss. Theorems [1] about knowledge gain and loss that are relevant to our discussion about directories are given next.

Knowledge Gain Theorem: A process can only gain knowledge about another process by receiving messages.

Knowledge Loss Theorem: A process can only lose knowledge about another process by sending messages. If a process p knows at some point in a computation that a predicate x holds for another process q (for instance if p knows that q is active), then x continues to hold for q at least until q receives a message from p (possibly via intermediate processes).

The informal argument for the last result is as follows. If p knows that x holds for q at some point in a computation, then x must hold for q at least until p changes its state. Furthermore x can be falsified only when q knows that p has changed its state. The only way for q to know that p has changed its state is for q to receive a message sent to it by p (or for q to receive the last message in a chain of messages initiated by p and sent via intermediate processes to q). Therefore x must continue to hold until q receives a message from p, possibly via intermediate processes.

Knowledge is too Strong a Concept for Directories. The traditional definition of knowledge is too strong for many applications as illustrated in the next example. A directory may contain information that a process p is participating in a collaborative group application, such as a videoconference. The knowledge-loss theorem tells us that if the directory *knows* that p is participating, then p cannot cease to participate without receiving a message from the directory. In most practical situations, process p can cease to participate in the videoconference on its own; it does not need to receive messages (directly or indirectly) from the directory. Therefore, though a directory may have some information about the participation of a process p, it does not *know* whether p is participating or not.

Volition. If a process p knows that some predicate x holds for process q's state, then process q can only falsify x after it receives a message; q cannot falsify x autonomously. The fact that q cannot falsify x autonomously is called q's giving up *volition*. In general, if p knows x about q's state then q gives up volition to change x. In many practical situations, a directory has information x about a process q but the process does not give up volition to change x; therefore, the directory does not *know* x. The question is: What does "the directory has information x about q" mean in this case? For instance, what is the meaning of "the directory has information that a remote process q is participating in a videoconference?"

1.2. The Role of Time

Global Clocks. The knowledge gain and loss theorems are based on a model of distributed systems that assumes that different processes share no state, and that there is no global clock. If there were a global clock then a process could gain or lose knowledge merely by the passage of time. For instance, if p knows at time 8 that q will participate in a videoconference between time 10 and time 11, then at time 10 p knows that q is participating in the videoconference, and at time 11 p does not know if q is still participating. Therefore, p gains knowledge about q merely by the passage of time *without additional messages from p to q*, and likewise p loses knowledge about q by the passage of time.

Note that in this situation as well, q gives up volition. If p knows at time 8 that q will participate between times 10 and 11 then q cannot choose autonomously to not participate in this interval. Since processes do not give up volition in most directory-based applications, a directory does not generally *know* predicates about the states of other processes, even in systems with global clocks.

Bounded Message Delays. The problem with knowledge and volition is not an artifact of unbounded message delays. Assume that message delays from q to p are between L and U in duration, and that L and U are known constants. Assume that messages are not lost; all messages sent will be delivered with delay at most U.

Consider a protocol in which if p does *not* receive a message from q in the time interval $[T, T']$, p knows that q is active at T'. This protocol requires q to send a message to p in the interval $[T - L, T' - U]$ if q is not going to be active at T'. A message sent by q before $T - L$ may arrive at p before T, and therefore to guarantee that the message arrives in the interval $[T, T']$ it must be sent after $T - L$. Likewise, the message must be sent before $T' - U$ because later messages may arrive only after T'. If such a message is not sent, then at time $T' - U$, q gives up volition to become idle at T'. Of course, there is no problem if $U = 0$ which means all messages are delivered instantaneously, but this situation does not arise in distributed systems.

1.3. Estimation

We propose a concept, estimation, that is weaker than knowledge, and that we believe is the appropriate structure for dealing with information in directories. We postulate that our distributed system is a stochastic process in steady state. We deal with the probability that a predicate x holds, or the conditional probability that x holds given y holds. For instance, we will compute the probability that a process q is active given that a directory process p received a message T units ago from q stating that q was active when the message was sent. Although we assume a global clock, the ideas can be extended to distributed clocks with clock drift; however, we do not do so in this paper.

Let us review the concept of process knowledge. In essence, knowledge deals with invariant implications of the form $[X \Rightarrow Y]$, where X is a predicate that deals with the states of a set of processes, D, and Y is a predicate that may deal with any other set of processes. For instance X could be a predicate on the state of a process p and Y could be a predicate on a different process q; by inspecting the state of p alone, we can determine whether X holds, and if X holds we can conclude (from $[X \Rightarrow Y]$) that Y holds for process q. Basically, information about one process gives us information about other processes. In terms of directories, data stored in a directory process gives us information about the states of remote processes.

To deal with volition, we extend *implication* denoted by $\Rightarrow$ to *estimates with confidence c* denoted by $\rightarrow_c$, as follows:

$$(X \rightarrow_c Y) \equiv (\mathit{probability}(Y|X) \geq c)$$

For instance, X can be the predicate that a directory process p contains data that a process q is active, and Y can be the predicate that q is active, in which case $(X \rightarrow_{0.99} Y)$ means that the conditional probability that q is active, given the information in the directory, exceeds 0.99.

For finite-state systems in which all states have positive probability measure,

$$[X \Rightarrow Y] \equiv (X \rightarrow_1 Y)$$

Computations with conditional probability are much more difficult than computations with implication; however, conditional probability is the concept of choice when dealing with directories. For instance if Y is the predicate "process q is active," and X is: "a directory process p has information that q is active," then q can transit autonomously from active to idle, provided $\mathit{probability}(Y|X) < 1$.

A useful special case of estimation is to extend the concept of the predicate *p knows Y* to the predicate *p estimates Y with confidence c*, which means that the conditional probability that Y holds exceeds c. We can now ask ourselves questions such as what is the meaning of: "p estimates (p' estimates Y with confidence c') with confidence c? This is analogous to the meaning of "p knows q knows Y. We do not carry out such exercises further because our goal here is to develop a practical framework for designing directories.

2. Specification of a Directory

In this section, we present a specification for a directory. Of course, the specification depends on the application that uses the directory; different directory-based applications have different specifications. The specifications of most directory-based applications have certain common characteristics, and we discuss those first. Then we consider the issues specific to a given application.

The general specification of a directory-based application is as an optimization problem. The objective function is to maximize the estimation probabilities that a directory has about the states of remote processes. The main constraints are the resources — computing, memory, bandwidth — used by *protocols* that maintain the directory. Other constraints deal with properties, such as fault-tolerance, of the given system.[1]

Though the general goal is to maximize estimation probabilities, the specifics of a problem depend on the use of the estimates. In particular, different applications have different consequences of error. For instance, if an application makes a decision based on the assumption that a remote process is active when the process is not active, the consequences of error depend on the

[1] In all optimization problems, the parameter to be optimized and the parameters to be constrained are a matter of choice; so we could formulate the problem in terms of constraints on estimation probabilities with the goal of minimizing the use of some resource such as bandwidth.

application. If the consequence of errors is small, it is less important to obtain high estimation probabilities.

The specification has the form:

Devise a protocol for a directory in a distributed system that estimates the states of remote information (remote processes or sites). The directory makes decisions based on these estimates that maximize the value of the decisions (or minimize the consequences of incorrect assumptions about the states of remote sites).

Subject to: constraints on storage, computational cycles, and message traffic.

Given: infrastructure properties, spatial and temporal locality, and the use of directory information, all of which we consider central issues in the specification and discuss informally next.

2.1. Central Issues

Infrastructure Properties. System infrastructure includes the communication network, storage system, and compute engine. Example communication properties are the distribution of delays or the occurrence of faults. Rapid or unpredictable state changes would make the estimation problem more difficult because a directory can obtain high confidence of estimation only by receiving timely information. A directory p can maintain high confidence of estimation about another process q even if the directory receives information infrequently from q provided q's behavior is predictable and the system has a global clock. For instance, if an aircraft follows its flight plan closely, then the air traffic control system can estimate the aircraft's location with high confidence at any point in time.

Spatial and Temporal Locality. Locality of information plays a role in the cost of a directory protocol and describes the manner of state change of component processes. It encompasses spatial locality, such as the topological dissemination of state changes to all or some directories, as well as temporal locality, such as the rate of state changes. In some directory applications, for instance certain directories of network routing information, a process needs to estimate aspects of the states of nearby processes (in this case the reachability and addresses of nearby routers) but does not care about processes far away; this allows for localized information flow.

Use of Directory Information. Who needs what information when, and for what purpose? If the consequence of erroneous decisions is severe, then higher confidence of estimation is required. One of the questions that we explore in our project is the increasing amount of resources required to obtain increasingly higher confidence.

Next, we present a few examples of directory-based applications to motivate our two-tiered approach of (i) a general specification common to most directory-based applications and (ii) issues that tailor the general specification to a particular application.

2.2. Examples of Directory-Based Applications

Metacomputing. A metacomputer is a collection of computing and communication devices that are integrated to offer an interface to a single resource collection. For instance, a large scientific visualization might marshall many resources across the network to complete not only the calculation, but also the display and storage of the results; this is in contrast to an approach that tries to tailor the problem to the abilities of a single (often parallel) machine. An important part of metacomputing integration are the directories that keep track of resources at remote sites. This information has some components that change slowly and others that change rapidly. Examples of slowly-changing information are the amounts of resources (computing, memory, input-output devices) located at each site. Examples of rapidly-changing information are the amounts of resources currently free at each site. A decision based on an erroneous assumption that a resource is free at a remote site (when the resource is in fact not free) has the consequence that the job schedule has to be changed. Conversely, a decision based on an erroneous assumption that a resource is not free at a remote site (when the resource is in fact free) has the consequence that jobs are delayed unnecessarily.

Videoconferencing. Informally speaking, a multicast videoconference is like a collection of people talking to each other, and the *scope* of the conference is analogous to how loudly the conversation is carried out: the louder a person speaks, the more people who can hear the speaker. A person should speak just loud enough so that all those who want to hear the speaker can do so. If the scope is too large, the multicast uses bandwidth on some links unnecessarily. If the scope is too small, some users that should participate in the videoconference cannot do so.

Directories contain information about which users are participating in the videoconference, and their respective scopes. This information is dynamic because new participants may arrive, and some participants may leave. The problem is to adjust the scope dynamically so that it has the minimum value to reach all those who wish to participate, as shown in Figure 1.

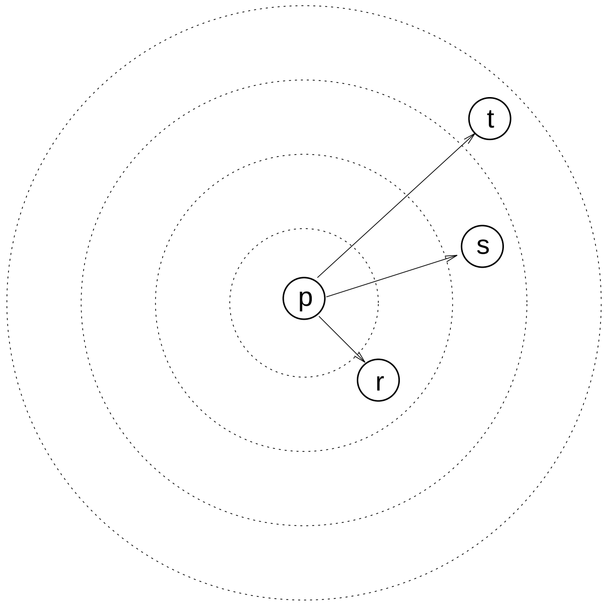

Figure 1. Session Scope: the maximum of the minimum scope needed by process p to reach participant processes r, s and t.

Telephone and IP Address Directories. The consequence of incorrect information in a telephone directory is either that a phone call is made to a wrong number or that a customer cannot make a phone call because the intended recipient is not in the directory. This situation is handled by a multilevel directory, which is comprised of a hard-copy directory that changes once a year and a more current directory maintained by the telephone companies.

In the Internet realm, the Domain Name Service (DNS) provides a mapping between machine names and machine addresses [6]. Again, the directory is multilevel, but the DNS directories change dynamically. Each entry is renewed regularly by periodic announcement messages sent by the process whose name and address is being stored, Without renewal, the entry is dropped from the directory, and then this change is propagated up the the directory hierarchy. As a result, the consequence of incorrect information in the DNS is that an incorrect mapping is provided or that the translation from name to address cannot be made because the entry does not exist; both of these scenarios cause an error in the requesting application. A third outcome is that there is delay as the mapping is retrieved from further up the directory hierarchy.

Other Examples. Information maintained by credit card companies about current usage of a credit card account is interesting in the context of knowledge, volition and the estimation framework. If the specification is that the directory maintained by the company *knows* the current expenditures in an account, then expenditures on that account can be carried out only after receiving permission from the directory. (In some cases, expenditures on large-ticket items require explicit permission while expenditures on small-ticket items are done autonomously by the seller of the item, in which case the directory only has an estimate — but not knowledge — about the state of the account.)

Directories to keep track of cellular phones and other mobile communication devices have interesting costs of error, depending on the nature of the call or the urgency of the communication. Directories in air-traffic control and other command and control systems have high consequences of error and therefore estimation probabilities are required to be high. In some cases, estimation probabilities are required to be so high that a good approximation is to design protocols in which directories know the states of remote sites; for instance, as a first approximation pilots give up volition to change flight plans without receiving permission from the air-traffic control system.

3. Design Space

In this section we identify two design issues common to most directory designs.

1. *Should the directory be organized in a centralized, hierarchical, or completely distributed manner?* In a centralized scheme, there is a single repository of information. In a completely distributed scheme, each process maintains its own directory and makes decisions based on its own estimates of remote sites. A hierarchical scheme partitions the directory information into a tree-like structure, and processes traverse the hierarchy to locate information about remote processes.

   Hierarchical and distributed schemes use a variety of caching strategies (or use no caching strategy at all). Typically these schemes are a function of the frequency with which a piece of information (process state) is either retrieved or updated.

2. *How and when does a process inform a directory about state changes?* A process can send informa-

tion periodically, with every state change or significant state change, or only when the information is requested by a directory.

To answer these design questions, we explore the parameters that effect these decisions.

3.1. Directory Organization

Centralization. A centralized directory architecture is suited to the situation where processes reporting state to the directory are within close proximity of the directory itself. Centralization also makes sense when the ratio between reporting processes to computational overhead at the directory is small, or the ratio between reporting processes to network overhead is small. If directory information is relatively static or warrants few accesses, a centralized scheme also is appropriate.

The main worry is that the directory is potentially a bottleneck and a single point of failure. As a result, scaling up becomes less viable, due to the sheer number of processes or the the geographic distribution of processes.

Hierarchy. When state information can be partitioned into manageable size components that reflect directory access patterns, then hierarchy should be introduced into the directory organization. A hierarchical scheme also should be used if information can be arranged so that estimates most frequently needed by a process are available in directories near the process, and estimates needed less frequently can be obtained by searching the hierarchical structure. In addition, the hierarchical scheme should be used if information can be organized so that directories higher up in the hierarchy have better estimates than those lower down. A process obtains information from a nearby directory, and if that information is incorrect it requests information further up the hierarchy chain, as in the telephone and IP address directories.

Distributed Directories. Distributed directories replicate information about process state in multiple places in the network. When is it appropriate to distribute the directory in this fashion? A distributed directory is an optimal choice if it is inexpensive to replicate data among multiple processes and if synchronization for consistency among the directories is either unnecessary or trivially enforced (due to the number of participating entities).

There are gradations of directory distribution. One extreme is to fully distribute the directory, where every process participates in the directory protocol. The key advantage to this approach is that the directory processes no longer obtain knowledge indirectly. Each directory gains remote process knowledge from the remote process itself, rather than through a proxy acting on behalf of the remote process. Consequently, the estimation probability will be higher.

Nonetheless, this approach exhibits some of the same shortcomings as a centralized architecture. It has the problem that storing all process state together in one place consumes significant memory or disk resources. In addition, because all processes are directory processes, message generation may present challenges, as all processes send state information to all other processes. Another scaling issue emerges if the frequency of message exchanges is high. This is problematic if the bandwidth used by the processes in the aggregate exceeds the link capacity or some predesignated upper threshold.

Therefore, a fully distributed directory architecture is a useful approach when there are limited directory processes, when it is inexpensive for this set of processes to fully communicate (perform multiway communication), and when storage and computational overhead of keeping full but approximate global state is low. For applications that permit inconsistencies among directory contents [7], or those that only need eventual consistency [3], the fully replicated directory provides the greatest immediacy of state information, as there are no intermediaries required.

More moderate forms of this architecture use coarser grain replication, providing just a few replicated directories. This approach is appropriate when a fairly small number of replicated directories sufficiently can manage the remote process state and accesses to it. Thus the disadvantages are eliminated. For example, there has been an increased amount of discussion about Web caching. The notion is that there should be mirror sites for heavily-accessed Web servers (directories) to distribute the load on the Web file server, as well as the network. Fist, where should the caches reside? Second, how many should there be? The most promising approaches have suggested that these directories migrate over time and/or replicate under heavy usage, in those instances where the server becomes inundated and must shed some of the load. When there are only a few directories, it is also possible to employ cache consistency without prohibitive penalties.

3.2. Directory Maintenance

There are two approaches for processes to share state information with directory processes: *passive* and *active* messaging. Passive messaging is a technique where processes distribute state information to directory processes, whereas active messaging is where processes send state information in response to the directory process or processes querying for that information.

Periodic Updates. One passive approach distributes state information on a regular basis. This is most appropriate if state information may change at a rate that matches the periodicity of the announcement interval. It is also useful if the information may not change, but needs renewal within that timeframe, if the application using the directory relies on having timely state information, i.e., higher estimation probabilities, Or the message delivery system is unreliable and leads to some degree of misinformation, i.e., if a renewal message is lost.

The update periodicity can be explicitly declared by the announcing processes, by embedding the value of the interval duration in the message itself, or by agreeing on a predetermined update interval. Alternatively, the periodicity can be inferred by the directory due to the regularity of message receipt, if in fact the announcement interval is highly regular. Yet another strategy is to make the periodicity adaptive, so that network bandwidth is conserved and processing overhead is reduced.

The choice in periodicity strategies is driven by how much the processes and the network can afford to carry additional timing information in the message, by how much processing is needed to adapt the periodicity to the load characteristics of the network, and how much link bandwidth is available.

To focus on the latter point, often network bandwidth is greater in the local area than in the wide area, and thus the frequency of state information updates can be more rapid within the local area than in the wide area. Yet, it may be the case that the processes that care most about timely state information are geographically further away, in which case the context of the directory plays as important a role as bandwidth constraints. One caveat is that the further away the processes from the directory where state is stored, then the more the network is impacted by state updates.

Furthermore, rapid state information dissemination does not necessarily improve knowledge estimation. If the periodicity of message announcements is too high, then the network may be inundated and more messages may be dropped, leading to worse approximations of global state.

Significant State Changes. Another optimization is to have processes only report significant state changes. If process state changes infrequently, then this approach produces useful savings. However, this depends on a reliable messaging infrastructure, which may or may not be provided by the underlying network fabric, and may have to be built in.

Active Requests. In situations where the rate of state changes is extremely low, an active messaging approach should be employed. In passive messaging approaches, the interactions between the directory and the announcing processes were considered critical to protocol design. With active messaging, the interactions between the directory and the requesting processes (the processes relying on the information in the directory) have more impact on the protocol outcome. Here, there are more query messages generated by requesting processes, than state changes produced by the processes reporting to the directory. As a result, the placement of the directory may be closer (in network delay terms) to the community of requesting processes than to the community of reporting processes.

4. Mathematical Issues

In this section we give a few examples that illustrate the basic mathematical issues. It should be no surprise that Bayes' rule is the fundamental rule in this analysis since the concept of estimation is based on conditional probability.

Our goal in this paper is to give a *qualitative* idea of how estimation probabilities are used in our design framework. We do so by considering a few simple examples. Lack of space prohibits a more detailed discussion of our framework and its associated mathematics.

4.1. A Simple Example: A Process with Two States

Problem Specification. Consider an example consisting of a directory p and a process q where q has two states, active and idle. We associate the boolean $q.a$ with process q, where $q.a$ holds *true* if and only if q is in the active state, and $q.a$ is *false* if and only if q is in the idle state. Associated with directory p is a real number $p.c$ which is the conditional probability that q is active given the information in p. Our problem is to explore different kinds of information stored in the directory and different ways for q to communicate information to the directory so as to maximize $p.c$ subject to computational, storage and communication constraints.

Assumptions. Assume that the duration of time that q remains continuously in the active state, before it next transits to idle, is a random variable X, and let the time that q remains in the idle state before it next transits to active be a random variable Y. All active (and idle) durations are independently and identically distributed. Assume that the average active and idle durations are much greater than message delays. This assumption is justified in applications such as those where states represent the availability or unavailability of a user to participate in a videoconference; in this case, the average time between state transitions is on the order of several minutes whereas message delays are in seconds.

Let us assume that messages from q to the directory p are delayed by a constant U time units. Later, we relax the assumption that all messages are delivered. The constant-delay assumption helps in giving an intuitive understanding to the issues involved.

We assume that the distributed system has a global clock.

A Simple Protocol. One of the simplest solutions is for q to send a message containing the value of $q.a$ to the directory p each time q changes state. The directory keeps the contents of the last message and the time that the message was sent by q. Let $p[q].m$ be the contents of the last message received by p from q, and let $p[q].t$ be the time at which the message was sent. Let T be the current time. Given $p[q].m$, $p[q].t$ and T, we wish to estimate the conditional probability that q is active, as shown in Figure 2.

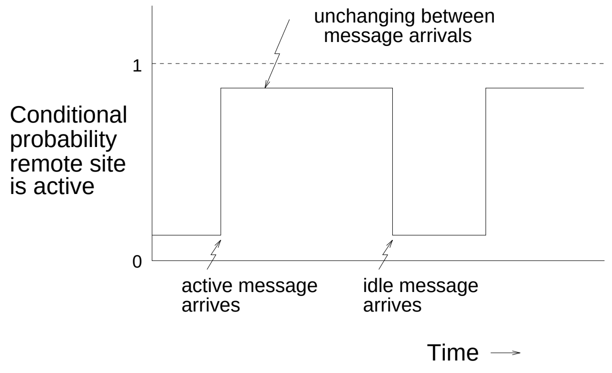

Figure 2. No Message Loss: exponential holding times.

Analysis. Let $p[q].m = true$, i.e., the last message p received from q was sent by q when q made a transition from idle to active. Let the value of $p[q].t$ be t; therefore, the time at which this message was sent by q was t. Recall that T is the current time. Since the message delay is a constant U, no message was sent by q in the interval $[t, T-U]$. Therefore, q remained active in that interval. If q is now idle, then it made the transition from active to idle in the interval $[T-U, T]$. Therefore, the conditional probability that q is idle at time T is the probability that q transits from active to idle in the interval $[T-U, T]$ given that q transits to active at time t and remains active in the interval $[t, T-U]$. Recall that we make the assumption that U is small compared with the average values of X and Y; so the probability of more than one state transition while a message is in transit can be ignored.

$$\begin{aligned} &prob(q \text{ idle at } T \mid p[q].m,\ p[q].t = true,\ t) \\ &\qquad = prob(T-t-U \le X \le T-t \mid T-t-U \le X) \end{aligned}$$

A similar analysis can be carried out for the case where $p[q].m = false$.

If X is an exponential random variable, then this conditional probability is independent of $T-t$ and increases as $(1 - e^{U/d})$ where d is the average duration of an active interval. For small values of U/d this conditional probability increases linearly (as U/d). The mathematical and qualitative analysis are similar if X has other distributions where X is bounded above by a constant U and where U/d is small. This approach is reasonable if $U/d < 0.05$ and estimates with confidence 90% are adequate, which is often the case.

4.2. Faulty Communication Channels

Our goal here is to gain a qualitative understanding of the effect of message loss. We analyze a case identical to that considered in the previous section except that messages may be lost. The protocol is the same as in the previous paragraph: process q sends a message to the directory p each time q changes its state.

Consider the case where a message sent by q to the directory is delivered with probability g and is lost with probability $1-g$. If the message is delivered, its delay is a constant U. Let us analyze the situation considered in the previous section in which the directory receives a message sent at time t by q, stating that q made a transition from idle to active at time t, and the directory has received no subsequent message. What is the confidence with which the directory estimates that q is active now at time T?

Analysis. The probability that process q is idle at T, given that it became active at time t, and given that the directory has received no message after the one sent at t is the sum of the probabilities that

1. q became idle some time in the interval $[t, T]$, and the message that q sent when it made the transition to idle got lost. This value is $g \times prob(X \leq T - t)$.

2. q became idle in the interval $[T - U, T]$ and the message sent will be delivered after delay U. This value is $(1-g) \times prob(T-t-U \leq X \leq T-t \mid T-t-U \leq X)$.

Figure 3 shows the shape of the graph of conditional probability that q is active at T given information that the directory receives.

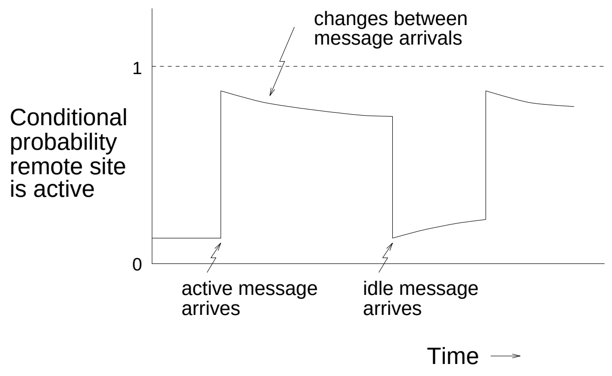

Figure 3. With Message Loss: exponential holding times.

For large T and g the estimation probabilities are low, so the protocol makes decisions based on erroneous estimates a significant fraction of the time. This is unsatisfactory in most cases.

An Alternative Protocol. An alternative protocol that solves this problem is for q to send messages periodically, where the information contained in the messages indicates q's state. Estimation probabilities get better as message frequency increases, but traffic required by the protocol increases as well. An analysis of this protocol for practical situations requires more space than we have here. Several interesting questions arise such as: Should the period at which q announces that it is active be the same as the period that it announces that it is idle? Should the periods change during the course of an execution (for instance to limit bandwidth usage by the directory protocol)?

These and other issues are handled systematically in our project. We examine in the next section the architectural tradeoffs that lead to different points in the design space.

5. A Case Study

In this section, we describe a case study that explores a particular point in the directory design space, and that illustrates how architectural decisions are driven by application context.

5.1. Design

We have designed a directory that dynamically updates user location information across wide-area networks [7]. It is a fully distributed architecture, meaning every process participates in the directory protocol. The motivation behind this experiment was to help collaborative applications find user addresses for those users it wishes to invite to participate in multimedia sessions. Such an application has a real need for up-to-date mappings between a user and the user's physical locations on the network. This is an increasingly difficult problem to solve because of several reasons; the number of users accessible via the network continues to grow exponentially, often each user is reachable in multiple locations at different times, and each user frequently is associated wth multiple aliases.

5.2. Implementation

Each directory process uses periodic *multicast* messaging [2] to disseminate and collect user state information. State information includes naming and addressing information for a user. As a result, a directory is able to track the most recent location where a user resides, as well as statistics on previous and alternate locations. Additionally, scope information is included in messages and thus stored in directories, so that network distances between users (i.e., between their directory processes) can be derived.

In turn, scope information is used by collaborative applications, such as teleconferencing, to ascertain the correct group scopes for real-time sessions, given a particular collection of users. An estimate of the appropriate group scope is obtained from estimates of pairwise scopes needed between all users in the group.

5.3. Analysis

How critical are the directory estimates for user location?

False Negatives. What if a user's state information is not found in the directory when the user is actually active in the network (i.e., participating in the directory protocol)? The ramifications of not storing information about a user is that that user cannot be invited into a conference. However, the only reasons why an active user's information might be missing from a remote user's directory is that the location announcement messages are not reaching the remote directory process(es). This would occur if there is congestion in the network, or the network is partitioned. In both these cases, it would do no good to try to rendezvous with the omitted user, since the subsequent conference could not be sustained by the underlying network. Therefore, the active user's omission from a remote directory is relatively harmless, and is actually a more accurate assessment of the user's reachability. Because the messages are periodic, when the network repairs itself, the announcement messages once again will flow to remote directories.

False Positives. What are the consequences of a user's information not being removed from the directory when the user becomes idle in the network? Typically, when the user is no longer active, it no longer multicasts location messages to peer user directory processes. User information is removed from the directory if it is not renewed within some timeout period. If the timeout interval is too short compared to the announcement interval, then the status of user activity risks being incorrect. If the timeout interval is too long, then a user may be reported as active, when in fact the user is no longer reachable. However, the collaboration software tries quite hard to reach a user to forward an invitation request. Consequently, it sends invitations to both past (idle) and present (active) addresses for a particular user (sequentially or in parallel); either the user resides at one of the locations, or at none of them. Therefore, the consequences of an incorrect estimate in the local directory is the overhead of storing the entry and the overhead of the local user sending an invitation request (or requests). This translates into either a minor delay at conference initialization if messages are sent sequentially, or false hope that a conference can meet when it cannot; both are fairly minor errors to endure.

Other State Inaccuracies. What if the address for a user is correct, but the scope is not? If the scope is too large, the real-time data flows potentially are sent to disinterested parts of the network, using up bandwidth. If the scope is too small, some session participants will be unreachable. This may cause a delay in full session connectivity as the scope value is incrementally adjusted to reflect the real group scope.

5.4. Future Work

In the future, we will evaluate the applicability of other fully distributed and hybrid architectures (a combination of distributed and hierarchical approaches). Our interest stems from two important developments. First, the emergence of IP multicast [2] allows a reduction in the overhead associated with group communication, which makes efficient multiway messaging tractable in the wide area. Second, the use of adaptive *soft-state* protocols (ones that rely on approximate rather than guaranteed agreement of global state) permits such an architecture to scale more effectively, provided the directory can withstand, and even take advantage of, temporary or permanent inconsistencies in global state of directories.

6. Summary

In this paper, we presented a new framework for thinking about directories and expanded the scope of what is defined as a directory. We showed that process knowledge is too strong a concept for distributed directories and suggested the weaker but more appropriate concept of estimation. We introduced a specification for distributed directories, with an objective function to maximize estimation probabilities, while minimizing the consequences of incorrect assumptions. Through examination of infrastructure properties, information locality, and specific examples of directory usage, we identified the key issues in the directory design space. We discussed informal guidelines to help a designer locate an appropriate architecture within this space, given certain constraints; our ongoing work is to refine and to formalize this methodology. An overview was given of the mathematical foundations for directory performance analysis, which provided a qualitative glimpse of how directory estimation might be derived and evaluated. Finally, we described a case study of a fully distributed directory service for collaborative applications needing user location in the Internet.

7. Acknowledgements

This research was supported in part by National Science Foundation grants CCR-912008 and CCR-9527130. This work is part of the Caltech Infospheres Project, which is described more completely at `http://www.cs.caltech.edu/~adam/CALTECH/infospheres.html`

References

[1] K. M. Chandy and J. Misra. How processes learn. *Journal of Distributed Computing*, 1:40–52, 1986.

[2] S. Deering. Host extensions for ip multicasting. RFC 1054, Stanford University, Stanford, CA, 1988.

[3] S. Floyd, V. Jacobson, S. McCanne, L. Zhang, and C. Liu. A reliable multicast framework for light-weight session and application level framing. *SIGCOMM Computer Communications Review*, 25:342–356, August 1995.

[4] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. In *SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Vancouver, Canada, August 1984. ACM.

[5] J. Hintikka. *Knowledge and Belief.* Cornell University Press, 1962.

[6] P. V. Mockapetris and K. J. Dunlap. Development of the domain name system. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 123–133, Stanford, CA, August 1988. ACM.

[7] E. M. Schooler. A multicast-based user directory service for synchronous rendezvous. Master's thesis, California Institute of Technology, Pasadena, California 91125, May 1996.