NAG2-593
NASA
IN-61-CR
270871
148.

Enclosed is a copy of a technical report produced by the ISIS group. This report was produced under contract number NAG2-593.

Respectfully yours,

Susan Allen,
ISIS Project Secretary
(607) 255-9198

THIS REPORT IS UNCLASSIFIED AND MAY BE DISTRIBUTED WITHOUT RESTRICTION

(NASA-CR-186413) ISIS AND META PROJECTS Progress Report (Cornell Univ.) 14 p
CSCL 09B

N91-14731

Uncles
G3/61 0270871

ISIS and META Projects: Progress Report

Kenneth Birman, Robert Cooper, and Keith Marzullo*

February 22, 1990

ISIS and META are two distributed systems projects at Cornell University. The ISIS project, led by Ken Birman, has developed a new methodology, *virtual synchony*, for writing robust distributed software. This approach is directly supported by the ISIS Toolkit, a programming system that is distributed to over 300 academic and industrial sites. As the basic ISIS techniques have matured, we have focused increasingly on some of the remaining "hard problems" of reliable distributed programming. Principally these include high performance multicast, large scale applications, and wide area networks. We are also developing several interesting applications that exploit the strengths of ISIS, including an NFS-compatible replicated file system.

The META project, led by Keith Marzullo, is about distributed control in a soft real-time environment incorporating feedback. This domain encompasses examples as diverse as monitoring inventory and consumption on a factory floor, and performing load-balancing on a distributed computing system. One of the first uses of META is for *distributed application management*: the tasks of configuring a distributed program, dynamically adapting to failures, and monitoring its performance.

This article reports our recent progress and current plans. But first we begin by explaining our approach to distributed computing, a philosophy that we believe significantly distinguishes our work from that of others in the field.

*This material is adapted from a short paper presented at the Workshop on Mission Critical Operating Systems, Washington, Nov. 1989. This work was supported by the Defense Advanced Research Projects Agency (DoD) under ARPA order 6037, Contract N00140-87-C-8904 and under DARPA/NASA subcontract NAG2-593 administered by the NASA Ames Research Center. The views, opinions, and findings contained in this report are those of the authors and should not be construed as an official Department of Defense position, policy, or decision.

Network transparency: Too much of a good thing?

Users of contemporary distributed computing systems rapidly discover how similar such systems are to the timeshared machines of the 1970's: the pervasive use of "network transparency" techniques lets us largely ignore the fact of distribution. Normally, this is a desirable property. For example, the dominant distributed programming technology, remote procedure calls (RPC), permits a program running on one machine to invoke a procedure residing in some other program. Given adequate language support, an RPC interface can hide many details of message-based interaction and connection management from the user. The idea of transparency also extends to other parts of a typical distributed system. Using a file system like NFS, a program can operate on files that physically reside on a remote machine in the network using the same interface as for local files.

Complete transparency is troubling, however when one considers the many reasons that distributed computing *should* be different from non-distributed programming. Parallel computing is in many ways analogous to distributed computing. Yet, whereas the effective use of parallel machines has triggered a search for fundamentally new programming languages and methodologies, this has not happened for distributed programs. If we are building distributed systems using technologies that proved unsatisfactory in parallel settings, is it not likely that our distributed systems are making ineffective use of parallelism?

The requirements placed on a distributed application often go beyond the exploitation of concurrency. In particular, one often wishes to monitor and control a distributed computer system while it is running. Moreover, a distributed system may need to remain operational in the presense of *partial* failures. By this we mean situations where one of the machines connected to a network fails or becomes partitioned from the others, while the majority of the machines remain operational and must reconfigure and continue executing. The complementary problem also arises, of reintegrating a recovered machine into an online system.

The Isis project is based on the premise that when we pretend that a distributed system is really a timeshared system, or encourage the user to program as if his or her application were the only process running on the system, as with transactional RPC, we discard a powerful resource: the fact of distribution itself. We lose the ability to employ a set of processes in a coordinated, cooperative attack on a problem. We lose the ability to apply highly adaptive, reconfigurable solutions to applications that must

remain online in the presense of failures and recoveries. And, we make it difficult to build a distributed system that is more fault-tolerant and offers higher performance than any of its components. The Isis Toolkit, and the Meta system that we are now building on top of the Toolkit environment, represent a significant step towards addressing these sorts of issues.

The Isis Toolkit: Process groups and multicast

At the lowest level, the Isis system provides a *toolkit* of distributed programming techniques. This consists of a layer of software to assist the programmer in building distributed applications. The toolkit is very much like an extension of the operating system, although implemented without changes to the operating systems on which Isis runs.

Central to Isis is the notion of a process group. These groups are a lightweight programming construct: a single process can belong to arbitrarily many groups, and there is minimal overhead in being a member of a group. A process can dynamically join and leave groups, and groups can span multiple machines. Groups have a hierarchical namespace, much like a file system namespace, and permit flexible, location-transparent addressing.

Isis provides multicast and unicast (point-to-point) communication primitives that are easy-to-use and flexible to the demands of the programmer. A multicast can be directed to all members of a group, and zero or more will respond, depending on the needs of the particular application.

Concurrent multicasts and unicasts, dynamic group changes and failures would seem to present a very complex, even daunting, execution environment. But in Isis all these concurrent events appear to happen one-at-a-time. We call this simplifying model *virtual synchrony.*

Virtual synchrony

Virtual synchrony is a general approach to solving distributed computing problems. Derived in part from the *state machine* approach (introduced by Lamport and Schneider), virtual synchrony permits the programmer to design a distributed program for execution in a simplified environment, in which all processes appear to observe events simultaneously and therefore in the same order. Events such as multicast and detection of failures are atomic in a virtually synchronous setting: all group members receive a message or observe a failure if any does, and in the same consistent ordering. The synchronous abstraction is relaxed when the program is executed by Isis

using application-specific knowledge. ISIS has several multicast primitives that differ in the kind of ordering they enforce on concurrent events. By selecting the appropriate primitive, the programmer tells ISIS what degree of synchrony is needed for that part of their application.

Virtual synchrony permits the ISIS programmer to work in an environment where many of the aspects that render distributed computing difficult do not arise, but the resulting program runs as asynchronously (and fault-tolerantly) as may be desired, without compromising correctness. Virtual synchrony has been exploited throughout ISIS, and leads to a simple step-by-step programming style that even relatively unskilled programmers can follow. Taken together with the wide range of tools represented in the toolkit, the approach leads to a major jump in programmer productivity, and major improvements in the robustness of distributed software.

Virtual synchrony has a well-developed theory, principally through the work of Ph.D. graduate Frank Schmuck, that explains when low-cost asynchronous techniques can be used to implement virtual synchrony. More recently, we have explored the relationship between global correctness and consistency properties in distributed systems and the ordering mechanisms needed to achieve them; a technical report on this subject is listed below.

ISIS Toolkit: Problem-specific tools

Using the process group, multicast and unicast primitives, ISIS provides a variety of higher level tools that solve common subproblems in distributed computing. For example, tools are provided to:

- Manage replicated data in memory or on a disk file
- Split a computation among several machines to exploit parallelism
- Coordinate an external action such as operating independent welding units that are jointly welding an automobile body
- Synchronize concurrent actions such as when several processes share a resource that only one can use at a time
- Monitor the status of a computation, process or computer, triggering user-programmed actions should it fail
- Dynamically reconfigure to adapt after a failure or to integrate a recovered machine into an operational system, restarting services that

should run at that location and bringing them up-to-date concerning the active state of the system.

This is just a partial list. Moreover, the tools are integrated with each other in a way that makes it easy to obtain consistent behavior even when several processes must react *independently* to the same event.

Isis Version 1.3.1

The present version of the Isis toolkit can be used from C, C++, and Fortran. Common Lisp interfaces are available for several versions of this language. Isis runs on (and between) Sun, DEC, HP, Gould, NeXT and Apollo equipment, on and between several versions of Unix (including Mach, Aix, HP-UX and Unicos). Ports to DEC's VMS system and IBM's VM operating system are being considered, as is an interface to PCs running OS/2.

The Isis Toolkit is in increasingly wide use, and our group has distributed more than 300 copies of the source for Isis V1.3.1. Among the users of the current system are a number of Fortune 500 companies, several industrial research and prototyping groups, and a number of academic researchers and instructors. Applications include controlling a world-wide nuclear testban and seismic monitoring system, automating a factory-floor VLSI fabrication system, dissemination of quotes and other real-time data in brokerage settings, and CAE/CAM systems. This diverse user base has been a source of invaluable feedback.

Isis Version 2.0

Although Isis V1.3.1 has proved extremely robust, it is also sluggish and hard to scale. Isis V2.0 will soon be released, and overcomes these limitations while preserving the robustness of V1.3.1. With regard to performance, V2.0 includes a new "bypass" communication protocol suite, which permits group communication at hardware speeds and enables the application designer to introduce new multicast transport algorithms that exploit special hardware or software features, or offer special properties such as real-time delivery guarantees. This facility represents a major advance for our group, and yields multicasts that are a match for alternative approaches that lack Isis's atomicity and ordering guarantees. We feel that it overcomes the widespread concern that fault-tolerance may simply be too costly a price

to pay in "real" distributed systems. On the contrary, we now feel that developers who build on a conventional software substrate are limiting their options, working with unnecessarily complex message-at-a-time interfaces, and not even gaining a performance advantage by doing so.

With regard to scale, ISIS V2.0 has two significant extensions that respond to the most urgent needs identified by our users. One permits us to connect applications on computers that don't run ISIS to the ISIS system as *remote clients*. The interface is largely transparent to the application designer and imposes little overhead. In the initial implementation of remote clients, the remote ISIS server may introduce a common failure point for those computers that are its remote clients. We plan to increase the fault-tolerance of this mechanism by permitting a remote client to switch dynamically between ISIS servers in case of failure. Nevertheless, the current implementation is a good match for diskless workstations where a client's remote disk server machine will also act as its ISIS server process.

A second extension permits users to develop services that span wide-area networks, residing on multiple ISIS local networks and communicating infrequently and asynchronously. For example, the large-scale seismology system cited earlier uses this facility to keep track of the location of files containing signal analysis output and to transfer these files from one ISIS system to another. The long-distance circuits are set up periodically, used intensively, and then closed down to minimize communications costs.

ISIS applications

In order to exercise and evaluate ISIS, we have developed several fault-tolerant applications. For example, we have built a distributed, fault-tolerant version of the UNIX program *make*, a main-memory distributed relational database, and a multi-user spreadsheet that can be used in a cooperative manner. The first two of these applications are available as part of the ISIS V2.0 release, and the spreadsheet should be available by the end of 1990.

As part of his research, graduate student Alex Siegel has been designing and building a highly-available file system called *Deceit*. This file system is completely compatible with NFS, yet uses replication for improved response time, higher availability and better scaling. Additionally, Deceit allows the clients to specify properties of individual files in order to tune access to the file. Currently, Deceit is running in a prototype form. It outperforms NFS for many operations (notably read and write), and equals NFS for almost all others operations.

Using Isis for large scale applications

Many systems that support process groups assume that any single application will use at most one group. Most Isis applications employ several groups, and many use large hierarchically structured groups. This is explained by two factors. First, the trend toward modularity and object-oriented programming in distributed systems leads many designers to think of a process group as a form of distributed object. Even if the components of the group are coded in different languages or have differing functionality, this proves to be a simplifying and powerful structuring methodology. Since a single process may make use of several services, each implemented using such a process group, it is not uncommon for a single process to belong to many groups.

A second factor is concerned with scale: Isis users are building surprisingly large distributed applications, with groups which contain many processes. It is unusual, and unwise, to multicast to the entire membership of such a large group, except where widespread dissemination of information is fundamental to the application itself. (This might be the case for a distributed network news application for example.) Designers of large systems are thus lead to use a hierarchical structure in which a large group contains a number of smaller groups. These smaller groups are chosen in such a way that most multicasts are destined to just one or two such groups.

Responding to these needs, Robert Cooper has designed a suite of hierarchical process group tools for Isis. These extend the basic tools to operate transparently on hierarchical groups, while augmenting the system with mechanisms for reliably broadcasting to a large group that is maintained hierarchically. A prototype of this facility is nearing completion.

High performance multicast

Looking to the future, we are exploring a number of theoretical and practical topics at the Toolkit level. The practical ones include adding a better security mechanism to the system, extending Isis to support real-time protocols and other special-purpose protocols, and integrating the system into environments with parallel processors and extremely high speed communications protocols. Mechanisms for exploiting new operating systems, such as Chorus and Mach, also represent an appealing direction. Graduate student Patrick Stephenson has developed a class of extremely high performance

multicast transport protocols suitable for use in the new ISIS system, and we plan to combine these in conjunction with an ISIS service knowledgeable about the communication topology of a local area network to develop a suite of protocols that adapt themselves to the environment, for instance exploiting Ethernet multicast when possible. We also hope to scale the size of local area network on which ISIS may run from the current limit of about 64 nodes up to hundreds or thousands of nodes, by introducing hierarchy at the lowest levels of the Toolkit. The Toolkit architecture now seems fairly stable, and is unlikely to change in visible ways as these extensions are made.

The META system

We mentioned above that ISIS involves software at several levels. The toolkit is a low-level technology, for use by programmers who actually code distributed programs. The META system is a collection of higher level tools that aid in gluing together distributed programs into a reliable and adaptive distributed system.

At the core of META is a set of routines that support building *reliable reactive systems*, such as factory floor management systems, process control systems, and the control aspects of distributed applications. This level provides a platform that can be used to monitor and control a distributed system. Supported at this level are routines for instrumenting a distributed system, monitoring for (perhaps complex) real-time conditions, and triggering actions on the controlled system.

There are two interfaces to the sensor/actuator platform. The low–level interface permits users to define *raw* sensors and actuators, namely routines (or variables) in user programs that can be queried to obtain the current sensor value. At this level, META also supports an entity–relationship database model describing sensors, their real–time properties, and the relationships between them. Some raw sensors are predefined, such as the ones giving the load on a computer or a process, while others can be defined dynamically, such as the length of a job queue maintained by some software component of a larger system. Also supported are mechanisms for composing multiple raw sensors into an abstract sensor. This is used to define such properties as the average over a set of sensors, as well as to support sensors tolerant of certain classes of failures.

The high–level interface to META is concerned with querying and monitoring sensors. This supports a Prolog–like query language for identifying individual sensors and sets of sensors satisfying user–defined predicates, as

well as a trigger language whereby the user can monitor for events of interest, triggering appropriate actions when the event is detected. Both of these interfaces are provided at the language level.

Built on the basic platform are a number of facilities for actually managing distributed applications. These help manage the allocation of system resources, control the initiation, migration, and termination of programs, and monitor the performance of the system. One interface to this distributed system manager is accessed through a powerful graphical interface: using this facility, one can achieve sophisticated fault-tolerant behavior without writing a line of code.

Parts of META are currently available, while other parts are still being built. The META platform of sensors and actuators facilities, built by graduate student Mark Wood, is provided in ISIS V2.0, and the design of our sensor query language is complete; an implementation is expected to be finished during 1990. Visitor Robbert Van Renesse has developed, on top of META, a distributed application management program called GARP, which is a prototype graphical monitoring and control program. This system will also be released sometime in 1990.

Support for the software

Although ISIS is an academic project, it has acquired an increasingly large commercial following. At present, all of the academically developed ISIS software is freely available in the public domain. We have made a major effort to provide high quality support for this software, and believe we have an excellent record of responsiveness—and of success in tracking down and fixing bugs. On the other hand, this sort of commercial responsiveness is making it increasingly difficult to maintain an active research program.

To address this problem, we have formed a company, ISIS Distributed Systems Incorporated, which is offering commercial services to companies in need of customized software or consulting. Starting in 1990, these will include support for the ISIS Toolkit and products that extend the Toolkit to respond to some of the specialized demands of our user group. For example, IDS is now building a collection of general purpose software tools for one client whose application demands certain specialized components that META currently lacks. In this particular case, the resulting software will eventually enter the ISIS public distributions. However, IDS is also engaged in proprietary software development, and is intended to operate as an increasingly autonomous commercial operation, freeing our research group to

focus on research.

Obtaining ISIS

To obtain information about ISIS, or a copy of the current software distribution, write to: The ISIS Project, Department of Computer Science, 4105 Upson Hall, Cornell University, NY 14853 (607-255-9198), or send electronic mail to isis@cs.cornell.edu. The group also maintains a mailing list to which announcements of all new papers are sent.

References

[1] K. Birman, R. Cooper, T. Joseph, K. Kane, and F. Schmuck. *ISIS V1.3 — A Distributed Programming Environment User's Guide and Reference Manual.* Department of Computer Science, Cornell University, June 1989.

This manual documents the system interface supported by ISIS V1.3.1 and is being extensively revised to cover ISIS V2.0. The revised manual should be available in late March, 1990.

[2] K. Birman, R. Cooper, K. Marzullo, and M. Wood. Tools for distributed application management.

Describes the basic architecture of the META system as used for distributed monitoring and control. In preparation, 1990.

[3] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the Proceedings of the Eleventh Symposium on Operating System Principles*, pages 123–138. ACM SIGOPS, 1987.

Discusses the idea of virtual synchrony and describes its application to building the ISIS Toolkit.

[4] K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(2):47–76, February 1987.

Discusses the multicast protocols on which ISIS V1.3.1 is based. These are sometimes used in ISIS V2.0, but have been superceeded by the bypass protocol suite for the most common types of communication.

[5] K. Birman and T. Joseph. Chapters 14 and 15. In S. Mullender, editor, *Distributed Systems*, volume 331 of *Lecture Notes on Computer Science*, pages 293–367. Springer-Verlag, 1989.

These two chapters discuss the idea of virtual synchrony in the context of a broad-based review of techniques for building distributed systems.

[6] K. Birman and K. Marzullo. The role of order in distributed systems. Technical Report TR 89–1001, Department of Computer Science, Cornell University, March 1989.

Concerns the relationship between what are are intuitively thought of as "consistency" properties in distributed computing, and the role of ordering mechanisms in achieving them. An early version is available as a technical report, currently being revised.

[7] K. Birman and R. Van Renesse. Robustness in distributed systems.

Examines the fundamental requirements that any fault-tolerant distributed system must fulfill, and presents universally applicable paradigms for achieving them. In preparation, 1990.

[8] K. Birman, A. Schiper, and P. Stephenson. Fast causal multicast. Technical Report TR 90–???, Department of Computer Science, Cornell University, March 1990.

Discusses a new suite of multicast protocols that "bypass" the basic ISIS system so as to achieve extremely high performance. Submitted to ACM TOCS, available as a preprint.

[9] R. Cooper and K. Birman. Supporting large scale applications on networks of workstations. In *Proceedings of 2nd Workshop on Workstation Operating Systems*, pages 25–28, Pacific Grove, CA, September 1989. IEEE Computer Society Press, Washington D.C., Order No. 2003.

Describes the design of the hierarchical process group facility.

[10] R. Cooper and K. Birman. Issues of scale in the ISIS system.

Discusses the role of large process groups in ISIS applications and the hierarchical mechanisms that are being added to ISIS in support of such groups. In preparation, 1990.

[11] T. Joseph and K. Birman. Low cost management of replicated data in distributed systems. *ACM Transactions on Computer Systems*, 6(2):54–70, May 1988.

This is a somewhat outdated paper discussing the basic ISIS replication technique but in a database transaction context. Our ultimate implementation of the method was much simplified when it was removed from this context. However, we continue to believe that the method presented in this paper dominates most data replication techniques known in the database community.

[12] K. Kane and K. Birman. Log-based recovery in object-oriented distributed systems. Technical Report TR 88–949, Department of Computer Science, Cornell University, November 1988.

Discusses the problem of recovery in object-oriented systems using asynchronous logging. An extended version is also available; it was presented to Cornell as the first author's doctoral dissertation. Submitted to ACM TODS, available as a preprint.

[13] M. Makpangou and K. Birman. Designing application software in wide area networking settings.

Discusses the development of distributed software for wide-area networks using a new long-haul spooling facility that the authors implemented as part of ISIS V2.0. In preparation, 1990.

[14] K. Marzullo. Implementing fault-tolerant sensors. Technical Report TR 89–997, Cornell University, May 1989.

Presents a method for building programs that can tolerate failures of sensors. The META system uses the techniques described in this paper in order to tolerate failures of continuous-valued sensors.

[15] K. Marzullo and P. Chew. Efficient algorithms for masking sensor failures.

Generalizes the material presented in [14] to a wider class of sensors. In preparation, 1990.

[16] K. Marzullo and F. Schmuck. Supplying high availability with a standard network file system. In *Proceedings of the Eighth International Conference on Distributed Computing Systems*, pages 447–455. IEEE Computer Society, June 1988.

Describes an early version of a highly-available file system (RNFS) that was built using ISIS. Experience with this project led to the more ambituous file system project described in [20].

[17] Keith Marzullo and Mark Wood. Tools for reliable reactive systems.

Describes the fundamental problems of sampling sensors and triggering actuators in fault-tolerant distributed settings. This material represents the theory underlying the META. system. In preparation, 1990.

[18] R. Van Renesse. The Garp management system.

Describes a graphical interface tool for building distributed application management software. In preparation, 1990.

[19] F. Schmuck. *The Use of Efficient Multicast Protocols in Asynchronous Distributed Systems.* PhD thesis, Cornell University, Department of Computer Science, August 1988.

Concerned with the theory of virtual synchrony. It develops a set of results showing when asynchronous casually ordered multicast protocols can be used in implementing a distributed system.

[20] A. Siegel, K. Birman, and K. Marzullo. Deceit: A flexible distributed file system. Technical Report TR 89-1042, Department of Computer Science, Cornell University, November 1989.

Describes the design and prototype implementation of the Deceit file system, which mimics an NFS interface but offers substantially extended functionality and fault-tolerance.