The following paper was originally published in the
Proceedings of the USENIX Windows NT Workshop
Seattle, Washington, August 1997

Moving the Ensemble Communication System to NT and Wolfpack

K. Birman, W. Vogels, K. Guo, M. Hayden, T. Hickey,
R. Friedman, R. van Renesse, Al. Vaysburd
Dept. of Computer Science, Cornell University
S. Maffeis, Olsen & Associates

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

Moving the Ensemble Communication System to NT and Wolfpack[†]

K. Birman W. Vogels K. Guo M. Hayden
T. Hickey R. Friedman R. van Renesse Al. Vaysburd

Dept. of Computer Science[‡]
Cornell University

S. Maffeis
Olsen & Associates

Abstract

Cornell University has developed a group communications and membership management tool, called the Ensemble system, which provides the basis for introducing guarantees such as reliability, high availability, fault-tolerance, consistency, security and real-time responsiveness into applications that run on clustered parallel computers or high speed networks. Ensemble tools are flexible, extremely transparent, and achieve high performance. Our development started under Unix in 1995, but by 1996 had enlarged to include NT as a primary target. This paper reviews Ensemble and then discusses the technical issues that arose when repositioning it to fit naturally and perform well under NT.

1 Introduction

Our research seeks to develop a new generation of groupware communication tools for modern networked environments, The Ensemble system offers support to the application developer who wishes to introduce guarantees such as reliability, high availability, fault-tolerance, consistency, security and real-time responsiveness into network applications. Although our project started on Unix platforms, a recent emphasis has been on the integration of the Ensemble tools into NT platforms and high-availability clusters of NT-servers such as those managed by Wolfpack.

The key to our work is to focus on what we call *process group communication structures* that arise in many real-world settings involving cluster-style computing, scaleable servers, groupware and conferencing, distributed systems management, and fault-tolerance. A process group is just a collection of processes running on some set of computers in a network. Our work uses process groups to support execution guarantees (for example, that a request to a critical server will be done even if a failure occurs, or will be completed within a real-time deadline), replicated data or computing, distributed coordination or control, and so forth. The emphasis of our effort is on the underlying communication systems support for this model, on simplifying and standardizing the interfaces within our support environment, and on making the model as transparent (hence, easy to use) as possible.

2 Isis, Horus, and Electra

Over the past 12 years, our project has developed a series of three related software systems. All solve the group communication problem, but they differ in focus. Our first was called the Isis Toolkit, and was developed at Cornell during the period 1985-1990. Isis was commercialized by Isis Distributed Systems (currently a division of Stratus Computer), and found success in settings like stock markets, air traffic control systems, electric power plant monitoring and control, factory-floor automation in VLSI fabrication plants, telecommunications systems, and banking/brokerage systems.

By 1990, we had encountered a number of potential Isis applications that pushed the system to its limits. These included very large-scale applications (with hundreds of participating computers) and cluster-styled systems, like the ones that Wolfpack is designed to support. Such systems often require more than the basic software fault-tolerance solutions offered by Isis. For example, many cluster-based systems require strong real-time

[†] Supported, in part, by DARPA ITO under ARPA/ONR contract N0014-96-1-10014, and in part by grants from Siemens, Intel and Microsoft.

[‡] Visit http://www.cs.cornell.edu/Info/Projects/Horus for information on how to contact the authors. Our software is available for general use at no fee; details on downloading and running it can be found on the web page.

guarantees as part of the fault-tolerance architecture. As an extreme case, a cluster computer used in an SS7 telephone switching system (an "IN coprocessor") must handle thousands or tens of thousands of call requests per second, guarantee 100ms response times for each event even as nodes fail and restart, and exhibit at most 3-secs downtime per year. Such requirements go beyond what Isis was able to provide.

We built the Horus system as a response to these needs. Developed over a 5-year period that started in 1991, Horus supports Isis-like functionality, but was designed to match its guarantees to the real needs of the application. We use this feature to tune Horus, so that it will give the best possible performance consistent with the reliability requirements of a specific use. The architecture is a modular one based on layered protocols. The protocol stack supporting a particular group communication application is "plugged together" at runtime. By hiding Horus behind standard interfaces, we obtained a good degree of transparency: this tactic allows us to slip group-based mechanisms into applications that were not designed with group communication as an explicit goal. Moreover, Horus supports an object-oriented interface to the CORBA architecture, which we call Electra.

Horus pushed well beyond the limits of Isis. For example, we used Horus to emulate an SS7 telephone switching coprocessor on a cluster of Unix systems. Our solution handled 20,000 telephone requests per second within the 100ms deadline, dropping less than 1% of calls (randomly) even if a failure occurred under peak load. For multicast applications over ATM networks, Horus achieved 85us end-to-end latency, and sustained throughputs of 85,000 1-byte multicasts per second.

As Horus matured, however, we encountered issues that lead to a complete reimplementation of the system. We are calling this most recent system *Ensemble*. In contrast to Horus and Isis, which were Unix-oriented, Ensemble is intended to run on NT as well as Unix and Linux, and is increasingly NT-centric in design. The body of the present paper focuses on issues raised by this shift in target platform.

3 The Ensemble System

Our key reason for moving to Ensemble concerns the method used to optimize Horus protocol stacks for good performance. It is well known that when protocols are built in a modular, layered manner, as in Unix streams or object-oriented communication architectures, the strict modularity of the layers can introduce substantial overhead. In Horus, we encountered this problem, and overcame it by developing a method for optimizing Horus protocol layers to drastically compress message headers and also shorten the critical path for most multicasts. When we applied these techniques to Horus, however, they increased the complexity of the system, reducing the benefits of modularity and flexibility.

Recoding Ensemble in OCaml, an object-oriented dialect of the ML language, made it possible to use a powerful theorem prover called Nuprl to do these sorts of transformations automatically. We are experimenting with using Nuprl to automatically generate highly optimized protocol stack that exhibits the low latencies and high throughputs that previously required a laborious hand-optimization. Although even compiled OCaml executes more slowly than C, the new protocols in Ensemble are currently faster than similar protocols in Horus, on the same hardware. Moreover, we are experimenting with using Nuprl to verify that Ensemble correctly implements critical protocols such as the ones that synchronize the reporting of group membership changes with the delivery of messages. Such formal proofs will be a valuable adjunct to exhaustive testing or other ad-hoc methods of convincing ourselves that these complex protocols work correctly. In the future, we may be able to gradually prove the correctness of more and more of Ensemble as a whole.

Users of Ensemble treat it as a collection of tools and communication primitives. Our users typically code applications in C, C++, Tcl/Tk, Java, SmallTalk or ML. They access Ensemble through "toolkits" that include libraries for replicating key portions of an application to make it fault-tolerant, replicating data for rapid access or parallelism, doing coordination and synchronization when multiple systems cooperate on a task, providing membership management for tracking the participants in an application, and automatically setting security keys. Ensemble is already integrated with Kerberos and PGP and we are now looking at support for other security architectures the RSA version of Kerberos, SSL, Fortezza, and IPSEC.

Like Horus, Ensemble can be configured at runtime to match the needs of the application and to exploit the properties of the network on which it is running. If desired, one application can run over several multicast groups simultaneously, configuring each to offer different properties. For example, a multimedia conferencing system could configure one multicast group for video, one for audio, and one for control. Each would employ

a protocol matched to the reliability and security needs for the kind of data it handles.

4 Porting Ensemble to NT.

Porting Ensemble to NT raised a number of issues, the first of which stems from our use of OCaml. OCaml is an object-oriented dialect of the ML language, and is structured to encapsulate portability issues in its runtime environment. OCaml can be used either as a platform independent bytecode interpreter, or a generator of native assembly code. Ocaml's support libraries are written in C, and the overall system exhibits only a slightly degraded performance when compared to implementations in to other high-performance languages.

Cornell researchers extended the OCaml runtime under Win32 with an interface to the communication subsystem (winsock). This proved difficult because of the need to retain compatibility with Unix. For example, WIN32 file and socket descriptors represent different objects and cannot be "mixed". In Unix, however, all types of file descriptors are considered to be of the same type. Thus, where Ocaml's runtime system previously could use a single *select* system call applied to all open file descriptors, the Win32 interface required a more complicated structure to treat the various file types separately but concurrently.

This said, the majority of the porting effort went into constructing a build environment that would work for Unix as well as Win32. The Win32 tools have different semantics or are severely crippled (nmake), but we decided not make use of ported versions of the Unix tools under NT to avoid a non-standard build process. The Ensemble source code is now maintained under Unix, but during the checkout process scripts are used to produce Win32 make and dependency files.

5 Matching the NT programming Model.

The de facto standard for programming open NT applications is the Component Object Model (COM), which was recently extended to take the distribution of objects into account (DCOM). Under NT, Ensemble provides its functionality through a collection of COM interfaces. Using these interfaces various levels of abstractions are exposed to the programmer, giving the developer full control at whatever level of detail is needed to implement the system under construction. Some of these are:

- *A low-level, event driven interface*. Internally Ensemble is a pure event driven system, and a subset of these events is exposed through a connectable COM interface. This allows higher level programming toolkits to be implemented using collections (groups) of COM objects.

- *A high level, object oriented interface*. We built a toolkit, called Maestro, which extends Ensemble with a collection of C++ classes that implement ADT's for essential types (endpoint ID's, messages, error handlers, etc), group-member/client-server abstractions and state-transfer for server objects. These types and their interfaces are exposed to COM programmers through a COM interface.

- *A highly available RPC interface*. Ensemble provides an RPC system that allows clients to access high-available objects in a transparent, fault-tolerant fashion. If the connection to an instance of an object fails, it will automatically reconnect the client to another instance of the object. Part of the RPC system is an inter-server protocol, which ensures that the servers are aware of the fail-over of clients between objects and are able to perform the necessary repair and garbage collection.

These three interfaces represent generic Ensemble services in the sense that they are available on all platforms that Ensemble runs on. The services are encapsulated in COM interfaces to ease the integration of this complex technology into the wold of everyday Win32 programmers.

We have found these interfaces successful in assuring a certain level of ease of integration, but they still require that the developer have some awareness of the distributed architecture of the system being developed. As such, these interfaces are normally used to implement high performance cluster-style servers that can provide load balancing, fault-tolerance, or parallel processing.

For clients accessing such servers, our intent is to offer a very high degree of transparency, so that the developer can work without being aware that the server is using Ensemble-based tools. We also interested in providing support for managing standard (Ensemble unaware) server objects in such a way that they can achieve the high reliability of a sort similar to that available for Ensemble aware server objects. This type of transparency is achieved through the use of Ensemble-aware proxy objects at the client nodes, through high-available referral objects that manage the access to server objects and through manipulation of the binding process at the OXID services.

For applications that use standard DCOM interfaces, it is possible to do completely transparent replication and management. Ensemble provides a *Distributed Object Container* that runs as an NT service at each available server node. Operations on the container server are automatically presented to the contained objects, which see identical invocation sequences (this is called "state machine replication"). By dropping a standard DCOM object into such a container, it can be replicated with no changes at all. In the future, we plan to extend the performance and functionality of our container by providing other replication options in addition to state machine replication, and by providing functionality for managing the group of objects as a whole. Eventually, we should be able to provide a "replication wizard."

Ensemble provides two additional services that increase the reliability of the overall distributed system: a high-availability directory service, accessible through COM interfaces as well through LDAP, and a replicated object store. The latter also provides highly-available storage for persistent object state. Regular COM objects can be initialized from this high-available storage using the familiar *IPersist** interface.

6 NT server clusters and Wolfpack

One of the areas for which Ensemble seems well matched is high availability cluster computing. The Ensemble tools can be used for cluster management (to achieve higher reliability both in availability and performance), and can also be used when developing applications that exploit the services of a high-availability cluster. Our research into cluster computing is looking at ways to combine the Ensemble tools with Wolfpack.

Wolfpack is a management system under development at Microsoft, which targets the market for highly available clusters of NT-servers. At present, Wolfpack provides generic application fail-over, whereby groups of applications are automatically restarted on an alternative node of the cluster in case of a failure of the first node. Selection of the alternative node is based on the (physical) resources that need to be present for the application to function. Clients using such a service experience a disconnect from the failed server, and must then reconnect to a new instance of the application running at the surviving node. Wolfpack assigns IP addresses that can migrate between the nodes with groups of cluster applications, enabling clients to use the same address to connect to a server independent of the node on which it is actually running.

To support fault handling by the restarted application Wolfpack provides disk sharing (at SCSI level) and registry replication between the two nodes. The new application instance is expected to read the state of the failed application instance from the shared disk and, through a recovery mechanism, to recover the state of the application at the time of a failure. In effect, an application that was running on a node when it fails will be restarted only on another node that offers an indistinguishable execution environment. This approach offers a practical means of supporting applications that are restartable but not cluster aware, and that cannot be actively replicated using process group software.

However, there are a number of drawbacks to this approach, relating primarily to scale and to achieving continuous availability instead of fail-over. For example, it would not have been possible to solve the telecommunications coprocessor problem using a disk-based state management approach, because recovery would vastly exceed the 100ms limit on latency for individual requests and the 3-second overall limit on downtime *per year* for such applications. Moreover, in this approach, adding nodes to the cluster does not increase the perceived reliability of an application, its availability (only a limited number of nodes can physically share a disk), or its performance (only one instance of the application can be active at the same time). In practice the scalability model presented by the current Wolfpack organization is that of islands of 2-node clusters.

Even on a 2-node system Ensemble can be used to provide nearly continuous availability without the need to physically share resources. This is done using a primary/backup configuration where the second application instance is used as a *hot standby*. Ensemble guarantees that the state of the second instance is in sync with the primary instance. Fail-over to the second instance can be achieved with the guarantee that no state has been lost and recover/rollback (with possible loss of transactions) is not needed. Ensemble also supports an alternative called full active replication, where the application is active on both nodes and provides continuous operational guarantees while exploiting the resources on both nodes. In the limit, Ensemble is able to generalize cluster and application management to achieve *n*-way fault-tolerance, where each node can be backed up by *n* other nodes.

N-way fault-tolerance offers the potential for load-balanced use of the available resources, and big wins through scalable parallelism and large in-memory cache or databases. Returning again to the telecommunications application, much of the complexity is associated with

keeping the right information in the database buffer pool (cache), because databases of customer profile information are too large to fit in the memory of a uniprocessor. A cluster can potentially support an arbitrarily large memory (simply by adding nodes), creating the potential for a completely memory-mapped solution. We believe that comparable requirements arise in many domains.

Integrating Ensemble closely with Wolfpack to augment its cluster management system presents challenges beyond the ones encountered in simply porting Ensemble to NT. The management API of Wolfpack provides what are called *clusters, groups* and *resources*, where a cluster manages a set of groups, which contain a set of resources. Failure detection is offered at the level of resources while migration policies apply to groups. In these respects Wolfpack goes beyond Ensemble, both by offering API's specialized to a problem that we have not focused upon in our work to date, and by supporting management functions extensible by third parties, that would typically be outside the scope of software developed in academia. Basically, Wolfpack supports the notion that a group of resources will automatically be "moved" from node to node if a node fails. Resources can be associated with preferred nodes, and hierarchies of dependencies can be created, so that resources will be restarted in the right order.

The best match for this organization within Ensemble arises in a tool that we call the *Maestro Group Manager*. The group manager is a service, built out of Ensemble components, which provides hierarchical group management by tracking the membership of a managed process or communication group and automatically reconfiguring a group when one of its members fails.

Wolfpack treats its membership management subsystem in a modular manner; hence it is possible to replace the standard membership module with a proxy component that has a tight interaction with the Maestro tool. Doing this would provide Wolfpack with a scalable node and process membership. In effect, Ensemble now functions as the subsystem responsible for tracking cluster and resource membership. However, this step also makes the basic Ensemble functionality available to Wolfpack developers. For example, parallel database query engines could exploit Ensemble's high speed data replication and synchronization support. Such mechanisms are needed in implementing basic database operations on parallel processors, and standard solutions would presumably appeal to database vendors. Secure applications could use Ensemble's secure replication mechanism to distribute and manage keys within cluster members. And the techniques we used to achieve real-time responsiveness in our telecommunications switch example would become available to Wolfpack application developers in domains that demand real-time performance guarantees. For example, we recently developed a high performance Web server with real-time response guarantees, using the same approach.

The functionality offered by the Ensemble tools is quite a bit more general than that currently used in Wolfpack or planned for the next releases. In particular, Ensemble can be used to manage a cluster, but can also be used as a groupware programming environment. In this case, Ensemble is accessed from the Java language combined with a Web browser plug-in or an Active/X control. The Wolfpack membership tracking and reconfiguration mechanisms, in contrast, are a module internal to Wolfpack and not exposed for purposes other than the ones just mentioned. We believe that for developers of applications in which groups of participants cooperate or coordinate (such as multi-user interactive games, virtual reality environments, or business applications involving conferencing and briefings with multiple participants), group communication tools are important, and also easy to use. We see big advantages to factoring out group membership tracking and communication functionality, so that a single subsystem can support these very varied potential uses.

7 Ensemble Roadmap

At the time of this writing, the NT version of Ensemble is stable, including the C++ and Java programming interfaces (our Unix versions have been stable for some time and we have a growing user community). Understanding how to fully embed group communication and multicast functionality into COM/DCOM, and thus Active/X, will take some time; we already knew how to do this on the Unix and CORBA side because we supported such an option for Horus. We expect that by the autumn of 1997, Ensemble will be easily useable from Java or C++ through several COM interfaces on NT platforms including Wolfpack, and that by late in the year, we will have a stable integration of Ensemble into DCOM. In the same time frame we will be designing a number of Active/X control interfaces, based on the Ensemble/COM interface, and targeting group collaboration and communication opportunities.

The longer-term vision of our effort revolves around the seamless, highly transparent, introduction of "strong properties" into network applications developed using standard tools and programming practices. A funda-

mental premise underlying our work is that most critical applications are being developed using conventional off-the-shelf building blocks and combined into applications using standard techniques. We have come to believe that even the most critical applications are essentially forced to do this because it represents the only practical way to take advantage of modern computing technology. The challenge, as we see it, is to "harden" such systems without requiring source-code changes.

We are also expanding our emphasis on security. We believe that the community seeking "high reliability" has a broad notion of what this should mean, in which security goals are at least as important as fault-tolerance. A major goal is to integrate Ensemble with all the prevailing security options for distributed computing, so that any Ensemble application can transparently obtain authentication credentials for the processes with which it interacts, can encrypt or sign sensitive data by simply specifying the need, and can obtain trustworthy services such as group membership management and routing.

Finally, we are developing a more flexible notion of reliability itself. In Isis, Horus and Ensemble, up to the present, reliability has tended to be of an all-or-nothing flavor. The ability to build flexible protocol stacks has opened the door to supporting other notions of quality of service within Ensemble, but we haven't taken very much advantage of this so far. A big goal of ours in 1997 is to begin to develop probabilistic protocol stacks, which might offer a way to trade off between reliability and other goals, such as steady latencies or scalability. As noted earlier, Isis began to run into performance issues as it scaled into the low hundreds of participants. We believe that Ensemble could scale to thousands or tens of thousands using protocols that substitute a rigorous notion of "very probable behavior" for the "guaranteed reliability" of the basic virtual synchrony model.

Success in this effort could have a broad impact on the reliability and security of mission-critical distributed computing systems. Today, a tremendous rollout of these systems is occurring in settings that include medical critical care, air traffic control, on-board avionics systems, financial systems, factory automation, and critical business applications. Such applications demand extremely high levels of reliability. By providing easily used reliability and security solutions, well integrated with standard platforms and programming tools, and flexible enough to match the properties provided to the needs of the user, these critical applications can be made safe and secure. Indeed, we look forward to the day when reliability and security tools will be a common feature of standard distributed operating systems, much like file systems, TCP/IP communication, and Web technologies.

Online information

http://www.cs.cornell.edu/Info/Projects/Horus

References

Building Reliable and Secure Network Applications. K. Birman, Manning Publishing Company (Greenwich, CT) and Prentice Hall, January 1997. 550pp.

Software for Reliable Networks. K. Birman and R. van Renesse. *Scientific American* 274:5 (May 1996), 64-69.

Horus: A Flexible Group Communications System. R. van Renesse, K. Birman and S. Maffeis. *Commun. of the ACM* 39:4 (Apr. 1996), 76-83.

Using Group Communication Technology to Implement a Reliable and Distributed IN Coprocessor. R. Friedman and K. Birman. *Proceedings of TINA '96: The Convergence of Telecommunications and Distributed Computing Technologies.* Heidelberg, Germany, Sept. 3-5 1996, 25-42. VDE-Verlag.

Software releases

The Horus project has produced two generations of software. The first generation, Horus, is stable, and available for general use. There are no licensing fees for research use of Horus. Horus commercial rights, however, have been exclusively licensed by Cornell University to Isis Distributed Systems, which is developing a commercial product in this area. Contact rcbc@isis.com (Dr. Robert Cooper) for details concerning the commercial product offering, support, or other services.

Cornell University is making Ensemble available at no fee in source form for both research and commercial researchers. The system is available today and additional releases are expected periodically during 1997 and 1998 as new functionality is completed.