ＧＡＬＡＸＹ分散オペレーティングシステムにおける名前とその解決

Pradeep K. Sinha　　清水　謙多郎　　芦原　評　　賈　小華　　前川　守

東京大学理学部　情報科学科

オペレーティングシステムの設計上、オブジェクトの名前付けは重要な位置を占める。特に、多数のオブジェクトがシステム全体に配置された分散オペレーティングシステムの場合、これらのオブジェクトにシステム全体で通用する名前を割当てる必要性から、名前付けはより重要な問題となる。名前付けのシステムは、ユーザーによって文字列名をオブジェクトに割当てさせ、以後その名前によってそのオブジェクトを示すようにさせる。ＧＡＬＡＸＹにおいて、名前の解決とは、このユーザーによって決められたオブジェクト名を対応するシステム内名にマッピングすることである。名前の解決は最も頻繁に行なわれる操作の一つであり、従ってその効率と信頼性はシステム全体を左右するものとなる。本論文は、ＧＡＬＡＸＹ分散オペレーティングシステムでのオブジェクトの名前付けと名前の解決の概念と機構を記述する。効率、拡張性、柔軟性、およびユーザー指定可能な信頼性は本機構の特徴である。

Names and Name Resolution in the GALAXY Distributed Operating System

Pradeep K. Sinha, Kentaro Shimizu, Hyo Ashihara, Xiaohua Jia and Mamoru Maekawa

Department of Information Science,
Faculty of Science, University of Tokyo
7-3-1 Hongo, Bunkyo-Ku, Tokyo-113, Japan

Naming plays an important role in the design of any operating system. Especially in case of distributed operating systems, where a large number of objects are distributed all over the system, the need to assign global names to all the objects makes naming a more important issue. A naming system allows users to assign character-string names to objects and subsequently use these names to refer to the corresponding objects. In GALAXY, name resolution is the process of mapping an object's user-defined name to its corresponding system-defined unique identifier. Name resolution is one of the most frequently used operations. Hence the efficiency and the reliability of this operation is very critical for the entire system. In this paper we describe the concepts and mechanisms for object naming and name resolution in the GALAXY distributed operating system. Efficiency, scalability, flexibility and user-definable reliability are some of the salient features of our mechanisms.

1. Introduction

This paper discusses the concepts and mechanisms for realizing network-transparent object naming in distributed operating systems. Multiple global naming contexts and descriptive naming based on hierarchical names are used in our approach. These are desirable features of names in distributed operating systems for better efficiency, flexibility, and usability. We also discuss our name resolution mechanism which maps an object's user defined name to its system defined unique identifier. Unlike the conventional mechanisms, this mechanism allows the users to define their own reliability requirements for the resolution of certain object names from certain nodes of the system. In addition, a hierarchical name cache is used for improving the efficiency of our name resolution mechanism. Our naming and name resolution mechanisms aim at transparency to physical location and structure of objects. This is one of the most important requirements in distributed operating systems. The concepts and mechanisms proposed in this paper are developed in the GALAXY [4, 11, 12] distributed operating system.

2. The Three Level Naming Scheme

The two basic classes of names widely used in operating systems are: *human-oriented names* and *system-oriented names*. Human-oriented names are defined and used by the users. For improved usability, human-oriented names should be independent of the physical location and structure of objects which they designate and should be flexible enough to allow a user to define his own names rather than simply identify the objects.

Human-oriented names are not unique for a particular object and are variable in length not only for different objects but even for the different names of the same object. Hence they cannot be easily manipulated, stored, and used by the machines for identification purpose. Therefore in addition to human-oriented names, which are useful for users, system-oriented names are needed to be used efficiently by the system. They should be capable of uniquely identifying the objects and should be generated automatically in a distributed manner. They are basically meant for use by the system but may also be used by the users.

Figure 1 shows a simple naming model based on these two types of names. The GALAXY distributed operating system is designed based on the above naming model. In GALAXY, human-oriented names are called *external names* and system-oriented names are called *unique identifiers* (ID in short).

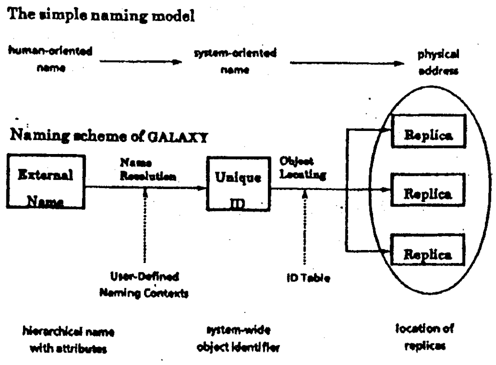

Figure 1. The simple naming model and the naming scheme of GALAXY.

3. Human-Oriented Object Naming

Network transparency is one of the most important requirements of the naming scheme in distributed operating systems. In addition, efficiency and flexibility of use are equally important. Basically, there are two approaches to realize global naming in conventional distributed operating systems:

(1) Using separate name space for each node.

(2) Using a single global name space for all nodes.

Early network operating systems such as Newcastle Connection [2] and COCANET [8] used the first approach of global naming. They did not have the property of location transparency because the location of an object had to be explicitly incorporated in its name. Many recent distributed file systems such as NFS [9] and RFS [6] use the first approach along with a method called *remote mount* for achieving the goal of location transparency in naming. The problem with these systems is that the same object may have different *absolute* names when viewed from different nodes. Moreover, to take care of the transparency issue in such systems, n^2 mounts have to be performed in a system having n nodes. This makes the management very difficult.

Many recent distributed operating systems such as LOCUS [13], ANDREW [5], V system [3], and Saguaro [1] use the second approach for the transparent global naming of objects. In these systems, requests from all the nodes are served by first searching this global name space. Although many of these systems support transparency to both the location of the accessing object and the location of the accessed object, the main problem with these systems is the poor reliability, efficiency, and scalability of their name resolution mechanisms.

In order to take care of the transparency feature, GALAXY also uses a single global name space for its user defined external object names. However, to overcome the limitations of the existing systems, GALAXY supports an efficient and reliable name resolution mechanism, the details of which are given in Sections 5 and 6. In addition to this, GALAXY also provides its users with the flexibility to define contexts and to assign attributes with external names as described below.

3.1. Multiple Global Naming Contexts

To avoid the problems of inconvenience of use and inefficient resolution of long pathnames in case of single global name space, the first step taken in GALAXY is to provide the flexibility to the users to define their own naming contexts. A *context* is basically a pathname of the single global name tree starting from its root. A GALAXY user can define his own contextname for a particular pathname. For example, a particular user may specify that the pathname */user1/project1/group1* be designated as *mycontext1*. Now when the user wants to use the object having pathname */user1/project1/group1/file1*, instead of specifying the complete pathname, he can only specify the contextname *mycontext1* and the remaining components of the object's pathname which is *file1* in this case. To facilitate this, the basic naming syntax of an external name in GALAXY is:

[Contextname]Pathname

Thus GALAXY's external names may be of two types: *absolute* and *relative*. In its absolute form, an external name consists of the complete pathname of the corresponding object starting from its root. On the other hand, in its relative form, an external name consists of a contextname and a pathname which is relative to the given context.

For the purpose of managing the user defined contextnames, we use the concept of *node groups* in our approach. In this method, the entire system is hierarchically partitioned into small groups of nodes. Each node group has a group leader which is responsible for maintaining the information about the nodes belonging to its group. For contextname management, the leader node maintains a *group context table* which is a mapping of all the contextnames and the corresponding nodes belonging to its group. Each node also has a *local context* *table* which contains the locally defined contextnames and the system-wide identifiers of the corresponding directory files.

Now when a user specifies a contextname to be used, the searching process for that contextname starts from the local node and proceeds by searching the family tree of the node groups in the order of their relationship. That is, closer relatives are first consulted as compared to distant relatives. Note that the efficiency of this scheme lies in the proper grouping of the nodes.

3.2. Descriptive Naming

Flat names with attributes are widely used in daily life. Names of people is a typical example of this type of naming. However, due to the problem of uniqueness of name representation and difficulty in address mapping, instead of flat name spaces, hierarchical names are used by most operating systems — UNIX [7], NFS [9], Sprite [14], LOCUS [13], V system [3] are few examples of such systems. In hierarchical names, it is easy to represent the hierarchical structure of objects and to manage them as a group. But, hierarchical names cannot easily represent various relations among objects and require much overhead to resolve. In order to incorporate the advantages of hierarchical names and to overcome their disadvantages, GALAXY uses the hierarchical names as the basis and provides the facility for attaching naming attributes with the links between two components of a hierarchical name. Naming attributes are the properties of the object being named. Such properties are represented as labels attached to the links between directories. By linking names, it is possible to define the semantic structure of objects.

4. System-oriented Object Names

Every object in GALAXY has a unique identifier called *Unique ID* (ID in short). In GALAXY, one object can have multiple replicas. All replicas of an object use the same ID irrespective of their locations. An ID identifies an object, but its structure and management mechanism are irrelevant to the contents, external name, or physical addresses of the object's replicas.

In order to guarantee the system-wide uniqueness of each ID, the ID format consists of two fields: *time stamp (TS)* field and *node number (NN)* field. Each field consists of 8 bytes. The TS field contains the time stamp assigned to the ID by the node that has created the ID. The NN field contains the node number where the ID is created. It may be observed that even if the unit of time for incrementing the TS field is 1 μs, the time period represented by the TS field is 2^{64} $\mu s \approx 3 \times 10^5$ years. Obviously, this time period is dominant over the life-time of the system. Similarly, the NN field is long enough to assign unique numbers to all the nodes.

5. Name Resolution

As shown in Figure 1, an external name is first mapped to its corresponding ID which in turn is mapped to the physical locations (node numbers) of the replicas of the concerned object. In this paper, we define *name resolution* as the process of mapping an external name to its corresponding ID and *object locating* as the process of mapping an ID to the replica locations of the concerned object. In this section we will discuss about the name resolution mechanism of GALAXY. GALAXY uses a direct object locating mechanism which will be briefly discussed in Section 5.1.2 because our name resolution mechanism is based on our object locating mechanism.

5.1. Data Structures

The two data structures used for our basic name resolution mechanism are discussed below.

5.1.1. Directories

In GALAXY, each node of the single global name tree represents an object. Excluding the leaf nodes, all other nodes of the name tree are *directory* objects. In conventional operating systems, directories are used to map an object's name to its physical location. Thus in these systems, a directory entry consists of a component name and the corresponding object descriptor pointer such as inodes in UNIX [7] and vnodes in NFS [9]. Unlike these systems, in GALAXY, a directory entry is a (*component name, ID*) pair which maps a component name of an object to its system-wide unique ID. These directories are regular GALAXY objects which are distributed among the various nodes of the system and can be replicated and migrated just like any other object.

5.1.2. ID Table

For mapping of IDs to the physical locations of replicas, all the IDs of the entire system are stored in a system-wide table called *ID Table*. An ID Table entry (called IDTE) contains information about the type of the object, access control list for the object, locations of the object's replicas (*replica list*), and locations where the copies of this IDTE exist (*copy list*). The replica list helps in returning all the locations of the desired object as a result of the object locating operation. By the copy list, all IDTEs of the same object are linked together so that any modification can be consistently made to all copies through this link.

In a usual non-distributed system, the ID Table can be managed in a centralized manner. But in case of a distributed multiple-host system such as GALAXY, it is not efficient and reliable that some *central* node keeps the entire ID Table. Conversely, it is also not realistic for every node to have a copy of the entire ID Table. Thus, in GALAXY, each node has a partial copy of the ID Table. The copy of the ID Table of a particular node contains entries for the following IDs:

(1) IDs that are contained in the directories on the node or in a name cache of that node or the context IDs present in the local context table of that node. The presence of these IDTEs in the local ID Table of a node is necessary for the direct locating of the directory file objects during the name resolution process.

(2) IDs that are being used by the processes running on the node. These IDTEs are necessary in a node's ID Table for the direct locating of the objects being used by the processes of that node.

Availability of the entries for these two categories of IDs in the copy of a particular node's ID Table ensures the direct locating of any object from any node given the object's ID. Further details of ID Table management and consistency control of IDTEs are given in [12] and [4] respectively.

5.2. The Basic Name Resolution Mechanism

There are two approaches to pathname expansion (name resolution) in a distributed environment [10]:

(a) Transparent pathname expansion and

(b) Remote pathname expansion.

Due to its advantage of low network traffic for expansion of pathnames that contain many directories all stored at a single node, the method of remote pathname expansion is used as the basic mechanism for name resolution in GALAXY.

To facilitate the use of the method of remote pathname expansion, the root directory is replicated at all the nodes of GALAXY. Now when a user requests for a pathname expansion, the name resolution process starts at the client node by first searching the local root directory for the next component name of the pathname. The corresponding ID is extracted from the root directory and is searched in the local ID Table to get the replica locations of the corresponding object. This ID will certainly be available in the local ID Table due to our replication policy of IDTEs discussed above. Now a message is sent to one of the nodes in the replica list with the remaining pathname components for further expansion. The next component of the remaining pathname is searched in the concerned directory at the selected node. The name resolution operation continues like this until all the pathname components have been resolved and the desired object's ID has been extracted.

The basic name resolution mechanism discussed above is for the resolution of absolute external names. In case of relative external names, the basic name resolution mechanism involves the following two steps:

(a) The first step is to get the location of the directory corresponding to the specified contextname.

(b) And the second step is to expand the specified pathname starting at the location obtained in step (a).

Step (a), is carried out by using the hierarchical node group concept of Section 3.1. When the desired contextname has been located, its corresponding ID is extracted from the local context table of the node on which it is found (say N_i) and the ID Table of node N_i is searched to get the replica locations of the directory object corresponding to the specified contextname. One of the replica nodes (say N_j) is selected from the replica list and the pathname of the external name is now forwarded from node N_i to node N_j for expansion. From now onwards, step (b) is carried out by the method of remote pathname expansion starting at node N_j.

5.3. Name Cache for Name Resolution Efficiency

The efficiency of the basic name resolution mechanism is very poor especially for those pathnames whose component name directories are scattered on different nodes of the system. To improve the efficiency of name resolution, in GALAXY name caches are used at each node for caching of necessary directory entries. At a particular node, a separate name cache is created for each context that corresponds to an object's relative external name cached on that node. The root context also forms a separate name cache at each node which is used for the expansion of absolute external names. Name cache structure is the same as that of hierarchical directory structure of the name space. A particular node's name cache consists of those directories and directory entries which correspond to the contextname and the component names of the pathname of an object that was recently used to access the object from the node. Further details of name cache management and name cache consistency control mechanism are given in [12].

5.4. The Improved Name Resolution Mechanism

In the improved mechanism, when an object is accessed from a particular node by using its external name, the pathname components of the external name are searched in the local name cache corresponding to the context of the external name. It may be noted here that the root directory acts as the context for absolute external names. The pathname components are searched one-by-one in the hierarchical name cache directories just as they are searched using regular directories.

If the pathname of the desired object consists of n component names out of which n_1 components were found in the local name cache, then for the remaining $(n-n_1)$ components, the searching has to be continued somewhere outside the local name cache. So the ID corresponding to the last component name found in the local name cache is extracted from the name cache and is searched in the local ID Table to get the locations of the corresponding object. This ID will certainly be available in the local ID Table due to our replication policy of IDTEs at various nodes. Now a message is sent to one of the nodes in the replica list with the remaining $(n-n_1)$ pathname components for further expansion. The next component of the remaining pathname is searched in the concerned directory at the selected node. The name resolution operation continues like this until all the pathname components have been resolved and the desired object's ID has been extracted.

In case of a *complete miss*, when a name cache corresponding to the desired object's external name context is not found on the client node, the hierarchical node group method of Section 3.1 is used for searching the node (say N_i) on which the ID corresponding to the concerned context is stored. In this case, the name resolution process starts from node N_i instead of the client node in a similar manner as discussed above.

6. Reliability of Name Resolution Mechanism

The name resolution mechanism of a distributed system can be called highly reliable if it is possible to satisfy all name resolution requests generated from any node of the system at any instance of time. To achieve such a high degree of reliability of the name resolution mechanism, all the directories must be replicated at all the nodes of the system. This may be prohibitive in terms of space and time overheads. Hence a better idea is to define a set of reliability parameters and let a user choose and specify through these parameters the degrees of reliability desired for the (node, external name) pairs for the various objects and object names being used by him from the specified node. This seems logical because all the objects in a system are not of equal importance to all the users and out of all the nodes in a large distributed system, a particular user normally works at only a few nodes of the system. By using this approach, it is possible to satisfy the degree of reliability of resolving various names as desired by the users without affecting the overall system efficiency to a large extent.

6.1. Reliability Factors

Given an object's pathname, the reliability of resolving the pathname from a particular node is greatly influenced by the locations of the directories or their replicas which correspond to the given pathname components. On the basis of the various possibilities of the locations of the replicas of the concerned directories, we define the following factors that influence the reliability of the name resolution operation.

6.1.1. Subpath reliability

For a (node, pathname) pair, the subpath reliability of the name resolution operation is the presence of the necessary directories on the client node (specified node) for tracing the components of the pathname up to the subpath right at the client node without communicating with any other node.

6.1.2. m-stage reliability

A name resolution operation is said to be m-stage reliable when m hops are required during pathname expansion for resolving the given pathname. In our method of remote pathname expansion, a *hop* is defined as the passing of the remaining pathname components from one node to another node which has the next component's directory when the remaining components of the pathname cannot be further resolved on the present node.

6.1.3. k-path reliability

A name resolution operation is said to be k-path reliable when there are at least k possible paths between any two contiguous directories corresponding to the components of the given pathname starting from its root directory to the last directory of the given pathname. Note that the term *path* here means the logical name resolution path (path of the pathname from one directory to another) and not the physical path of the network.

Using the reliability parameters mentioned above, a user in GALAXY can specify his reliability requirements for resolving certain pathnames from certain nodes. Depending upon the users' specifications, the system automatically replicates the necessary directories at the proper nodes.

6.2. Applicability to Context Based Relative External Names

We have shown the usefulness of the reliability parameters discussed above for the resolution of our absolute external names. However, as discussed before in Section 5.2, in case of relative external names, the name resolution process involves the following two steps:

(a) Searching the contextname using our hierarchical node group concept to get the location of the directory corresponding to the specified contextname. This process may require message transfers from one node to another several times until the node having the contextname in its local context table is reached.

(b) Expanding the specified pathname starting at the location obtained in step (a). In this step, the message transfers from one node to another is done exactly in the same manner as the resolution of absolute external names.

Thus in case of a relative external name, not only the concerned directories corresponding to the pathname of the given name but the corresponding contextname entry is also replicated at a suitable node of the system in order to satisfy the user-defined subpath, m-stage, and k-path reliability requirements for the (node, relative external name) pair.

7. Evaluation and Conclusion

The naming mechanism discussed above is reliable, flexible and efficient. We have not come across any distributed system in which the users have the flexibility to define the reliability of the name resolution operation. In GALAXY, the user-definable reliability parameters provide the flexibility to the users to choose and balance between their wide range of reliability and efficiency requirements. Unlike several other naming mechanisms, in our naming mechanism the unit of name resolution is an object instead of a group of objects. Thus individual objects can be migrated freely from one node to another or from one object manager to another with no change of object's name or no degradation of performance. Due to the use of name cache for caching the recently used object names, the efficiency of our name resolution mechanism is highly dependent upon the degree to which locality is exhibited in the use of object names. Measurements made by Sheltzer [10] and Cheriton [3] clearly show that a high degree of locality does exist in the use of object names. In addition, unlike V system [3] and Sprite [14], no broadcasting is used in GALAXY during name resolution even if there is a cache miss. This improves the scalability factor of our design and makes our mechanism suitable for both small and large networks. We believe that the concepts presented in this paper will be useful for the design of other distributed systems.

References

[1] Andrews, G. R., Schlichting, R. D., Hayes, R., and Purdin, T. D. M., The Design of the Saguaro Distributed Operating System, *IEEE Trans. Softw. Eng.*, 13, 1, (1987), 104-118.

[2] Brownbridge, D. R., Marshall, L. F., and Randell, B., The Newcastle Connection, *Softw. Pract. and Expr.*, 12, 12, (1982), 1147-1162.

[3] Cheriton, D. R., and Mann T. P., Decentralizing a Global Naming Service for Improved Performance and Fault Tolerance, *ACM Trans. on Computer Systems*, 7, 2, (1989), 147-183.

[4] Jia, X., Nakano, H., Shimizu, K., and Maekawa, M., Highly Concurrent Directory Management in GALAXY Distributed Operating System, *Proc. 10th Int. Conf. Distributed Computing Systems*, (1990), 416-423.

[5] Morris, J. H., ANDREW: A Distributed Personal Computing Environment, *Comm. ACM*, 29, 3, (1986), 184-201.

[6] Rifkin, A. P., Forbes, M. P., Hamilton, R. L., Sabrio, M., Shar, S., and Yueh, K., RFS Architectural Overview, *USENIX Conference Proceedings*, Atlanta, Georgia, (1986), 248-259.

[7] Ritchie, D., and Thompson, K., The UNIX Time-sharing System, *Comm. ACM*, 17, 6, (1974), 365-375.

[8] Rowe, L., and Birman, K., A Local Network Based on the UNIX Operating System, *IEEE Trans. Softw. Eng.*, SE-8, 2, (1982), 137-146.

[9] Sandberg, R., Goldberg, D., Kleinman, S., Walsh, D., and Lyon, B., Design and Implementation of the SUN Network File System, *USENIX Conference Proceedings*, Portland, Oregon, (1985), 119-130.

[10] Sheltzer, A. B., Lindell, R., and Popek, G. J., Name Service Locality and Cache Design in a Distributed Operating Systems, *Proc. 6th Int. Conf. Distributed Computing Systems*, (1986), 515-522.

[11] Shimizu, K., Maekawa, M., and Hamano, J., Hierarchical Object Groups in Distributed Operating Systems, *Proc. 8th Int. Conf. Distributed Computing Systems*, (1988), 18-24.

[12] Sinha, P. K., Shimizu, K., Utsunomiya, N. and Maekawa, M., A Highly Reliable and Efficient Object Locating Mechanism in the GALAXY Distributed Operating System, submitted to the *IEEE Trans. on Parallel and Distributed System*.

[13] Walker, B., Popek, G., English, R., Kline, C., and Thiel, G., The LOCUS Distributed Operating System, *Proc. 9th ACM SIGOPS Symp. Operating Systems Principles*, (1983), 49-70.

[14] Welch, B., and Ousterhout, J. K., Prefix Tables: A Simple Mechanism for Locating Files in a Distributed System, *Proc. 6th Int. Conf. Distributed Computing Systems*, (1986), 184-189.