Annual Report: 7/15/96 - 12/31/96

A Scalable Distributed Multicomputer For Massively Parallel Applications
DAAH04-96-1-0327

James Griffioen, James Lumpp, Rajendra Yavatkar, Raphael Finkel Dept. of Computer Science, Univ. of Kentucky

Abstract

The goal of the Unify project is to investigate the design of a large-scale geographically disperse distributed shared memory system. This report describes the advances and accomplishments made during the past half year. First, we developed new weak consistency and synchronization abstractions that improve scalability and performance by reducing the amount of communication and avoiding high-latency synchronization messages. Second, we completed the design and began a prototype implementation of $(M \times N)$ $TMTP$, an efficient reliable $M \times N$ group-communication protocol used for efficient data transmission between Unify participants. Third, We began instrumenting the Unify prototype to efficiently record Unify events that occurs on Unify machines. The resulting logs can be filtered and display via a visual performance-monitoring tool called XUnify. Finally, we ported various scientific DSM applications to Unify and evaluated the system's performance. We also began developing and implementing a Treadmarks and DIB library that will allow us to quickly port a large number of existing Treadmarks and DIB applications to run on Unify. Performance results from the current prototype shows that Unify's relaxed consistency and synchronization abstractions combined with efficient communication protocols results in significantly better scalability and performance than existing DSM models.

Unify Project Progress Report

1 Manuscripts

The following papers describe research supported by this grant:

- J. E. Lumpp, Jr., J. Griffioen, R. Yavatkar, T. Kay, and C. Diaz, High-Level Reactive Support for Scalable Distributed Systems, *IEEE Bulletin of the Technical Committee on Operating Systems: Special Issue on Multiprocessor and Distributed Operating Systems*, Vol. 8, No. 2, pp. 13-27, Fall 1996.
- W. R. Dieter and J. E. Lumpp, Jr., Fault Recovery for Distributed Shared Memory Systems, *Proceedings of the IEEE Aerospace Conference*, February, 1997.
- T. J. Rademacher and J. E. Lumpp, Jr., High-performance Simulation of Neural Networks, *Proceedings of the IEEE Aerospace Conference*, February, 1997.
- S. Lakkapragada, J. Griffioen, and R. Yavatkar, An Extension to the Tree-based Multicast Transport Protocol (TMTP) to support $M \times N$ Communication, University of Kentucky Masters Project Report, December 1996.
- A. M. Levy and R. Finkel, Measure of Efficiency on Heterogeneous Machines, submitted to *IEICE Transactions*, January 1997.
- A. M. Levy and R. Finkel, Distributed behavior of constraint-satisfaction heuristics, in preparation.

2 Personnel Supported

The following Masters and Ph.D students were supported during the Fall 1996 semester or will be supported during the Spring 1997 semester.

- Chris Diaz (Ph.D. Student)
- Bill Dieter (Ph.D. Student)
- Amy Levy (Ph.D. Student)
- Shobhan Lakkapragada (Masters Student, Degree Awarded January 1997)
- Sivakumar Kuppuswamy (Masters Student)
- Babu Yama (Masters Student)
- Rajitha Sumanasekera (Masters Student)

3 Scientific Progress and Accomplishments

3.1 Unify Project Overview

The Unify research project is investigating scalable approaches for designing distributed shared-memory (DSM) multicomputers. Unify's goal is to create a massively parallel multicomputer from geographically disperse machines interconnected by wide-area networks such as large corporate/campus networks, the Internet, or evolving networks like the vBNS or Internet II. Unlike existing software packages such as PVM, MPI and others, which impose a message-passing model on the application programmer, Unify provides a distributed shared memory abstraction that hides all communication from the application programmer. Although it is widely accepted that distributed shared memory is a more convenient programming model than message passing, existing large-scale systems have turned to message passing because the overhead needed to support DSM was considered excessive and thus will not scale to massively parallel systems.

The Unify project is investigating a multicomputer design that will support DSM with minimal overhead. To support DSM across a wide-area network, Unify (1) minimizes communication, (2) hides the high latencies of a wide-area network, and (3) introduces an efficient group communication protocol that reduces traffic and reacts quickly to network errors and congestion. Some of Unify's novel contributions include:

Multiple Memory Types: Unify supports three basic memory abstractions for shared data. Random-access memory is directly addressable. Sequential-access memory is accessed in a read/front, write/append fashion. Associative memory is accessed via (key,value) pairs. Sequential-access and associative memory can often be supported with weaker spatial consistency guarantees, that can be implemented with fewer high latency messages than random-access memory segments.

Multiple Grades of Consistency: Unify allows an application to select the appropriate consistency semantics from a spectrum of consistency protocols, including "automatic" methods in which the operating system enforces consistency, and "application-aided" methods where the user defines consistency synchronization points. Several existing DSM systems have demonstrated the benefits of weak application-aided temporal consistency models. In addition, Unify supports weak automatic methods where the memory becomes consistent after some given time period. This type of consistency is particularly useful for applications such as Grapevine that can detect stale data. Weak consistency also allows Unify to batch/combine updates to reduce network traffic.

Unify also introduces a new consistency dimension called "spatial consistency", that determines the ordering of data in a replica. For many distributed applications that use keyed lookups or sequential access, the order of the data items within a segment is unimportant; only the data values are important. Relaxing the spatial ordering constraint can help avoid the high network latencies and serialization that would otherwise occur in writing to a sequential or associative segment.

Scalable Synchronization Primitives: Most DSM designs support locks, semaphores, and/or barriers as the basic synchronization methods. These synchronization abstractions pose a serious bottleneck, particularly for large systems with high network latencies. To provide efficient synchronization in the presence of long and varying network latencies, Unify provides a modified form of eventcounts and sequencers. For a large class of applications, eventcounts can significantly reduce communication traffic, because they allow participants to observe events at different times instead of forcing all participants to observe events simultaneously.

Scalable Reliable Group Communication: Unify employs an efficient reliable multicast protocol called TMTP for large-scale dissemination of shared data. Even networks that provide Quality of Service (QoS) guarantees do not provide the end-to-end reliability guarantees needed by distributed applications. TMTP provides reliable multicast communication using a combination of rate-based flow and error control, and a hierarchy of receivers that use localized retransmissions and NACK suppression.

Prior to the start of this grant, the basics of the system architecture had been outlined and a crude prototype system had been developed to demonstrate the viability of the model. The following documents our achievements since the inception of the grant.

3.2 Recent Accomplishments

During the half-year reporting period, we have made significant progress toward achieving the objectives and milestones defined for the project.

1. We have made significant strides in developing and enhancing the design of the Unify System. We have developed new weak consistency abstractions and synchronization abstractions that improve the scalability and performance of the system.

2. We have completed the design and have begun a prototype implementation of $(M \times N)$ $TMTP$, an efficient reliable $M \times N$ group-communication protocol used for efficient data transmission between Unify participants.

3. We have begun instrumenting the Unify prototype to record all Unify events that occur across all machines. The resulting logs can be filtered and display via a visual performance-monitoring tool called XUnify.

4. In addition to developing and testing new applications on Unify, we have begun developing and implementing a Treadmarks library and a DIB library for Unify that will allow us to quickly port a large number of existing Treadmarks and DIB programs to run on Unify.

The following briefly describes the specific achievements made in each of these four areas.

To improve the scalability and performance of the Unify system, we designed or modified several abstractions that further relax the consistency and synchronization demands placed on Unify's shared memory system.

We modified Unify's basic consistency primitives, `PutUpdates` and `GetUpdates`, to be non-blocking. This change increases concurrency by allowing applications that do not care when the system reaches a consistent state to execute concurrently with the distribution of updates. To support applications that depend on a consistent state, we added new operations to block a process until the distributed shared memory reaches a consistent state. The non-blocking feature of `PutUpdates` is particularly useful in the context of sequential segments where reads are blocking and thus the transmission of updates can be done asynchronously, reaching different Unify hosts at different times.

We modified the `GetUpdates` operation to allow any host to obtain an unlocked, up-to-date copy of a shared segment, thereby reducing the synchronization cost of read-only accesses. Many applications, such as the Stanford SPLASH *water* benchmark, constantly change the way in which they access shared segments, alternating between read-only and read-write access patterns. Our results show that Unify's ability to support different access patterns in the Water benchmark improved performance by a factor of 8 for a problem size of 512 molecules (as opposed to a system that does not allow the access method to change).

We developed a highly efficient implementation of Unify's eventcount abstraction using unreliable multicast to communicate event-state information. The lack of ACKS and retransmissions when using unreliable multicast reduces the total amount of data transmitted and also reduces link congestion over shared data paths. In most cases, errors (lost messages) can be corrected by future messages. Only when the algorithm detects or suspects incorrect event-state information is the overhead of error correction incurred. As in the original Unify design, synchronization accesses need not be consistent, thereby increasing concurrency.

We developed a no-overhead local-locking synchronization abstraction for use with the `GetUpdates` operation. The new operation prevents a host from giving out updates to a segment until the local-lock is released. Local-locking requires no communication with other hosts and thus allows Unify to support a lock-based synchronization abstraction at no cost.

We have developed and implemented a differencing algorithm that reduces the amount of data transmitted with `GetUpdate` and `PutUpdate` operations. The differencing algorithm uses a combination of version numbers and difference merging to reduce the size of the difference list. Differencing also allows Unify to support false sharing, which is useful when emulating packages like Treadmarks.

We have made significant progress in the development of the Unify prototype implementation. Several of the recent design decisions listed above have been incorporated into the implementation, including non-blocking `PutUpdates` and `GetUpdates`, the local-locking abstraction, and read-only `GetUpdates`. We have also made changes to the flow control mechanism in the communication protocol to make better use of the available bandwidth and avoid overrunning slow receivers. Because the Unify library is a multithreaded library, we developed an architecture independent threading package that will allow us to quickly port the system to new architectures or switch between C-threads and kernel threads. Eventcounts based on unreliable multicast are in the process of being implemented and evaluated. All current test results still use our original implementation of eventcounts. We have released a beta version of Unify locally for use by our students and other researchers in the CS and EE departments. After further evaluation, we intend to make this release available on the web for the general public.

We have completed the port of two distributed applications from the Stanford SPLASH benchmark suite, namely *water* and *mp3d*. Because Unify supports various consistency models and synchronization models, we have developed several versions of each application and shown that Unify's relaxed consistency and synchronization models offer superior scalability and performance over conventional consistency and synchronization models. We compared our implementation with systems such as CRL (from MIT) and shown significant performance improvements. We continue to port new applications to Unify, and with the addition of a Treadmarks library and DIB library, we expect to be able to execute all existing Treadmarks and DIB applications without modification.

To make the Unify prototype easier to use, we have added code that allows it to inter-operate with PVM's task spawning capabilities. Consequently, users can use the PVM console or XUnify console to select machines and launch Unify applications.

We have begun instrumenting the Unify library with calls to gather network-communication and host performance information. This information is logged and can be displayed at runtime or in postmortem format via a graphical user interface called XUnify. XUnify was developed from the XPVM application and thus includes all the features and performance-monitoring support of XPVM. In addition, we have added several new features to collect Unify specific information and events and display them. The current package monitors all `PutUpdates`, `GetUpdates`, segment creations/deletions, synchronization events, and network communication packets including message fragment ACKS and NACKS. The Unify segment number is recorded along with information about the event to correlate events with memory segments. A prefiltering mechanism allows users to dynamically select which information should be recorded; a postfiltering mechanism allows users to select which information should be displayed.

To provide efficient, reliable, and scalable multicast communication in a wide-area environment, we designed an $M \times N$ reliable multicast communication protocol called TMTP, which Unify hosts use to distribute updates to shared memory regions and internal Unify state information. Our achievements include the design of self-organizing shared hierarchical control trees. As new senders and receivers join and leave a communication group, the tree is dynamically constructed or reorganize to identify the shortest paths through existing domain managers.

We implemented a prototype version of the TMTP protocol and its session manager as an application-level library that can be linked with any application, not just Unify. Domain managers are automatically spawned as senders and receivers join the communication group. Distribution of multicast messages occurs via standard IP multicast (possibly via the MBONE) while all ACK traffic flows over the shared hierarchical control tree. We have used the protocol in simple video-dissemination applications over the Internet MBONE and have shown that it exhibits excellent scalability and rapid error recovery with minimal network traffic. A recent paper by J.J. Garcia-Luna and B. Levine at UC Santa Cruz comparing various multicast approaches shows that TMTP's hierarchical approach exhibits the best scalability of current reliable multicast approaches.

To provide support for existing distributed applications written for other systems, we are writing two application-level libraries that will run on top of Unify; one to support Treadmarks applications and another to support DIB applications. The Treadmarks package provides a lazy-release-consistent DSM computational model. Our treadmarks library provides standard Treadmarks operations such as `acquire` and `release` by mapping them onto the appropriate Unify operations. The DIB package (Distributed Implementation of Backtracking) provides a distributed-tree computational model for implementing applications based on backtracking. Like the Treadmarks library, our DIB library maps DIB operations onto the appropriate Unify operations. Support for these two models of distributed computing demonstrates the power and flexibility of the Unify's consistency models. In addition, it will allow us to support a large amount of existing software without any modifications to the software.

3.3 Research Plan for 1997

During the next reporting period we will continue to focus on the design of Unify while simultaneously moving forward on its implementation. Some of our research plans include:

- complete the design of our automatic consistency models, namely eventual-consistency and best-effort consistency
- implement random-ticket support in the Unify prototype
- complete the implementation and evaluation of the Treadmarks library and the DIB library
- port other distributed shared memory benchmark applications (from the SPLASH suite and NASA application suites) and evaluate their performance under Unify.
- re-implement clusters and workspaces in Unify to more efficiently share state information
- complete the implementation of eventcounts over unreliable multicast
- thoroughly test the TMTP protocol library and implementation across the MBONE
- package and release the TMTP source code on the web for public use and evaluation (we have already had some inquiries)
- complete the design of our concast-style communication model
- continue development of our *reactive object* system which includes defining the host and network layer performance monitoring system and the invocation of adaptation methods.
- development a checkpointing system that supports fault-tolerance and interactive debugging of Unify applications.
- continue development of a Unify simulator that we recently began implementing to generate performance models for large systems.

4 Technology Transfer

In addition to publishing our findings in conference proceedings, journals, and technical reports, we will make our prototype implementations available. We have currently restricted the release of the Unify beta prototype to students and researchers on the Kentucky campus. Similarly, we have restricted the distribution of the TMTP prototype to the University of Kentucky and collaborators at Intel's Communication Architecture Lab. Sometime during the next year, we intend to release the beta version of both software packages to the general public via the web.