The Role of Network Traffic Statistics in Devising Object Migration Policies[*]

Ivan Marsic and Kanth S.L. Jonnalagadda
CAIP Center, Rutgers University
Piscataway, NJ 08855–1390
{marsic,kanth}@caip.rutgers.edu

Abstract

To achieve the goal of improving performance, reliability, and concurrency control in our real–time groupware system DISCIPLE, *we are designing a knowledge–based system for resource control and management. An important part of our strategy is fine–grained resource control in terms of managing object location. Objects migrate to different hosts according to migration policies to accomplish their tasks. Our goal is to develop a mechanism for run–time learning of migration policies which is transparent to the user as well as to the application programmer. This paper addresses the role of network traffic statistics in learning object migration policies for real–time groupware applications.*

1 The DISCIPLE Real–Time Groupware System

As a part of the real–time collaborative groupware project DISCIPLE, we are investigating the role of network traffic statistics in resource management. The DISCIPLE system at CAIP is an advanced groupware design that enables multiple participants, using networked computers at different locations, to collaboratively access, manipulate, analyze, and evaluate multimedia data. The system uses knowledge–based planning and learning strategies for discerning the communication needs of participants and computational task demands.

The goal of DISCIPLE is to provide software reliability, quality of service, and concurrency control in a distributed environment. To achieve this goal, we are developing knowledge–based systems for management and control of various system resources. One such knowledge–based system manages and controls object location in a distributed environment. This system moderates the work of an object communication infrastructure (CORBA–compliant Object Request Broker [5]) and its functioning would ideally be transparent to the user and to the application programmer (Figure 1).

[*]The research reported here was supported by the DARPA Contract No. N66001–96–C–8510 and by the Center for Computer Aids for Industrial Productivity (CAIP). The CAIP Center is supported by the New Jersey Commission on Science and Technology and the Center's Industrial Members.

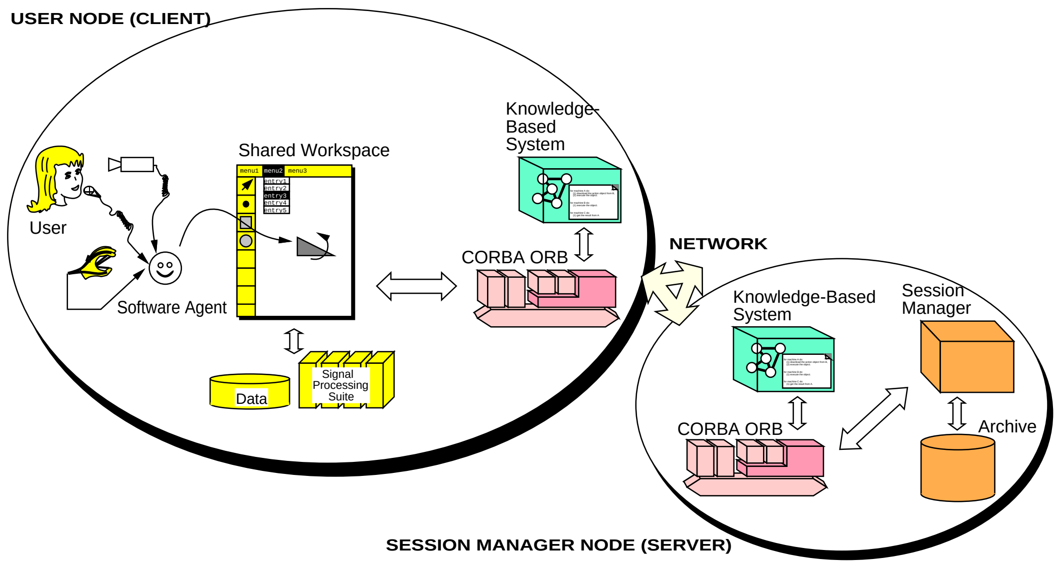

Figure 1: The software architecture of the DISCIPLE system. There is one server and several client nodes. The knowledge–based system works transparently to the user and interacts only with the ORB.

2 Object Migration

CORBA [5] provides remote calls by reference but does not provide calls by value. Applications which receive object references normally need to ask back for object attributes and services, and these questions translate into network traffic. We want to avoid this traffic (in some cases) by effectively allowing "call by value" through object migration. There are other benefits of object migration [1, 2], but the present paper addresses only network traffic reduction. However, as we will see, not all remote calls should be accomplished by value. The *migration policy* makes the decision about when and where to migrate an object.

Migration policies in existing systems are predefined, and there have been a few attempts in making flexible migration policies that can change at run–time [3]. The novelty of our approach is that we are addressing the case where the software architecture of an application is not known in advance. The migration policy is thus derived from an application's behavior rather than from its software architecture. The reason is that we want to free the application programmer from giving lengthy descriptions of the application architecture.

In the ideal case, the functioning of the knowledge–based control system would be completely transparent to the user or the application programmer. It would appear as a "black box" which can be attached to any collaborative system; the black box would first undergo a learning phase, and then act as an autonomous system which moderates the work of the object communication infrastructure.

In a less ideal case, the user (or application programmer) would need to fill out a set of forms to describe the classes of collaborative tasks and computing resources in order to put the domain knowledge into the system. Besides, complete transparency is sometimes undesirable since the user needs to be involved in cost–per–service issues. Our current goal is to explore the feasibility of the "black box" approach.

The form of object migration proposed here does not move object resources of an object from one host to another. Instead, it clones the object on all participants' hosts, while keeping it on the source host. This type of migration is suited for synchronous groupware where we need to multicast multiple replicas of an object, rather than migrating it sequentially from one host to another.

3 Knowledge Sources

Our goal is to make a decision making system, which will decide on making remote calls to an object which lives in its "birthplace" (remote address space), as opposed to copying an object to the local address space of the caller. The decision should be based on the knowledge that will be gathered from the following sources:

1. A user describes computing and network resources through specially designed forms.
2. An application programmer describes a class of tasks that will be performed with a given collaborative application, or even more precisely, a class of commands to be performed in a session. The application programmer also describes the object's characteristics in terms of the amount of data, complexity of the operations performed by the object's methods, etc.
3. A separate unit observes the amount of network traffic and other dynamic conditions relevant in making the decision about object migration.

Here we focus only on the last mechanism. This mechanism would be redundant if the software architecture were always known in advance. However, we would like to put this module in an ORB, and since the ORB should not be application–aware, we need some other means.

Real–time groupware mostly exchanges messages about user commands performed in the user's own workspace (Figure 2). Each editor receives a list of peers from the session manager and sends them commands to be performed. Command objects are good candidates for migration. On the other hand, Editor objects should not be migrated since each Editor is assigned to a particular user and performs its task on the user's host.

We plan to conduct several collaborative sessions, measure network traffic, and analyze statistics of network traffic. In order to do this, we have implemented a multiuser graphics editor in the Java programming language and put a "probe" in Sun Microsystems' Portable Java ORB [4]. The probe intercepts each remote call and records its absolute time, time to execute the call, amount of data transmitted, and the method being called. It also records the object's lifetime.

The following scenario is designed for an initial measurement of network traffic in collaborative sessions. Five geographically separated users are asked to draw a complex technical drawing, say a new design for an automobile. Each user is assigned his/her part of the entire design. Since the current implementation has a single public workspace, all users draw simultaneously in the public workspace. At the end they have to assemble the parts together into a single drawing. The users will be encouraged to complete the whole process in the shortest possible amount of time. The scenario will be augmented as more features become available in the multiuser graphics editor. A more complex scenario would include negotiating until they reach agreement on the best design.

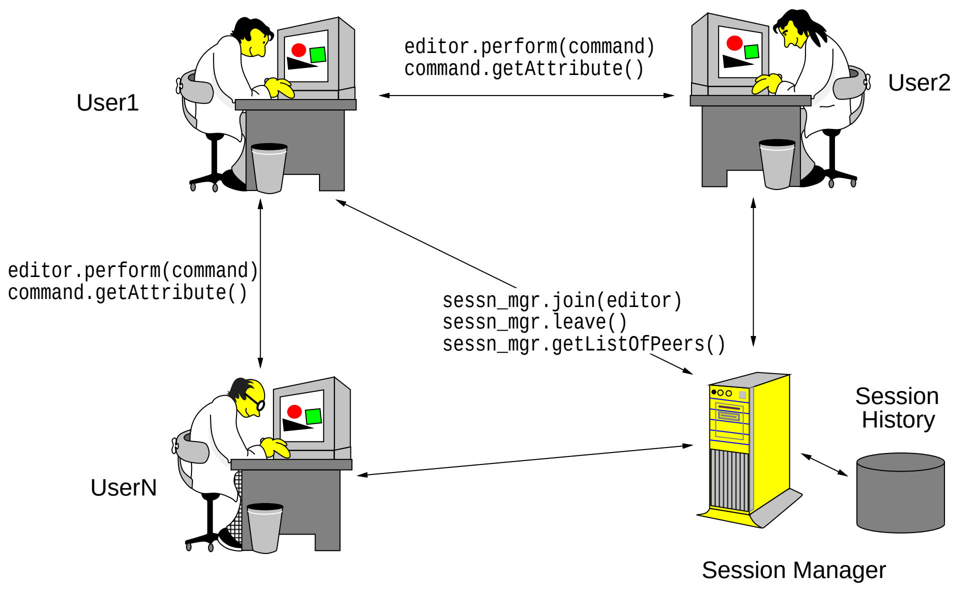

Figure 2: The interactions in the DISCIPLE system. Shown are examples of remote calls. Each user has an Editor object, and every user's action generates a Command object.

4 Analysis of the Network Traffic

Once we gather the data on network traffic during collaborative sessions, we plan to perform statistical analysis and look for patterns which will be used in devising migration policies. Some relevant parameters are listed in [2]. Here we are primarily interested in parameters relevant to real–time synchronous collaboration.

We plan to observe the following parameters in order to make the object migration decision:

- mutable vs. immutable object;
- momentary vs. persistent object (length of object's lifetime);
- distribution of method calls over the object's lifetime;
- distribution of calls on a particular object or object's methods: uniform vs. in bursts;
- amount of data transmitted per each call;
- direction of data flow (in, out, inout);
- inner calls homogeneous (methods of this object) vs. heterogeneous (methods of other objects).

The significance of these parameters is the following:

For an immutable object, it is better to err on the side of over–transmission, rather than risk having more than one remote call of the object's methods.

If an object has a short lifetime, it is likely that it will not accumulate any information, neither in its birthplace nor anywhere else, i.e., it is immutable. If an object will live permanently in many locations the problem of keeping all copies up to date has to be solved.

If the method (service) calls appear only in the beginning of the lifetime, it may signal that the object is a single–purpose object. Multipurpose objects have long–term goals and interaction with the environment is dispersed over time.

The direction of data flow will influence the decision of whether to move a client object to the server object's address space or vice versa.

An important parameter is whether an object just provides services or also asks for services and where its server objects are located. Relocating the object to another address space may generate more remote calls (e.g., back to its origin place) than the number of calls needed if the object stayed in its original place.

5 Conclusions and Future Work

The goal of this work is to perform measurements of network traffic during several types of collaborative sessions, analyze the data and extract patterns. The patterns will then be used by the knowledge–based system for resource management to reduce the network traffic and thus improve application reliability and quality of service.

After the behavioral patterns are recognized, we will investigate the possibility of establishing a direct relationship between the behavioral and architectural patterns and will use this knowledge in devising migration policies. For example, from an object's behavior we hope to be able to distinguish a Command object from an Editor object in Figure 2. If the system acquires this knowledge, it would decide to migrate the Command object, and not to migrate the Editor object.

Once the system is ready, we plan to experiment with copying/moving objects and observe how the object migration changes system performance.

Acknowledgments

The authors had many discussions with Drs. J.L. Flanagan, C.A. Kulikowski, P. Meer, and L. Gong that have shaped the work presented here. S. Sundaram, Y. He, G. Shankavaram, and S. Veeramani helped develop the multiuser graphics editor used in the experiments.

References

[1] A. Ciampolini, A. Corradi, L. Leonardi, and F. Zambonelli. The Benefits of Migration in a Parallel Objects Environment. In *Proceedings of the EUROMICRO Workshop on Parallel and Distributed Processing*, Malaga (E), January 1994.

[2] E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine–Grained Mobility in the Emerald System. *ACM Transactions on Computer Systems*, 6(1):109–133, February 1988.

[3] W. Lux. Adaptable Object Migration: Concept and Implementation. *ACM Operating Systems Review*, 29(2):54–69, April 1995.

[4] Sun Microsystems, Inc. Java IDL. Mountain View, CA, 1996. available at: `http://splash.javasoft.com/JavaIDL/pages/index.html`.

[5] The Object Management Group. The Common Object Request Broker: Architecture and Specification. Technical Report 96-03-04, Object Management Group, Inc., Framingham, MA, July 1995. Revision 2.0.