Resource Location in Very Large Networks

Partha Dasgupta
Dept. of Computer Science, Courant Institute, New York University.
and
Dept. of Computer Science and Engg., Arizona State University.
partha@cs.nyu.edu

Abstract

Networks such as the Internet and the telephone system are examples of ubiquitous large networks. While such networks are quite common, what is not so common is completely scalable, non-hierarchical naming that is independent of the entities location or affiliations. The problem of providing a unique, logical name for each nameable entity that can be kept immutant over the lifetime of the entity has been studied in many contexts. However, most results are not scalable for really large worldwide networks.

In this paper we first motivate the need for such naming schemes and then explore the possible name translation solutions. We develop a set of pragmatic criteria that a solution has to provide. The scenario we use is a hypothetical, but desirable naming scheme for telephony. In our scheme, each resource (human, telephone or computer) has a unique name that is neither mutable, nor dependent on any hierarchy of domains or geographical boundaries.

We then provide an engineering solution for resolving the location of a resource given its name. This solution has to be efficient, scalable and fault-tolerant. The solution uses several disjoint techniques such as caching, physically hierarchical servers and hashing. We show how our scheme meets our goals and argue its feasibility.

1. Introduction

Locating a named resource in a large distributed system has always been an interesting problem, especially if the name is not location dependent and the location is not static. This problem is getting renewed attention with the advent of mobile computing. Actually, the problem is quite a bit more interesting and significantly challenging if we consider it in the context of the possible realm of worldwide networks and the human element.

A logical name of a resource is a unique identifier for that resource. Sometimes this name is also referred to as a capability or surrogate. Possessing the name (or capability) allows a user to access the resource. Resources can have several types of names, that is user-given names, system-generated names, physical-identifiers, logical-identifiers and so on. Of these various types of names, a system-generated physical identifier is the most efficient form of naming. That is, this identifier has information about where exactly the resource is located and hence allows the fastest access to the resource. However, most systems do not use physical identifiers as using them produces lack of flexibility.

A well known problem in distributed systems is the name resolution problem. That is given the logical name (or capability) of a resource, we need to obtain the physical name. For example, the physical name of a particular Unix file is the tuple <*ia, dn, pn, in*>, where *ia* is the Internet address of the machine, *dn* is the device number for the storage device on which the file is located, *pn* is the partition number within the device and *in* is the i-number of the file. However, such names are never used in programming, as people tend to prefer user-friendly names such as "`cs.nyu.edu:/pub/tech/report1.ps`".

Naming gives rise to a host of problems. How are names generated? How are they used? What does a name mean? What are the different name types and why? And so on. In this paper, we want to address only one important facet of this scenario, i.e. given a name, which physical entity does it refer to? Since a physical entity is located at a particular controlling site (a computer node, a telephone exchange, etc.) at any point in time, knowing the controlling site allows us to access the resource. Hence the problem of name resolution boils down to identifying the physical site that currently controls (or has access to) the entity.

Formally, the problem can be stated as follows: There are p resources $\{R_1, R_2, \ldots R_p\}$, and q sites $\{S_1, S_2, \ldots S_q\}$. At some point in time, resource R_i is actually located at site S_j. Problem: *Given R_i, determine S_j.*

This paper assumes the existence of a very large (possibly worldwide) network which contains a very large number of resources (possibly hundreds per human). To intuitively model this scenario we use the example of a telephone system throughout the paper. However, we will point out

how the results are applicable to any large network (such as the Internet).

2. Uniform Telephone Numbers

The formal description of the name to location translation problem does not intuitively capture the real scale and complexity of the problem, so let us view it in a more practical setting. Consider the largest network of all: *the telephone network*. Suppose that each of all telephone subscribers in the world were given (or were allowed to pick) a unique "number" (or rather name). The assigned name would not have to be changed, regardless of the location of the person or the type of phone access used by the person.

Currently the telephone system uses physical identifiers. That is the complete telephone number is a physical address, composed of the country code, the city (or area) code, the code for a particular exchange in that city and a offset for the actual line that is connected to the recipient. This form of physical addressing is very efficient in routing calls, but not so efficient from the users point of view. When a person moves s/he has to change all telephone numbers. Numbering for mobile telephones are assigned with the assumption that the phone will be used primarily in one service area. Again, if the mobile phone user moves primary service area, the number has to be changed.

Let us assume the existence of a different, globally standard telephone numbering (or naming) scheme, such that a unique identifier is assigned to John, irrespective of where he lives. Now the problem can be restated as follows: *If Mary dials John's number, how does Mary's local telephone exchange determine where John is currently located?*

2.1. A Numbering Scheme

We digress a little from the focus of the paper to describe, informally, how a uniform numbering scheme may be designed and how it becomes quite useful.

Suppose in this scheme, each human is assigned a unique name, based on his/her real name. For example, I could be designated as: *partha.dasgupta.xxx*. The xxx ensures that the name is unique worldwide. How to ensure uniqueness is a subproblem that we do not discuss here.

Once a person is assigned a unique id, it can be used like a domain. Various different ids can now be set up to provide different services pertaining to the individual. For example, given that I have the id *partha.dasgupta.xxx*, I have the ability to set up several logical identifiers for publicly accessible resources owned by me.

For example, the following names can be set up:

*partha.dasgupta.xxx.**home*** - home telephone.
*partha.dasgupta.xxx.**office*** - office telephone.
*partha.dasgupta.xxx.**urgent*** - will find me anywhere.
*partha.dasgupta.xxx.**voicemail*** - just messages.
*partha.dasgupta.xxx.**tcp*** - telnet access.
*partha.dasgupta.xxx.**ftp*** - ftp access.
*partha.dasgupta.xxx.**smtp*** - electronic mail access.

Of course the above scheme is useful but too simplistic. After a standards committee works on this it will be converted to something too complex to be used!

Each physical device (telephone, computer etc.) actually has a physical address, such as the present telephone number or IP address. When I plug my home telephone into *any* phone jack, it informs the exchange that it has the logical id "*partha.dasgupta.xxx.home*". From that point all calls to my home telephone arrives at the physical number of that phone jack.

This scheme can be extended to cover mobile telephones and mobile computers seamlessly. When a mobile telephone enters a service area, it informs that cellular service provider of that event, and calls are to be directed to its current location, henceforth.

3. Finding Physical Locations - Related Results

As stated earlier, when Mary dials John's logical phone number, the local exchange Mary is connected to at this point in time has to determine where John is located currently. We postpone the discussion of our scheme, but we survey some related results.

3.1. Telephony Solutions

Currently, location finding in telephone systems are used in several cases. These include "800" type numbers in the US, cellular numbers and roaming services for cellular subscribers. None of the present schemes are extensible to the above scenario:

800-numbers: In the US, an 800 number is a logical number that translates to a plain telephone number. This translation is handled by the local telephone exchange. The translation depends upon the company handling the 800 service for that particular 800 number. Most companies have a centralized database and the local exchange sends a message to that database server to obtain the translation. After the translation is done, the call is placed like a regular call. Replicated databases for higher availability is being phased in.

Regular Cellular Calls: When a cellular number is dialled, the cellular central office (also called a MTSO) recalls the last known location of the called party. It then starts a broadcast search for the cellular phone starting from the last known location and radiating outwards, until all the cell sites have been searched twice.

Automatic Roaming on Cellular Systems: In this scheme, when a cellular phone enters a new service area (that is a different city than where it was last), it informs the local cellular exchange of its presence. The cellular exchange then informs the home site of the cellular phone (based on its area code) of the phone's current location. All incoming

calls to the phone first reaches the home system and then is call-forwarded to the current location.

Worldwide Satellite Systems: Systems that provide worldwide mobile communications, such as the soon to be realized *Iridium* system have the ability to do worldwide broadcasts. The phone number assigned to a Iridium phone will be recognized as a such, by the local exchange and sent to a uplink. The Iridium satellites will then broadcast a call signal to find that phone anywhere in the world. Of course, this can be used in a special purpose, low-traffic scheme such as Iridium, and does not scale to the regular telephone system.

The downside of the above techniques are discussed in section 3.4.

3.2. The Domain Naming System

The Domain Naming System (DNS) is used by the Internet. It is a popular, scalable naming system that has proven itself. In DNS, the worldwide network of machines are grouped by country. In each country it is optionally grouped by type of organization (academic, government, commercial and so on). Within each group, a particular entity has a unique domain-name. Inside each domain the machine names are assigned by the organization owning the domain name.

For each domain, there is one or more well known name server. At the top level there are countrywide servers that are cognizant of the servers for all the domains in that country. To translate a name that is not in the country of the caller, the caller's machine contacts a set of domain servers starting at the closest name server, going through a search set until the name of the machine in question is resolved by the domain-name server of the domain containing the destination machine.

The scalability of the domain naming scheme is a product of the hierarchical nature of the name. The complete name of a site is based on countries, affiliations or organizations. Thus this system is not quite location independent. Similar schemes are used in the OSI Directory Service and the X.500 directory service [BCDS93, OSI88].

3.3. Resource Location in Distributed Systems

The name resolution problem has been extensively studied as the resource location problem in distributed computer systems. Early work in this area is described in [Sal78, Sal81]. Many distributed operating systems have built-in name resolution services. These include Amoeba [MvRT*90], The V System [CM89], Clouds [PD88] and Locus [PWE*83]. Most of these systems maintain caches of recently translated names and fall back on broadcast or multicast methods when there is a cache miss.

The Clearinghouse system uses a naming scheme that is similar to the Internet DNS [OD83]. It creates hierarchies of domains. The Profile naming system [Pet88] is an attribute based naming system that extends the DNS type names and allows lookups to use semantic attributes. The role of contexts in naming is studied in [CP86].

The Galaxy distributed operating system [SMS*91] uses a well designed name resolution system that keeps track of Unix style names in a distributed environment. Replicated tables and servers [SMS92] provide fault-tolerance. The Galaxy scheme is somewhat similar to the Prefix Tables used in [Wo86]. Replicated name services are also presented in [SFP89].

In this paper we advocate against using hierarchical naming. Non-hierarchical naming is also proposed by the "structure free" naming system proposed in [Ter86]. This concept is taken further into multi-structured naming in [SM92]. These results are in some way similar to our work, however the techniques used or proposed do not really apply to the problem as stated in this paper. A scheme for handling naming and addressing of mobile objects without the use of unique identifiers is proposed in [FY92]. The problem statement of [FY92] is really close to the problem we are trying to solve. The solution however uses physically hierarchical servers and forwarding links. Our solution has a similar structure, but goes beyond the hierarchical servers to provide greater scalability.

3.4. Problems with Existing Solutions

Most of the above schemes use one or more of the following conceptual approaches. The problems with these approaches are described below. Note that we are envisioning *all* telephone call traffic (plus Internet traffic, plus other communication traffic) to be channelled through one unified naming scheme.

- *Centralized Databases*

Using one centralized database (as in the 800 system) is not plausible not only due to the amount of data but also the need for looking up the database on every connection. Also, the reliability of the system is very important.

- *Replicated Databases*

The entire name table can be replicated into many different servers. Even if each server is capable of holding all the data (expected to be 10-1000 GBytes), then the updating of all the replicated databases whenever anyone changes location will be an intolerable expense.

- *Hint Table and Broadcast*:

This approach keeps a list of recently used translations at each site. If a name translation request matches an entry in the cache, the physical location is retrieved (of course, this is a hint, the actual location may have changed). If the location is not available or if the cache is wrong, a broadcast is initiated to find the resource.

This solution is infeasible due to impossibility of broadcast on worldwide networks. Also, every "wrong number" will result in a non-revenue generating broadcast!

- *Home Site and Forwarding Links*

An appealing solution is to embed a "*birthmark*" in the logical address. This is used in cellular telephony, Domain Naming and many computer resource location algorithms. That is, the site generating the name, embeds its own id into the name. All present telephone numbers are generated this way, as well as social security number generation uses this technique. The use of birthmarks seems to be an appealing solution for the stated problem. The site identified by the birthmark is considered to be the "*home site*". When a name is to be resolved, the home site is contacted. The resource keeps the home site informed as it moves. Also, the resource can leave forwarding links from the prior site to the next site.

This method has two flaws. Firstly, all (or many) calls to a person goes via the homesite. This is considerable overhead especially if the homesite and the present (semi-permanent) location are on different continents.

Secondly, the failure of the homesite will cause communication outages for a user who may have no connection with the homesite, i.e. a person born in Los Angeles, living in New york will become inaccessible to his/her friends in New York if a catastrophe disables communication into Los Angeles.

- *Domain Based Name Services*

The problem with Domain Naming Service (DNS) is the concept of the domain. The domain is actually a classification that makes the naming scheme logically hierarchical. Any classification (geographical, organizational, professional etc.) that is used also makes the naming location dependant. We feel that the naming scheme needs to be completely flat, if total location (and affiliation) independence is to be achieved. In the context of the DNS, a flat naming scheme is realized if all entities are domains. For example, my workstation can be named *bat.cs.nyu.edu* as long as I am affiliated with New York University. If I change affiliations or move away then the name has to change. If every person decides to have a unique domain, (*partha.dasgupta.xxx* as in the prior example), then we need domain servers for each person, and resolving the address for the domain server for a person is exactly the same problem we are trying to solve. This scheme is not workable under the present method used in DNS.

Many distributed systems solutions referenced above use the concepts similar to domains (also known as contexts). While the solutions vary, the underlying usage of domains make them not extensible to a worldwide system of structure-less unique identifiers.

- *Hierarchical Location Services*

Another appealing solution is the usage of hierarchies. As stated before, the naming system is logically non-hierarchical, but the lookup system can be structures as a geographical hierarchy. For example each country can have a top-level name server, that knows of the location of all people currently located in the country. This name server is connected to a set of statewide name servers, each having knowledge of all people in that state. Which in turn is decomposed into servers for cities and even possibly neighborhoods.

When a name is to be resolved, the caller contacts the closest, lowest level name server. If it is unable to resolve it, it sends it to the next higher name server. This method has some advantages. Due to localized knowledge at each server, the "local" calls are processed fast. Even regional calls do not go above the statewide servers.

However, there are several flaws with this approach. First, the national server *must* know of all locations of all subscribers in the country. Second, fault tolerance of the near-top-level servers are very important. Third, every change in every location has to propagate to the top-level servers, making the traffic too high at the upper levels (especially if mobile hosts are included in this scheme).

Finally, the scheme is infeasible to implement, considering that some names cannot be resolved within a country. Since a worldwide name server is too complex (it would be a centralized database), this means all country servers have to be contacted. This would also happen on *every* wrong number!

3.5. In Search of the Panacea

While it is obvious that such a scalable name service scheme is not only appealing, but may actually be implementable (or desirable) in the future, solutions to the scheme will be invaluable.

The bad news is that we have been unable to come up with a straightforward *algorithmic* solution to this problem, that is scalable and fault tolerant and works with feasible complexity. Note that a solution has to be sublinear, or rather as close to constant time as possible. Even a worst case behaviour of N (where N is the number of sites) is not feasible. Actually, the hardest problem is the detection of names that do not exist. Since in commercial terms, a call placed to a non-existent number is non-revenue generating, detection of such numbers efficiently is of importance.

We feel such a algorithmic solution indeed may not exist. So our view is that a good solution would be an engineering solution based on merging several existing solutions, intelligently.

4. A Scalable, Fault-Tolerant Solution

We present a solution that can address the location problem, as stated. This solution uses the following strategies:

- Client caching (or hint tables)
- Forwarding links.
- Limited physically hierarchical servers.
- Hashing functions.

We present the solution in terms of the telephone system. It can be modified to work on the Internet.

4.1. Client Caching

First, we start with intelligent telephones (or computers). Whenever a telephone successfully completes a call, it retrieves from the exchange (or network) the physical identifier (or location) of the called telephone (or site) and stores it in a local cache. On subsequent calls to a cached name the telephone presents both the logical-id and the physical-id of the called number to the telephone exchange. The physical-id is used to place the call and is checked at the receiving end, as sometimes the cached id may be wrong. Client caching allows a large number of calls to bypass the translation scheme and thus reduces the traffic on the location translation servers. The only times the logical number has to translated is when:

- A caller makes a call to a number s/he has never called before.
- The called party has moved.
- A number that has not been referenced for some time is not in the cache any more due to cache size limitations.

4.2. Forwarding Links

Cache misses on moving resources can be mitigated using forwarding links. When a resource moves from location A to location B, a forwarding pointer is kept at location B. Any calls coming to A is forwarded to B and also a return message is sent by B to the caller to update its cache entry.

The length of the forwarding links should be kept small to avoid too many hops and possible cycles. Forwarding links by itself generates some peculiar problems, such as long chains, disconnected chains due to failures, cycles and so on. We do not discuss this further. The forwarding links will further decrease the number of calls needing translation and make the caching scheme more efficient.

4.3. Limited Hierarchical Servers

The next step is to make all the translation that can be made locally, to be made locally. This will divert all local calls (low revenue) from needing external translations.

Every local exchange first searches its local tables to see if the call is local to that exchange. If not is passes it on to a city-wide (or regional) server. Thus the regional server will have to have information on *all* subscribers in that area. This hierarchy should be built as far as it is practical. After a few levels, the size of the regional (or super-regional) servers get too large and traffic gets too high causing bottlenecks.

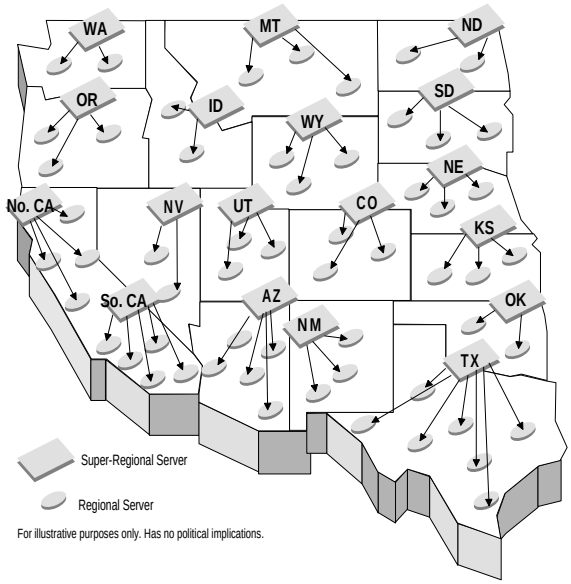

Fig. 1: A set of hierarchical servers for Western U.S.

Political boundaries can be used to set up regional and super-regional servers. For example, in the US one can conceive of the "state" as being the super-region while cities or counties as the regions. This method is somewhat self balancing as more populated parts of the county have geographically smaller states compared to the sparse parts of the country. Similar methods can be used in other countries and continents. (A hypothetical case is shown in Figure 1.)

If a call cannot be translated by the limited hierarchy of servers, then it uses the next scheme. Wrong numbers are also relegated to the next scheme.

4.4. Super Servers and Hashing

If we reach this stage, the number dialled is non-local, does not exist in the region and in some cases is simply wrong (or disconnected, or some other special case). The number of such calls are expected to be significantly smaller than the total traffic.

The solutions so far uses variants of existing methods. However when all the above optimizations fail, and a number is presented which cannot be translated in the limited contexts, there is need for a possibly worldwide search. As we mentioned earlier, it is not possible to have a worldwide search, nor is it possible (or feasible) to have one (or more) server(s) that keeps track all possible names.

So we use a somewhat novel scheme based on hashing to distribute the name management and name search problem to a set of partially disjoint servers and yet preserve some notion of locality. This scheme uses a worldwide network of *super-servers*. Each super-server has a name translation table that contains a fraction of the set of all names. Each name is however replicated in a set of super-servers. The name to server mapping is controlled by a set of well known hash functions. Our objective is to ensure that:

- A translation of a given name has to occur in a server relatively close to the calling site.
- The calling site does not have to (extensively) search for a server that has the name translation.
- Failure of a set of super-servers should not cause failure of the name translation mechanism.
- Wrong names are detected quickly.

Consider a set of N super-servers, scattered around the world. For every subscriber there are m super-servers that contain its current (or approximately current) location. If the total number of names in the entire system is P, then each super-server has translation entries for about $(mP)/N$ names.

As an example, consider:

- P = 20 billion or $2*10^{10}$ (total number of names),
- N = 500, (total number of super-servers) and
- m = 10 (number of super-servers, per name).

Then each super-server has 400 million entries and each entry is replicated in 10 super-servers. Even if the world population grows to 10 billion (current about 5 billion) the numbers above look reasonable. As networks get faster, more reliable and size of data storage devices get bigger, even larger numbers of names can be feasibly handled.

To make this scheme workable, we need to devise a globally known set of m hash functions $h_1, h_2, .., h_m$. These hash functions have the property, that given a logical number L, $h_i(L)$ returns the id of *one of the super-servers*. Thus every number hashes into m, hopefully distinct, super-servers. Actually the hashing functions should distribute each name into m distinct server-ids, and the set of all names evenly amongst the server-ids.

Now let us assume that for a logical number L, all m super-servers $h_1(L), h_2(L), .., h_m(L)$ are aware of the location of L. When a call to L cannot be completed locally or hierarchically, the calling site generates the list of m super-servers of L. It then contacts exactly *one* of the super-servers, presumably the closest one. The physical location is then obtained from this super-server. Of course, if the closest super-server is "dead" then any one of the others that is reachable will provide the translation. A wrong number will of course be detected by one call to a super-server. An illustration of the m and N supers-server configuration is shown in Figure 2.

The parameters m and N are important. In the worst case, m=N=1. In this case, there is 1 super-server, which is a centralized database of all resource locations. If m=N,

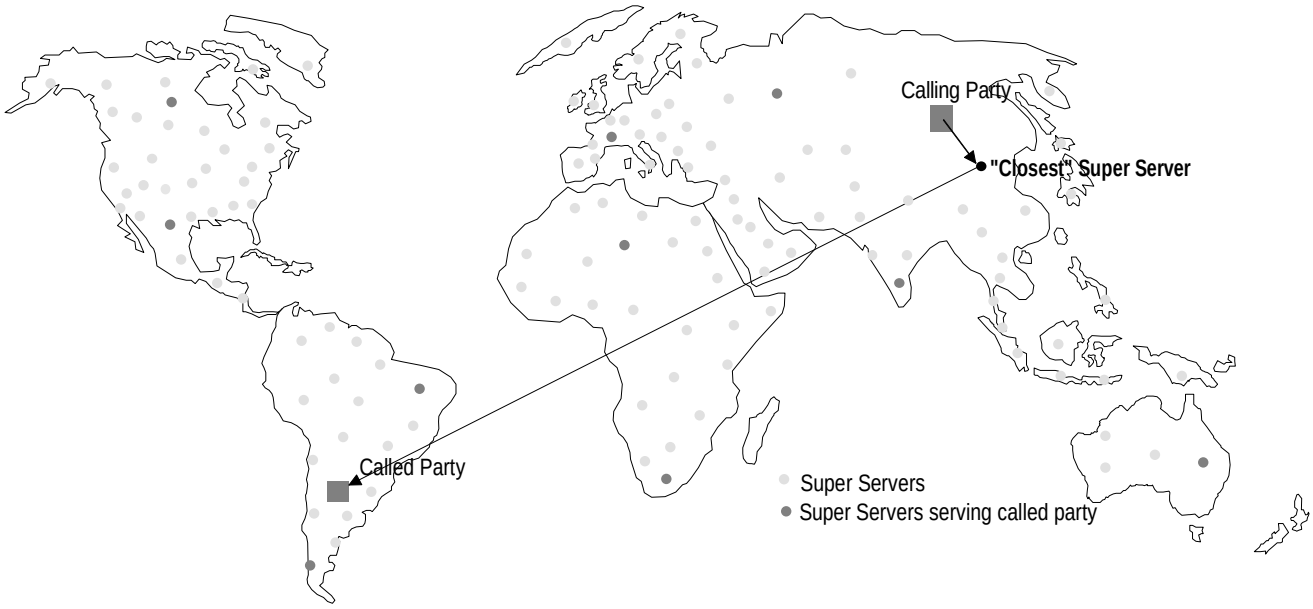

Fig. 2: The Super Servers and servers selected via hashing.

m>1, then we have m centralized databases, which is more failure tolerant, but the updating of the replicated database is a problem.

The system gets interesting if m<<N. As N gets larger, the size of the super-servers get smaller. *N* should be as large as possible, however, large *N* implies more cost. Also the value of *m* is critical. If *m*=1 then there is no fault-tolerance at the super-server level, that is each resource is controlled by exactly one super-server. If *m* is large, the fault-tolerance increases as does the chance of the proximity of a super-server to the calling site. However, the cost of updating the servers for mobile resources becomes larger (next section).

The hash functions have a very important and interesting part to play. Out of the (say) 500 super-servers in the world, only 10 (or less) know about the name that needs to be translated. Finding all 10 of these servers can be hard using any other technique. The hash functions immediately tell the calling site which 10 servers to contact. Then the calling site makes a determination as to which one of the 10 sites is closest. Given a good design of the hashing function, the 10 chosen sites will be reasonably scattered around the globe that the translation lookup over a data network will be quite cheap.

When a resource moves, it has to inform the 10 (or, rather, *m*) super-servers. Again the hash functions determine which *m* from all the *N* super-servers..

4.5. Resource Migration

When a resource changes location it has to:

- Notify the new site of its presence. The new site has to notify all the hierarchical servers above it.
- Notify the old site of its new location (forwarding chain). The old site may notify all the hierarchical servers of the forwarding (an optional step).
- Notify *m* super-servers of its new location. If not all *m* sites are available at the time of the move, the requests should be queued up and forwarded later.

The third step can be expensive, and is not recommended for highly mobile users. Updating the super-servers has to be intelligently combined with forwarding for highly mobile users. This aspect is not discussed in this paper.

4.6. Pitfalls

There are some situations when a call can fall between the cracks. This happens when a client moves and not all *m* of the super-servers are reachable from the new site. However, a caller may reach a server that has not been updated and get the old location. If the old location is dead, then the call will not complete even if the new location is reachable. Similar problems can appear as wrong numbers for valid numbers and so on. These problems occur if the availability of the super-servers are erratic. However, these problems have the property that they will disappear with time, and can be prevented by retrying other servers.

Failure of regional servers can also interfere with the translation scheme. However a name can be translated through the super-server scheme if a regional server is unavailable. This increases the translation costs. Other issues that are outside the scope of this paper are the actual costs of server updates, maintaining coherence between the regional and super serves. Also maintaining and garbage collecting forwarding links is an issue. Finally, the repair of the server information upon server crashes or inadvertent divergence of the information due to temporary communication outages need to be further studied.

Updating the super-servers for each move involves worldwide messages. This may be fine is resources move infrequently (e.g. a person changes residences or jobs) but will be too expensive for daily moves. Mobile resources hence should use a modified algorithm, that has an "*anchor point*" for each mobile resource and uses forwarding out of the anchor point. Note that the anchor point is not the same as the home-site in current cellular systems as the anchor point can be changed without changing the name of the resource.

4.7. Properties

Above, we briefly presented a location system that can track very large number of resources in a very large network. It works with low overhead and is scalable. The system is tunable using parameters (such as *m, N,* depth of hierarchical servers, number of retries on servers and so on.)

The algorithm is suitable for implementing worldwide location independent site addressing and will even work for telephone networks. Using it for computer networks is actually simpler. The system can also find non-existent identifiers without heavy translation penalty.

This algorithm, with optimization, can be used for highly mobile systems such as cellular and paging systems as well as wide area mobile computing (as described above).

As long as one of the *m* super-servers serving a destination is operational, the name can be resolved. Thus the system is fault tolerant.

5. Conclusions

The problem of providing globally unique site identifiers in a large mobile network is not simple. There is no straightforward solution. However, by carefully synthesizing existing techniques, we can provide a workable solution.

We feel the above solution has plenty of room for improvement. The parameters and their impact need to be studied as well as additional techniques can be added, particularly for the maintenance of coherence amongst the super-servers.

6. References

[BCDS93]
P. J. Bumbulis, D. D. Cowan, C. M. Durance, and T. M. Stepien. An Introduction to the OSI Directory Services. *Computer Networks and ISDN Systems*, (26):239–249, 1993.

[BLNS82]
A. Birrell, R. Levin, R. M. Needham, and M. D. Schroeder. Grapevine, An Exercise in Distributed Computing. *Communications of the ACM*, 25:260–274, 1982.

[CM86]
D. E. Comer and T. P. Murtagh. The Tilde File Naming System. *6th International Conference on Distributed Computing Systems*, pages 509–514, IEEE Computer Society, 1986.

[CM89]
D. R. Cheriton and T. P. Mann. Decentralizing a Global Naming Service for Improved Performance. *ACM Transactions on Computer Systems*, 7:147–183, 1989.

[CP86]
D. E. Comer and L. L. Peterson. A Model of Name Resolution in Distributed Systems. *6th International Conference on Distributed Computing Systems*, pages 523–530, IEEE Computer Society, 1986.

[FY92]
N. Fujinami and Y. Yokote. Naming and Addressing of Objects without Unique Identifiers. *12th International Conference on Distributed Computing Systems*, pages 581–588, IEEE Computer Society, 1992.

[ISO88]
ISO/IEC, Information Processing System–Open Systems Interconnect–The Directory, IS 9495, Part 1 to Part 8. *The International Organization for Standardization*, Geneva 1988.

[MvRT*90]
S. J. Mullender, G. van Rossum, A. S. Tanenbaum, R. van Renesse, and H. Staveren. Amoeba: A Distributed Operating System for the 1990s. *Computer*, 23(5):44–53, May 1990.

[OD83]
D.C. Oppen and Y. Dalal. The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment. *ACM Transactions on Office Information Systems*, 1:230–253, 1983.

[Pet88]
L. L. Peterson. The Profile Naming Service. *ACM Transactions on Computer Systems*, 6:341–364, 1988.

[PD88]
D. V. Pitts and P. Dasgupta. Object memory and storage management in the Clouds kernel. *8th International Conference on Distributed Computing Systems*, pages 10–17, IEEE Computer Society, June 1988.

[PWE*83]
G. Popek, B. Walker, R. English, C. Kline, and G. Thiel. The LOCUS Distributed Operating System. *ACM-SIGOPS Operating Systems Review*, 17(5), 1983.

[RR91]
K. Ravindran and K. K. Ramakrishnan. A Model of Naming for Fine-grained Service Specification in Distributed Systems. *11th International Conference on Distributed Computing Systems*, pages 98–105, IEEE Computer Society, 1991.

[Sal78]
J. Saltzer. Naming and Binding of Objects. *Lecture Notes in Computer Science*, 60, 1978.

[Sal81]
J. Saltzer. Identifiers (naming) in Distributed Systems. *Lecture Notes in Computer Science*, 191–210, 1981.

[SFP89]
S. Sarin, R. Floyd, and N. Phadmis. A Flexible Algorithm for Replicated Directory Management. *9th International Conference on Distributed Computing Systems*, pages 456–464, IEEE Computer Society, 1989.

[SM92]
S. Sechrest and M. McClennen. Bendig Hierarchical and Attribute-based File Naming. *12th International Conference on Distributed Computing Systems*, pages 572–580, IEEE Computer Society, 1992.

[SMS*91]
P. K. Sinha, M. Maekawa, K. Shimuzu, X. Jia, H. Ashihara, N. Utsonomia, K. S. Park, and H. Nakano. The GALAXY Distributed Operating System. *Computer*, 24(8), pages 34–41, 1991.

[SMS92]
P. Sinha, M. Maekawa, and K. Shimizu. Improving the Reliability of Name Resolution Mechanisms in Distributed Operating Systems. *12th International Conference on Distributed Computing Systems*, pages 589–596, IEEE Computer Society, 1992.

[Ter86]
D. B. Terry. Structure-free Name Management for Evolving Distributed Environments. *6th International Conference on Distributed Computing Systems*, pages 502–508, IEEE Computer Society, 1986.

[WO86]
B. Welch and J. Ousterhout. Prefix Tables: A Simple Mechanisms for Locating Files in a Distributed System. *6th International Conference on Distributed Computing Systems*, pages 184–189, IEEE Computer Society, 1986.