

# Supporting Large Scale Applications on Networks of Workstations

Robert Cooper

Ken Birman

April 12, 1989

Distributed applications on networks of workstations are an increasingly common way to satisfy computing needs. However existing mechanisms for distributed programming exhibit poor performance and reliability as application size increases. The ISIS distributed programming system is being extended to support large scale distributed applications by providing *hierarchical process groups*. The principal idea is to incorporate hierarchy in the program structure and exploit this to limit the communication and storage required in any one component of the distributed program.

## 1 Motivation

The explosive growth in the number and power of workstations has made possible a fundamentally different type of distributed computing. Whereas distributed systems have traditionally taken the form of a network interconnecting some number of application programs with centralized servers, it has recently become practical to talk about fully decentralized software, in which workstation-based applications cooperate directly without the need for separate servers. The motivation for choosing a decentralized solution is often the better price/performance of a collection of workstations compared to a single larger server, the potential of increased reliability of a distributed implementation, or organizational and social reasons. Yet, the construction of this type of software raises new and extremely difficult issues and can introduce substantial system complexity.

Over a period of several years, our group has developed a technology for writing small scale distributed applications using an approach based on *virtually synchronous process groups*. This technology takes the form of a distributed programming toolkit, called ISIS, which has now been made available to nearly 200 sites. The present version of ISIS is limited to relatively small-scale applications, containing fewer than 50 workstations. Here we discuss the extension of the basic ISIS technology to much larger systems. Our approach seeks to maintain the advantages of virtual synchrony, while controlling those costs that grow as the size of a distributed application increases.

A claimed advantage of workstation-based distributed systems is that they can be extended easily as a user's needs grow. We argue that when using conventional technologies, this advantage cannot be fully realized. On the contrary, reliability tends to drop in large systems, because the probability of component failures rises steadily with the number of components. Extensibility is thus something of an illusion: the hardware is trivially extended, but the software costs of a large system can grow substantially with size. The work presented below seeks to fulfill the potential of large distributed systems by attacking this software reliability "bottleneck".

## What are the large scale applications?

It is easy to forget that not everyone who uses a workstation is a programmer or a chip designer. Yet *outside* these areas we find some of the most demanding applications. Consider two examples:

- **Trading room systems for stock brokerage firms and banks:** A typical installation will comprise perhaps 100 to 500 trading analyst workstations which filter, process and analyze large volumes of information continuously supplied from numerous outside data feeds. Users of these systems demand surprisingly high performance, often requiring sub-second response to events detected over the data feeds.
- **Manufacturing control systems:** Hundreds of work cells distributed throughout a factory communicate with production monitoring and inventory control stations. Consistency and reliability are important here.

It is easy to imagine the number and size of applications like these growing considerably over the next few years.

## 2 The existing ISIS toolkit

As noted earlier, our previous work has led to a toolkit for building distributed applications [Birman and Joseph 87]. In this section we describe ISIS as it currently exists, but point out the problems it has with large scale systems. In the remaining sections we explain the approach we are taking to solving these problems.

### What is ISIS now?

The principal abstractions that ISIS provides the programmer are process groups and broadcasts. Groups are the primary means of addressing collections of processes in a distributed program, and they are the destinations of broadcasts. Groups are the only addressable entities which survive individual processor failures. The ISIS broadcast protocols provide guarantees about the ordering among concurrent broadcasts, and between broadcasts and other significant events such as process failures and group membership changes. These ordering properties dramatically simplify programming in ISIS by preventing many problematic execution sequences which would otherwise occur.

Various stylized ways of using groups are encoded in the ISIS toolkit, providing ready-made solutions to common subproblems of distributed programming. For instance, the *coordinator-cohort* tool uses a group to implement a reliable service. A client of such a service broadcasts its request to all members of the group, one of whose members is chosen to handle the request. This member, the *coordinator*, is monitored by the other group members, the *cohorts*, and should the coordinator fail, one of the cohorts is selected to take over as the new coordinator. When the coordinator has completed the request, the result is returned to the client, and copies of the result are broadcast to the cohorts. Other tools in the toolkit provide subdivided parallel computation, data replication, distributed mutual exclusion, and distributed transactions.

## Performance problems with large numbers of processes

As groups grow large, the methods used in the toolkit, and the underlying algorithms for maintaining group membership, become very costly. In recent versions of ISIS we have succeeded in reducing basic costs, such as for multicasting to a small group, to the limits imposed by the hardware technology and the operating system. We expect to be able to speed up multicasts even more by specializing the implementation when using networks with an effective hardware multicast facility, such as Ethernet.

Reducing the cost of communication has allowed us to support larger applications than in the past, but the key to supporting truly large scale applications is reducing the amount of communication going on, and in particular the number of destinations for typical multicasts. The real problem in the current versions of ISIS, is that all members of a process group cooperate to process most requests and events relevant to the group, and this style of programming clearly does not scale up very well.

For example, in the coordinator-cohort example above, if there are  $n$  members in the process group, a service request will involve  $2n$  messages in the absence of process failures, and will require action by all  $n$  members, even though only one member will actually perform the request. As the number of clients of the service grows, we must increase the size of the group providing the service to cope with the increased demand. Thus message traffic will grow as the square of the number of clients. Broadcasting a request to the  $n - 1$  cohorts is not completely wasted work since the cohorts provide resiliency to failure of the coordinator. However there is no practical advantage to having more than perhaps five cohorts for a request. There is no reason to increase the “resiliency” of the service to extremely large numbers, and given the increasing load imposed by ever larger broadcasts, reliability will actually decrease.

Upon group membership changes, including the failure of a group member, a broadcast is sent to the new membership of the group, since each member maintains an up-to-date membership list. As group size increases the probability of one of the members failing increases, and with it the cost of processing membership change broadcasts.

## 3 Hierarchical process groups

Our approach to handling large scale applications consists of exploiting hierarchy to bound the number of destinations for many broadcasts and to control the cost of operations involving large numbers of processes. All the processes implementing some service are made members of a single large group, but this group is organized into many resilient subgroups. The large group is used for naming purposes to identify the service, but requests are broadcast to a individual subgroups. In fact a group may have multiple levels of subgroups, but at each level the number of children of a subgroup will be bounded. The critical issues are how to represent and manage hierarchical group membership lists, and how to map large scale applications onto this structure.

### Group structure

We first define the following quantities on a group:

- *Size*: The number of member processes.

- *Resiliency*: Any communication with (or among) the group is guaranteed to occur despite *resiliency*-1 failures of group members. This means that the process initiating a broadcast must receive acknowledgements from at least *resiliency* destinations before reporting success to the caller, and critical state concerning group membership and structure must be replicated by at least *resiliency* group members.
- *Fanout*: A process may communicate directly with no more than *fanout* group members. If *fanout* < *size* then some multistage broadcast algorithm must be used.

Typically  $size \geq fanout \geq resiliency$ . A group with  $size = fanout = resiliency$  is called a *small* group. (Currently in ISIS all groups are small.) A group with  $size > fanout \geq resiliency$  is called a *large* group.

A large group comprises several subgroups of at least  $max(resiliency, fanout)$  member processes. These *leaf subgroups* are organized into a hierarchical structure of *branch subgroups*. When a process fails, or leaves the large group, only the other members of its leaf group need to be informed. If a leaf experiences total failure (all its members fail) then only the *parent* group is informed. Thus any single process failure results in a broadcast to a bounded number of other processes.

## Managing group views

A *group view* is the fundamental data structure representing a group. For a leaf group, the group view contains a list of the member processes. For a branch group, the membership is represented as a list of the immediate child groups, *not* as a list of the individual member processes. So a complete list of the processes in a large group is not explicitly stored anywhere, bounding the storage required within any single process for storing a group view.

Each member of a leaf group maintains a copy of the leaf group view. But where are the group views for branch groups stored? Having copies in each member would be too costly for a branch group with many members, and would provide an unnecessary degree of replication. Instead a new resilient group, called the *group leader* is constructed, whose function is to manage the group view. It is the leader which is informed of the total failure of one of the child subgroups, and which is responsible for splitting subgroups which have grown too large, and merging subgroups which are too small. The leader may perform group-wide application-level functions such as partitioning data or processing between subgroups.

## 4 Extensions to the ISIS toolkit

The hierarchical group abstraction is more complex than the existing ISIS primitives. Nonetheless, we have been successful in concealing this complexity at the toolkit level of ISIS, which is the level at which most users work. We are maintaining a high degree of compatibility with the previous small-scale versions of the tools, allowing many application programs to take advantage of large groups with only minor changes. In other cases a program may have to be restructured in order to scale to larger applications. We also expect to identify new patterns of usage of the ISIS primitives in larger applications, stimulating the addition of new tools to the toolkit. Extending the toolkit to use large groups will demonstrate how useful and usable large groups are, since the toolkit is representative of many distributed applications.

## 5 Other work

Even with the techniques described in this paper, there will remain situations in which it is necessary to communicate with all the members of a large group. For this reason we have designed a tree-structured broadcast algorithm which maps the broadcast tree onto the hierarchical group organization [Cooper and Birman]. We are also addressing the issues of group name-to-address mapping in the large scale setting, coping with network partitions, and considerations of long-distance links.

Implementation of the hierarchical version of ISIS is now underway. It is expected that a completed version will be released to our users late in 1989.

## References

- K. P. Birman and T. A. Joseph "Exploiting Virtual Synchrony in Distributed Systems" *Proc. of the 11th ACM Symposium on Operating Systems Principles*, November 1987, p. 123.
- R. Cooper and K. P. Birman "A Large Scale Atomic Broadcast Algorithm" *in preparation*.