Object Migration in the Guide System

D. Hagimont, P.Y. Chevalier, J. Mossière, X. Rousset de Pina

Bull-IMAG/Systèmes, 2 avenue de Vignate, 38610 Gières - France
E-mail: Daniel.Hagimont@imag.fr

1 Introduction

Object migration is a strong requirement for distributed persistent object stores. The purpose of object migration is not only to allow disk location modification for objects managed in the permanent store, but it is also often used as a means for gathering objects in some partitions also called clusters, in order to improve object management at the system level. It therefore provides the system and the applications with the ability to use their knowledge about inter-object relations for the management of clusters as some kinds of object working-sets (as in [3]).

Unfortunately, existing implementations of object migration incur an important overhead on naming functions (e.g. location, addressing). In the Guide system, we have designed and implemented a mechanism for object migration between clusters that avoids these drawbacks. The advantages of our location/migration policy may be summarized as follows: our uniform naming scheme facilitates distributed concurrent sharing (as opposed to local identifiers), object location is simple and efficient for objects that don't move, and the migration scheme reduces the location cost when correlated objects are gathered in clusters since it avoids extra mapping operations for migrated object locations.

This paper presents the proposed location/migration mechanisms, their integration in the Guide object-oriented distributed system, a performance evaluation based on analysis and a short comparison with related work.

2 The Guide system

The Guide object-based system has been implemented on top of the Mach 3.0 micro-kernel [1]. In this section, we summarize the aspects of the Guide system that are needed for describing the rest of this paper. A global description of the Guide implementation is available in [8].

In the Guide system, the execution structures are composed of Mach tasks (the abstraction that roughly corresponds to a Unix process).

Object sharing between tasks is implemented using virtual memory mapping between tasks (a mechanism identical to Unix shared virtual memory). Actually, since objects are grouped in clusters, the unit of mapping in tasks is the cluster. A cluster is dynamically mapped in the address space of a task at first use. In the current prototype, a cluster can only be mapped on tasks running on the same node. A thread of control requiring a cluster already mapped on another node executes a remote procedure call in order to share the cluster on that node.

For object addressing and particularly for object invocations, in order to avoid a systematic interpretation from the runtime system, we implemented a Multics-like segmentation mechanism [11] that allows interpretation to be only performed at first call. A reference in one object (say *O1*) to another object mapped in the same task is made through a linkage segment associated with *O1* in this task. If the reference is bound (i.e the linkage segment knows the address of the referenced object in the current task), then no system call is required. This scheme is detailed in [7].

3 Object Location and Migration in the Guide System

Requirements and motivations

In this paper, we investigate the design of object migration facilities in persistent stores in the purpose of object management improvement. If it is easy to move an object between clusters, it is much more difficult to implement object migration without an important overhead on mechanisms that relate to object naming. Therefore, we are particularly interested in the following requirements:

1) The migration of an object should reduce the cost of further locations when the object that requests the location is in the destination cluster (i.e. when the locator and the searched object are in the same cluster).
2) The migration mechanism should not penalize object location for objects that never move.
3) The migration mechanism should not degrade the efficiency of operations for objects' names usage such as object addressing, name assignment or name comparison.

Also, it is important to note that the migration mechanisms we want to provide are intended to be used at administration time (i.e. while the application is not running). This paper is not dealing with active objects migration that would be needed to implement load balancing.

The naming/location/migration scheme

In the Guide system, objects are named with 64 bit identifiers. This allows us to manage a very large persistent store. An object identifier is allocated at creation time and contains the initial location of the object (its cluster's identifier). Assuming that object migration is unfrequent, an object is often found in its creation cluster. With this naming and location scheme as a base, we designed the following migration mechanism.

The basic idea of our scheme is to avoid the major drawback of forwarders. With a solution based on a simple forwarder managed in the cluster where the object was created, the location of a migrated object always requires an indirection through the creation cluster. If this indirection was performed in physical memory, it would be acceptable. However, in a distributed persistent store, such an indirection may need the mapping of the creation cluster, and in the worse case, the mapping of the creation cluster may involve a request to a remote storage server through the network. This mapping is needless if the requested object migrated to a cluster that is already mapped (i.e. in the object working set of the application).

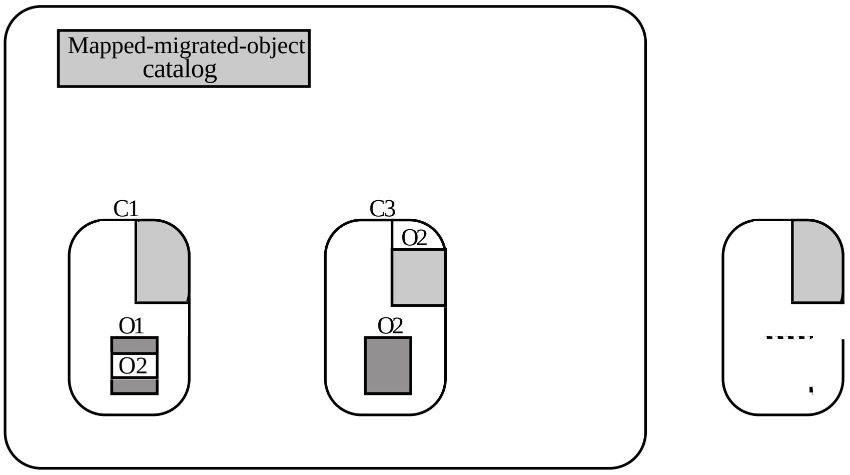
Therefore, the solution we are describing is based on the following assumptions:

- A mapping operation is expensive. Our main motivation is to avoid (when possible) the mapping of the creation cluster for the location of an object that migrated to another already mapped) cluster.
- Compared to a mapping, following forwarders or accessing a catalog (in physical memory) are cheap operations. These operations can be used to avoid needless clusters mappings.

Our technique is based on both migration catalogs and forwarders. An object that migrates from its creation cluster is registered in a catalog associated with its new location cluster (the catalog is a table that is part of the data of the cluster) and a forwarder is left in its creation cluster.

Suppose an object *O2* is invoked and *O2* has to be located. If the creation cluster of *O2* (given by the name of *O2*) is already mapped, then we try to locate *O2* in it (this is the default case); if *O2* migrated to another cluster, a forwarder is found in the creation cluster and gives the actual location. If the creation cluster of *O2* is not already mapped, then the catalogs of the already mapped clusters in the current context are used to check if *O2* migrated to one of these clusters (if so, we avoid the mapping of the creation cluster). If not, the creation cluster of object *O2* is mapped and *O2* is searched in it.

With this strategy, we never map a cluster for a migrated object that resides in an already mapped cluster. We only pay for a lookup in the migration catalogs when a cluster mapping is required. Remember that the cost of a map operation is much greater than the cost of a lookup in the catalogs of the currently mapped clusters. Moreover, if clusters are judiciously managed, the probability to find the object in a catalog will increase in a large way.



O2 is recorded in C3's migrated-object catalog. Since clusters C1 and C3 are mapped, the mapped-migrated-object catalog includes C1 and C3's catalogs.

Then, if cluster C2 is mapped, the location of O2 will be performed using the forwarder in C2. If C2 is not mapped, O2 will be found in the mapped-migrated-object catalog without mapping C2.

Two remarks have to be done about our mechanism:

- Another solution could have been to check only in the migration catalog of the current cluster (i.e. the cluster of the object that requested the location) rather than in every mapped cluster. Since object invocations are not systematically interpreted in the Guide system, it was not possible to have the knowledge of the current cluster at location time .
- Our solution has the drawback that it needs to keep a forwarder to the actual location of a migrated object in its creation cluster. This means that the location process of an object is highly dependent on the object's creation cluster and its availability. This scheme should be improved by the management of reliable migration servers that register migrations for objects that need this high availability.

Analysis

We now examine how the proposed solution fulfills the requirements described above:

1) The first requirement is met since this solution avoids needless mappings when the objects to locate are gathered in the application's working-set, composed of already mapped clusters.
   This would not be the case with a solution based on simple forwaders (DEMO/MP [10] for example) since the mapping of the object creation cluster would be always necessar.
2) The location-dependent naming scheme allows a simple and fast location for objects that don't move (the object name contains the location), therefore fulfilling the second requirement.
   This would not be the case with a solution based on global location servers (Grapevine [4] for example) since each object location would require to send a request message to a server.
3) The third requirement is met for two reasons.
   First, we kept the size relatively small (64 bits) and we rejected solutions that extend identifiers size in order to include both a unique identifier and the current object location for migration purpose (the first prototype of Guide [2] for example). It would be both space (disk) and time (CPU) consuming.
   Second, we manage global identifiers that always refer to the same object anywhere in the distributed system. Therefore, we avoid identifiers translations that would have to be performed with local identifiers when they are made available to user programs (like in Amadeus [5] for example).

More comparison elements are given in [9].

4 Evaluation

The Guide system has been operational for 2 years and is still being used for further experiments. It is running on a set of Bull-Zenith P.C. 486 (33 MHz) machines connected by a 10 Mb/s Ethernet network.

For the evaluation of our design, we measured the cost of the basic functions involved in its implementation, and verified our initial assumptions about these functions were coherent:

- A mapping operation is expensive,
- Compared to mapping, following forwarders or accessing a catalog (in physical memory) are cheap operations.

Case	Operation	Elapsed time
(1)	cluster mapping	8 ms
(2)	lookup in a global migration catalog	7N+33 µs
(3)	direct object location in a mapped cluster	22 µs
(4)	object location in a mapped cluster with a forwarder	43 µs

In this table, (1) gives the cost of a mapping operation when the cluster is managed by a server local to the machine that requested the mapping; a mapping would be much more expensive with a remote server. (2) is the cost of a lookup in the global migration catalog, the parameter N being the number of clusters mapped in the current context. (3) and (4) are the respective costs of an object location in a cluster when the cluster is already mapped, and when a forwarder (in memory) has to be used.

From this table, we can see that there's a gap between the cost of cluster mapping and the cost of operations (2), (3) and (4). Therefore, the previous assumptions are satisfied.

We also experimented with a simple application based on the Cattell benchmark [6]. This application implements a travel in a graph in which each node has three children nodes randomly chosen among 1000 nodes. The travel is done down to seven levels (a total of 3280 nodes are visited).

Instead of creating these objects in a single cluster, some of the nodes were created in N additional clusters (N varying between 0 and 20. We then used object migration to gather objects in one cluster. We obtained the following results:

N	0	5	10	15	20
Before migration	270 ms	320 ms	400 ms	450 ms	520 ms
After migration	-	270 ms	270 ms	270 ms	270 ms

While the travel cost increases with the number of clusters when migration is not used, using migration allows to keep this cost close to the ideal cost (when a unique cluster is used). The overhead of our mechanism is not visible, since it is two orders of magnitude less than the mapping cost.

5 Conclusion

In this paper, we have described our experience in designing and implementing a mechanism for object migration in a distributed persistent store.

The purpose for providing a mechanism for object migration between clusters is to provide users the ability to gather co-related objects in clusters, in order to improve object management at the system level. The management of clusters as some kinds of working-sets allows to reduce IO tranfers, page faults and mapping operations.

In most of the cases, the implementation of object migration is at the expense of object location (at least for migrated object). The main motivation for the design of our scheme was to provide this facility while preserving the efficiency of the location process.

In the current implementation of the Guide system, object migration is based on migration catalogs stored in clusters and a forwarder managed in the creation cluster. This strategy allows to save needless mappings in most cases, and consequently to gain on further locations.

Bibliography

[1] M.J. Acetta, R. Baron, W. Bolowsky, D. Golub, R. Rashid, A. Tevanian, M. Young, “Mach: a new kernel foundation for Unix Development”, *USENIX 1986 Summer Conference,* July 1986.

[2] R. Balter, J. Bernadat, D. Decouchant, A. Duda, A. Freyssinet, S. Krakowiak, M. Meysembourg, P. Le Dot, H. Nguyen Van, E. Paire, M. Riveill, C. Roisin, X. Rousset de Pina, R. Scioville, G. Vandôme, “Architecture and implementation of Guide, an object-oriented distributed system”, *Computing Systems,* 4(1), Winter 1991.

[3] V. Benzaken, C. Delobel, “Dynamic Clustering Strategies in the O2 Object-Oriented Database System”, *4th International Workshop on Persistent Object Systems: Design, Implementation and Use,* September 1990.

[4] A. Birrell, R. Levin, R. Needham, M. Schroeder, “Grapevine: An Exercise in Distributed Computing”, *Communications of the ACM,* Vol 25, April 1982.

[5] V. Cahill, S. Baker, C. Horn, G. Starovic, “The Amadeus GRT - Generic Runtime Support for Distributed Persistent Programming”, *8th ACM Conference on Object-Oriented Systems, Languages and Applications (OOPSLA),* October 1993.

[6] R. Cattell, J. Skeen, “Object Operation Benchmark”, *ACM Transactions on Database Systems,* 17(1), March 1992.

[7] P.Y. Chevalier, A. Freyssinet, D. Hagimont, S. Krakowiak, S. Lacourte, X. Rousset de Pina, “Experience with Shared Object Support in the Guide System”, *4th Symposium on Experiences with Distributed and Multiprocessor Systems (SEDMS), San Diego,* September 1993.

[8] D. Hagimont, P.Y. Chevalier, A. Freyssinet, S. Krakowiak, S. Lacourte, J. Mossière, X. Rousset de Pina, “Persistent Shared Object Support in the Guide System: Evaluation & Related Work”, *9th ACM Conference on Object-Oriented Systems, Languages and Applications (OOPSLA),* Octobre 1994.

[9] D. Hagimont, P.Y. Chevalier, J. Mossière, X. Rousset de Pina, “Object migration in the Guide System”, *Technical Report, extended version of this paper,* May1995.

[10] M. Powell, B. Miller, “process Migration in DEMO/MP”, *9th ACM Symposium on Operating Systems Principles,* October 1983.

[11] E.I. Organick, *The Multics system: an examination of its structure,* MIT Press, 1972.