Simple Reliable Multicast for Parallel Processing in Extended LANs

Jaiwant Mulik Phillip Conrad Yuan Shi
jmulik@unix.temple.edu
Temple University
Department of Computer and Information Sciences
Philadelphia, PA, 19122, USA

Abstract

A typical problem for a parallel processing system involves broadcasting large amounts of data from a master to several worker programs. This paper describes a reliable-multicast method to reduce the communication costs of this distribution. Our solution relies on a tuple-space mechanism as implemented in the Synergy system. We present results showing that even a simple implementation of reliable multicast can dramatically improve performance.

1. Background: The Synergy system

The *tuple space* model was introduced in the LINDA [2] system. Synergy [4] is a tuple-space based SIMD parallel processing system that runs on a network of workstations. From a programmer's point of view, Synergy consists of three types of entities: exactly one master, one or more workers, and one or more tuple spaces. The master and workers are user programs, while tuple spaces are object repositories to/from which the master and workers can write/read objects called tuples. A tuple is a named data object. Once put into a tuple space, tuples are persistent until removed. Another example of a tuple-space based system is LIMBO [1] which uses reliable multicast in conjunction with tuple spaces to provide Quality of Service management in mobile-aware distributed applications.

A parallel solution to a problem is implemented in Synergy by having the master divide the problem space into sub-problems, and then put one or more tuples for each sub-problem into tuple space. The workers then obtain tuples from the tuple space, compute the (partial) solution and return the solution to the same or another tuple space. The master, after reading all the solution tuples, generates the complete solution and returns the result. Currently, workers use Synergy system calls to read tuples from the tuple space over a network. Prior to this work, Synergy system calls used only unicast communication.

2. Multicasting

In parallel computing, the total solution cost is the sum of the computation and communication costs. For many problems the total cost is dominated by the initial problem distribution cost, i.e. the time to distribute the sub-problem and relevant data to the workers. In cases where a large proportion of the sub-problem data to be distributed is identical, multicasting can improve performance.

IP Multicasting is inherently unreliable. In unicasting, reliability is acheived by having the sender retransmit. However, unicast reliability techniques are not scalable for multicast due to the well-known *ack-implosion problem*. Many sophisticated reliable multicast algorithms have been investigated [3]. In this paper, we show that significant gains can be obtained even with a minimal use of multicast.

Specifically we have exploited two features of Synergy: (1) All data to be distributed is present in the tuple space and the tuple space is persistent. (2) Workers normally go to the tuple space to get their sub-problems. We shift the responsibility of reliability from the sender to the receiver. The idea can be summarized as:

- MASTER: (1) Generate the data and place in tuple space. (2) Wait for acks from workers indicating readiness. (3) Multicast the data.
- WORKER: (1) Send ack to master indicating worker is ready. (2) Receive data via multicast. (3) If the next expected tuple is not received within an (application-specific) interval then retrieve the lost data from tuple space using regular Synergy functions.

3. Experiment Design and Analysis of Results

The experiment involved distributing an $N \times N$ integer matrix to P processors, as in a parallel implementation of matrix multiplication. We calculated the average delay over ten repetitions of the experiment for each combination of

0-7695-0912-6/00 $10.00 © 2000 IEEE

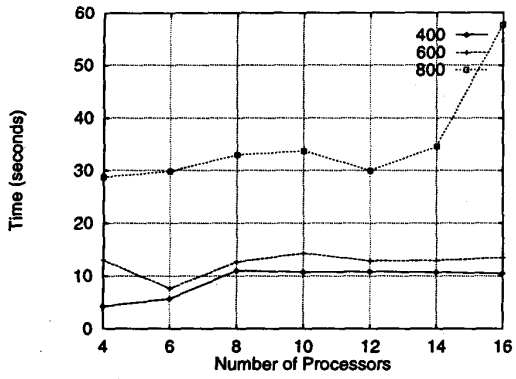

Figure 1. Performance: Reliable Multicast

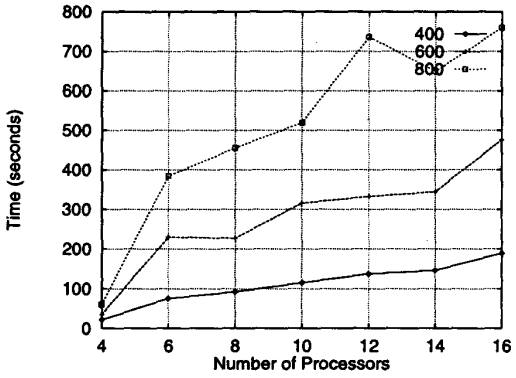

Figure 2. Performance: Unicast

$N = \{400, 600, 800\}$ and $P = \{4, 6, 8, 10, 12, 14, 16\}$, first using UDP multicast and then UDP unicast packets. Each packet contained one matrix row and 4 bytes of application header, plus the normal IP+UDP headers (28 bytes).

Fig. 1 indicates that for multicast the average total time taken is relatively constant as P increases. By contrast, Fig. 2 charts the amount of time taken to deliver data using multiple unicasts. Fig. 3 shows the performance gain (reduction in latency) obtained when reliable multicast is used instead of unicast. Our data indicates that multicasting can distribute data to workers faster than having each worker 'pull' the data from tuple space. Note that the average time taken to deliver a fixed amount of data using unicast increases with P, while the average time taken to deliver this data using multicast *does not increase* with P (except in the case of P=16). Hence not only is reliable multicast faster than reliable unicast but is also inherently more scalable. Performance and scalability are key requirements of any distributed computing architecture.

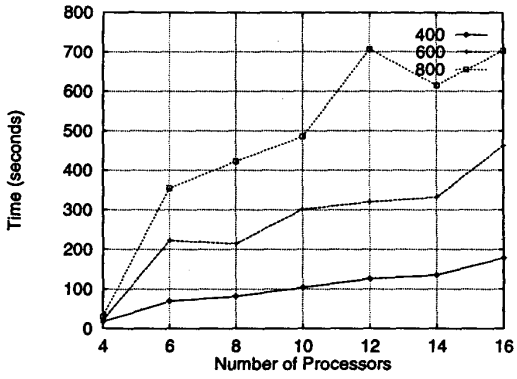

Figure 3. Performance Gain

Regarding P=16, note that on a LAN, UDP data losses are caused mainly due to context switches that lead to flow control errors. This is visible in Fig. 1; note the dramatic increase in time for N=800 and P=16. In this case the workers lost between 2% and 26% of all multicast data.

4. Summary and Future Work

This paper describes the performance benefits that can be obtained from a simple modification to an existing distributed system: adding a feature for multicast distribution to an implementation of tuple space. Our data shows that a dramatic performance improvement can be obtained from even a very simple application of multicast techniques.

A more sophisticated approach to solving this problem might take the form of a replicated tuple space, using reliable multicast protocols to maintain consistency among the replications of the tuple space on various systems. Such an approach would introduce a layer of indirection between the master and the workers, allowing the master to start transmitting without waiting to synchronize with all the workers.

References

[1] G. S. Blair, N. Davies, A. Friday, and S. P. Wade. Quality of service support in a mobile environment: an approach based on tuple spaces. In *IWQOS'97*, New York, May 1997.

[2] D. Gelernter. Generative communication in Linda. *ACM Transactions in Programming Languages*, 1(7):80,112, January 1985.

[3] B. Levine and J. Garcia-Luna-Aceves. A comparison of reliable multicast protocols. *ACM Multimedia Systems Journal*, 6(5):334–348, August 1998.

[4] Y. Shi. Building worthy parallel applications using networked computers – a tutorial for synergy v3.0. `http://www.cis.temple.edu/~shi/synergy.html`, January 1995.