End-to-End Latency Analysis and Evaluation of a Fault-Tolerant CORBA Infrastructure [*]

W. Zhao, L. E. Moser and P. M. Melliar-Smith
Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
wenbing@alpha.ece.ucsb.edu, moser@ece.ucsb.edu, pmms@ece.ucsb.edu

Keywords: Software Performance, Distributed Computing, Client/Server, Network Protocols, Cluster Computing

Abstract

This paper presents an end-to-end latency analysis and evaluation of a fault-tolerant CORBA infrastructure that uses the Totem group communication system. We measure the probability density functions (pdfs) of the end-to-end latency for synchronous remote invocations, and analyze the latency values at the peak probability densities. Our study shows that the Totem single-ring protocol directly affects the end-to-end latency, in particular, the message-send delay. To alleviate the message-send delay for passive or semi-active replication, the relative position between the client and the primary server, the token circulation time, and the processing time at the client and the server must be considered to determine the most favorable position for the primary server. For active replication, the presence of duplicate messages adversely affects the performance. However, if an effective sending-side duplicate suppression mechanism is implemented, active replication is more advantageous than the other replication styles because of the automatic selection of the most favorable position of the server replica that sends the first non-duplicate reply.

1 INTRODUCTION

Distributed computing has become widely used for many kinds of applications. To facilitate distributed computing, a number of architectures, *e.g.*, CORBA, DCOM and EJB/J2EE, have been developed. The Common Object Request Broker Architecture (CORBA) [15] supports distributed object applications with client objects invoking server objects, which return replies to the client objects after performing the requested operations. CORBA's advantage over other distributed computing architectures lies in its support for heterogeneous architectures, operating systems and programming languages.

Fault-tolerance infrastructures [3, 4, 7, 8, 11, 13, 14] often use a group communication protocol for reliable totally-ordered message delivery. Such a protocol significantly simplifies the replication, logging and recovery mechanisms that are needed to achieve strong replica consistency.

There has been extensive research on the performance of stand-alone group communication protocols. However, there have been very few studies of the performance of group communication protocols in the context of fault-tolerance infrastructures. The strategies that the group communication protocol uses to achieve reliable totally-ordered multicasting of messages must be considered when deciding on the replication style, and the distribution of replicas. Investigation of the interactions between the replication mechanisms and the group communication protocol is also necessary to build a high-quality, fault-tolerance infrastructure.

In this paper we present such an investigation, by analysis and by measurement, of a fault-tolerant CORBA infrastructure [20] that uses the Totem group communication system [1]. To obtain an accurate picture of how the fault-tolerance infrastructure operates, we measured the probability density functions (pdfs) of the end-to-end latency for synchronous remote invocations. The average (or mean) latency is a common metric, but it often obscures the principal factors that determine the end-to-end latencies. Our analyses and measurements show that, because of the potential message-send delay imposed by Totem, care must be taken in selecting the node to run the primary server replica for passive or semi-active replication. The relative position of the client and the primary server replica, and the server processing time, must be considered together when making this choice. If an effective sending-side duplicate detection mechanism is implemented in the fault-tolerance infrastructure, active replication is more advantageous than passive or semi-active replication, because it benefits from the automatic selection of the most fa-

[*] This research has been supported by DARPA/ONR Contract N66001-00-1-8931 and MURI/AFOSR Contract F49620-00-1-0330.

vorable primary server replica for sending replies and, hence, it minimizes the total message-send delay. We also show that the fault-tolerance infrastructure can incur significant computation-related runtime overhead, in addition to communication-related runtime overhead, if the server replicas engage in intensive computation.

2 BACKGROUND

2.1 Replication Techniques

In a fault-tolerant distributed application, critical components are replicated to mask faults. To maintain replica consistency, and to provide a single-copy image to the other components in the system, such systems often run on top of a fault-tolerance infrastructure. Group communication systems are typically used in fault-tolerance infrastructures to provide reliable totally-ordered multicast messages and group membership services to the replication mechanisms.

Active replication and passive replication are the two most common replication styles. In active replication, all replicas perform exactly the same operations in the same order. To protect against value faults, active with voting can be used where the inputs and outputs from all replicas in a group are voted. The replies or requests are delivered to a replica only if the majority of such messages are identical. In passive replication, only one replica (the primary) executes in response to the client's requests. Passive replication has two variations: warm passive replication and cold passive replication. In warm passive replication, the primary periodically checkpoints its state at the other replicas (backups). In cold passive replication, the backups are not launched until the primary is detected to have failed. The fault-tolerance infrastructure must retrieve checkpoints from the cold passively replicated primary and log them.

Semi-active replication is a hybrid replication style developed in the Delta-4 architecture [17] to combine the advantages of active and passive replication. In semi-active replication, all replicas execute the same operations in the same order except that only one replica (the primary) issues responses or nested invocations. With semi-active replication, no duplicate messages are sent under fault-free conditions, and recovery is generally faster for semi-active replication than for passive replication. However, semi-active replication cannot tolerate value faults as in active replication with voting.

2.2 The Totem System

Totem [10] is a group communication system that provides reliable totally-ordered multicasting of messages to processors operating in a single local-area network (LAN), or in multiple LANs interconnected by gateways. In this investigation, we employ the Totem single-ring protocol [1] that operates over a single LAN. Totem provides a process group interface that allows applications to be structured into process groups and, thus, maintains information about the current memberships of the groups.

The Totem single-ring protocol uses a logical token-passing ring superimposed on a local-area network, such as an Ethernet. The token circulates around the ring as a point-to-point message, with a token retransmission mechanism to guard against token loss. Only the processor holding the token can broadcast a message. The token conveys information for total ordering of messages, detection of faults and flow control.

The sequence number field in the token provides a single sequence of message sequence numbers for all messages broadcast on the ring and, thus, a total order on messages. When a processor broadcasts a message, it increments the sequence number field of the token and gives the message that sequence number. Other processors recognize missing messages by detecting gaps in the sequence of message sequence numbers, and request retransmissions by inserting the sequence numbers of the missing messages into the retransmission request list of the token. If a processor has received a message and all of its predecessors, as indicated by the message sequence numbers, it can deliver the message.

The all-received-up-to field of the token enables a processor to determine, after a complete token rotation, a sequence number such that all processors on the ring have received all messages with lower sequence numbers. A processor can deliver a message as "stable" if the sequence number of the message is less than or equal to this sequence number. When a processor delivers a message as stable, it can reclaim the buffer space used by the message because it will never need to retransmit the message subsequently.

The token also provides information about the aggregate message backlog of the processors on the ring, allowing a fair allocation of bandwidth to the processors. The flow control mechanism provides protection against fluctuations in the processor loads, but is vulnerable to competition for the input buffers from unanticipated traffic.

2.3 Pluggable FT CORBA Infrastructure

The Pluggable Fault-Tolerant (FT) CORBA infrastructure [20] follows a novel non-intrusive integrated approach to render CORBA applications fault-tolerant. The fault-tolerance mechanisms are plugged into the CORBA ORB by exploiting the pluggable protocols framework that most modern ORBs provide [6]. On the server-side, fault tolerance is provided by a FT protocol plug-in, the Totem protocol, the Replication Manager, the Fault Notifier and the Fault Detectors. The Replication Manager and the Fault Notifier are implemented as CORBA objects running as separate processes, and are replicated using active replication. The Fault Detector that detects faults at the object level is implemented as a component of the FT protocol plug-in.

The Replication Manager implements most of the relevant interfaces specified in the FT CORBA standard. In particular, the Replication Manager is responsible for generating an Interoperable Object Group Reference (IOGR) for each replicated CORBA object (as indicated by the `create_object()` method of the `GenericFactory` interface). Because a group communication system is used to provide messaging to the replication mechanisms within a fault tolerance domain, the IOGR profiles address gateways to the fault tolerance domain, rather than object group members. The object group membership is maintained by the replication mechanisms separately from the IOGR. The current prototype supports the infrastructure-controlled consistency policy; therefore, the `ObjectGroupManager` interface is not implemented.

To accommodate the FT protocol plugged into the ORB, we have rewritten the process group layer of Totem. In addition to providing the reliable totally-ordered delivery service, Totem also serves as a process-level and host-level fault detector. When it detects a fault, Totem creates a fault report and delivers it to the Fault Notifier. Totem also conveys, to the Fault Notifier, the fault reports that the object-level fault detector generates.

Unreplicated clients connect to the replicated servers through passively replicated gateways that provide access into the fault-tolerance domain that contains the replicated servers. The client-side failover mechanisms specified by the FT CORBA standard are also implemented as a protocol plug-in.

3 RELATED WORK

Several researchers have developed object replication and fault tolerance infrastructures for CORBA. Some of those infrastructures [3, 4, 7, 11] were developed before the adoption of the Fault-Tolerant CORBA standard [16]. Others [8, 13, 14, 20], that were developed after that standard was adopted, implement that standard partially or completely. None of those papers presents analyses or measurements of the pdfs for the end-to-end latency of a Fault-Tolerant CORBA system.

Substantial work, both analytic and experimental, has been undertaken on the performance of the Totem protocols. Budhia [2] measured performance with a test message driver located in the Totem protocol stack, so that the test messages never waited for the arrival of the token. Thomopoulos [18] analyzed and measured the probability density functions of the latency for sending oneway messages in Totem. In his analysis, he assumed that, at the moment that a message is generated, the token is randomly located on the ring. In addition, he assumed that the send events are independent of each other. However, the measurements are similar to [2]; in effect, only the tail of the probability density functions are measured and compared with the analysis. In our analyses and measurements, we consider both fixed and random distributions of the client "think" times. In particular, we recognize that, for synchronous remote invocations in a fault-tolerance infrastructure, the send events can no longer be considered independent. We study the consequence of the correlation of the sending events for requests and replies, and present guidelines for achieving the best end-to-end latency by minimizing the message-send delays at the server.

Karl *et. al.* [5] investigated the effects of faults and of scheduling delays on the latency of the Totem protocol, and showed that both can induce considerable variation in the latency. We have not, in this work, investigated the effects of either.

Narasimhan *et. al.* [11, 12, 13] measured the performance of the Totem protocol and of the fault-tolerance mechanisms mounted on the Totem protocol. Her measurements focused primarily on throughput rather than on latency.

4 LATENCY ANALYSIS

For the latency analysis, we focus on the determination of the end-to-end latency at the peak probability densities when there are no faults. We neglect all of the factors that affect the latency with low probabilities, such as message (or token) loss and retransmission, replica processing uncertainties, operating system scheduling uncertainties, etc.

4.1 The Model

We assume that the Totem logical ring consists of n nodes, with a single client/server application running on those nodes. We assume that the client is unreplicated and acts as the ring leader and that the server replicas are distributed across two or more nodes on the rest of the ring.

We label the nodes of the Totem logical ring $node_0$, $node_1$, ..., $node_n$, starting with the ring leader. The token circulates around the ring sequentially from $node_0$ to $node_1$ and so on to $node_n$ and then back to $node_0$. It takes time T_0 for Totem to pass the token from one node to the next. The token circulation time, *i.e.*, the time for the token to circulate around the ring once, is T_r. A server replica running on $node_k$ is k steps away from the client in the direction in which the token circulates, and the client is $n - k$ steps away from the server. We assume that the processing time for a request at the server is T_{sproc}, and that the processing time for a reply at the client is T_{cproc}.

The interval between two consecutive remote method invocations issued by the client (the "think" time) is T_{think}. T_{cdelay} is the delay after a client issues a request and before the node receives the token and sends the message; likewise, T_{sdelay} is the delay for the reply message to be sent at the server. We assume that the cost T_m for Totem to send and handle a request or reply is the same for every message. (We will see in the next section that this is an approximation.) There are several other factors that affect T_m, including the message length, the type of message and the relationships between the replica at the sending node and the replica at the neighboring node in the next token step. The end-to-end latency for a synchronous remote invocation is T_{e2e}.

4.2 Determination of the End-to-End Latencies

The end-to-end latency T_{e2e} is the sum of the processing time at the client T_{cproc} and the processing time at the server T_{sproc}, the message-send delay at the client T_{cdelay} and at the server T_{sdelay}, and twice the user message transmission time T_m, as given by

$$T_{e2e} = T_{cproc} + T_{sproc} + T_{cdelay} + T_{sdelay} + 2T_m \quad (1)$$

For a server replica running on $node_k$, the send delay for the reply is given by

$$T_{sdelay} = iT_r + kT_0 - T_{sproc} \quad (2)$$

where i is the minimum non-negative integer such that T_{sdelay} is greater than or equal to zero. The term iT_r is present for two reasons. First, the processing time at the server replica might span several token circulation times, in addition to the interval between the time that the server receives the request and the first token visit (*i.e.*, kT_0). Secondly, a node might have to wait up to one additional token circulation time after the reply is written to Totem's send buffer before the message is actually broadcast.

Likewise, the send delay for the request at the client is determined by

$$T_{cdelay} = jT_r + (n-k)T_0 - (T_{cproc} + T_{think}) \quad (3)$$

where j the minimum non-negative integer such that T_{cdelay} is greater than or equal to zero. The meaning of the term jT_r is similar to that for the server-side send delay. The only difference is the additional "think" time at the client.

The token circulation time T_r is given by

$$T_r = nT_0 + pT_m \quad (4)$$

where n is the number of nodes on the ring, T_0 is the token passing time, and p is the number of user messages sent in one round of the token circulation. The meaning of T_m deserves further explanation.

After expanding the terms in Equation (1), we have

$$T_{e2e} = (i+j)T_r + 2T_m + nT_0 - T_{think} \quad (5)$$

It might appear to be counterintuitive that the end-to-end latency does not depend on the server-side and client-side processing time. Actually, the processing time is included in the first term $(i+j)T_r$ of the equation. Equation (5) predicts that the client will not see a continuous increase in the end-to-end latency if the server processing time varies continuously, assuming that T_{think} is constant. Furthermore, the sending of undesirable user messages, such as duplicate messages for active replication, or checkpoint messages for passive replication, can incur unnecessary contribution to the end-to-end latency.

Another implication of Equation (5) is that the client invocation patterns, reflected by the term T_{think}, also affect the end-to-end latency. In particular, the client "think" time T_{think} affects the end-to-end latency when duplicate replies are present for active replication. If T_{think} is longer than the time that it takes the client to receive duplicate replies and filter them, then the duplicate messages will have no negative effect on the end-to-end latency as seen by the client. However, if T_{think} is shorter than that time, the send delay for the next request might be increased by Totem's sending duplicate replies, because it takes longer for the client to receive the token.

If there is no extra message other than the non-duplicate request and reply for each invocation, Equation (5) can be simplified to:

$$T_{e2e} = (i+j+1)nT_0 + 2T_m - T_{think} \quad (6)$$

Semi-active replication corresponds to this case. Assuming that there is no "think" time at the client, and that no duplicate message appears on the ring, Equation (5) can be further simplified to

$$T_{e2e} = (i+j+1)nT_0 + 2T_m \quad (7)$$

Because the position of the primary server replica on the ring can affect the values of i and j, running the primary server replica at different nodes can lead to end-to-end latencies that differ by a complete idle token circulation time nT_0. Thus, care must be taken to determine the best position on the ring to run the primary server replica to achieve the best end-to-end latency. For active replication, however, because all of the server replicas send replies, the replicas actually enter a competitive mode for sending replies. The replica at the most favorable position on the ring successfully sends the reply first. The replies sent by the replicas at the less favorable positions, if any, are deemed to be duplicates. A consequence of this observation is that the replication degree (*i.e.*, the number of replicas) should be as close as possible to the total number of nodes on the ring. In practice, we also need to consider the additional processing involved by running more copies of the replicas, which is particularly a concern for single processor machines where multiple processes compete for the CPU.

The analysis above is intended to capture the latency, at the peak probability density, of the fault tolerance system under investigation. A more extensive model is needed to represent more completely the behavior of such a system (*e.g.*, the tails of the pdfs).

5 MEASUREMENTS

5.1 Experimental Setup

The latency measurements were performed on a testbed of four Pentium III PCs, each with 1GHz CPU, 256MBytes of RAM, running the Mandrake Linux 7.2 operating system, over a 100Mbit/sec Ethernet, using Vertel's e*ORB [19]. During the measurements, there was no other traffic on the network.

Our testbed consists of a high-quality local-area network. We have determined that the loss rate of packets is below 10^{-5} for a test client/server application, using UDP to send and receive messages with 1KByte of payload synchronously. Retransmissions have an insignificant effect on the pdfs that we measured.

Four copies of Totem run on the four PCs, one for each PC. We refer to these PCs as $node_0$, $node_1$, $node_2$ and $node_3$, in the order of the logical token-passing ring imposed by Totem, where $node_0$ is the ring leader. Because one instance of Totem runs on each node, we use the same node designation to refer to the Totem instance on a node.

In the experiments, a CORBA client sends 1KByte of payload (in the form of a sequence of 256 longs) to a replicated server, and the replicated server echoes back the same payload (which corresponds to our assumption of identical user message transmission time). The 1KByte payload length is chosen so that the complete user message can be put into a single Totem packet (1.4KBytes). The Totem flow-control mechanism is not exercised in our measurements.

The client runs on the ring leader, $node_0$. The server is three-way replicated, with one replica running on each of the other three nodes, $node_1$, $node_2$ and $node_3$. For each run, the client issues 10,000 remote method invocations on the server.

5.2 Measurement Methodology

At the client, the current time is recorded immediately before the client issues a request and after the reply returns. The difference between the two measured times is the end-to-end latency for that method invocation.

Because the operating system rounds the time of the sleep calls (*e.g.*, nanosleep(), select(), etc.) to clock ticks, we use an alternative way to control the intervals between two subsequent calls ("think" times) at the client. We insert an empty iteration loop (*e.g.*, a `for` loop) between two consecutive remote invocations. The amount of client "think" time is controlled by setting different upper bounds for the iteration loop. The same approach is used to simulate the server processing time. A drawback of this approach is that the intended distribution of the "think" time at the client, or the computation time at the server, might be distorted by the interference of token processing, as shown in the next section. To control the server processing time for each remote invocation, the upper bound for the iteration is generated at the client and piggybacked in the payload of the request, so that the server replicas have consistent information about the computational load. The actual processing time is recorded at the server replica and is piggybacked in the payload of the reply.

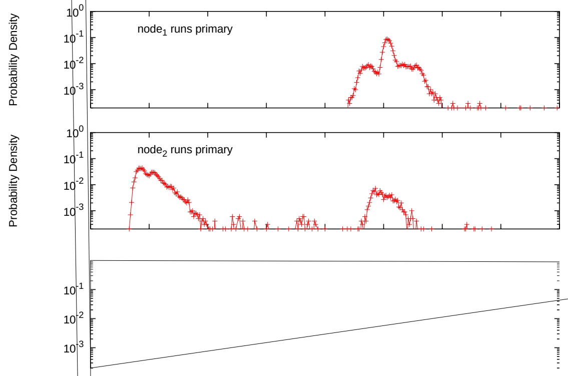
The token circulation time is the difference between the time Totem receives the token and the time it last sent the token. The token passing time is obtained by dividing the idle token circulation time (when there is no user message) by the number of nodes on the ring.

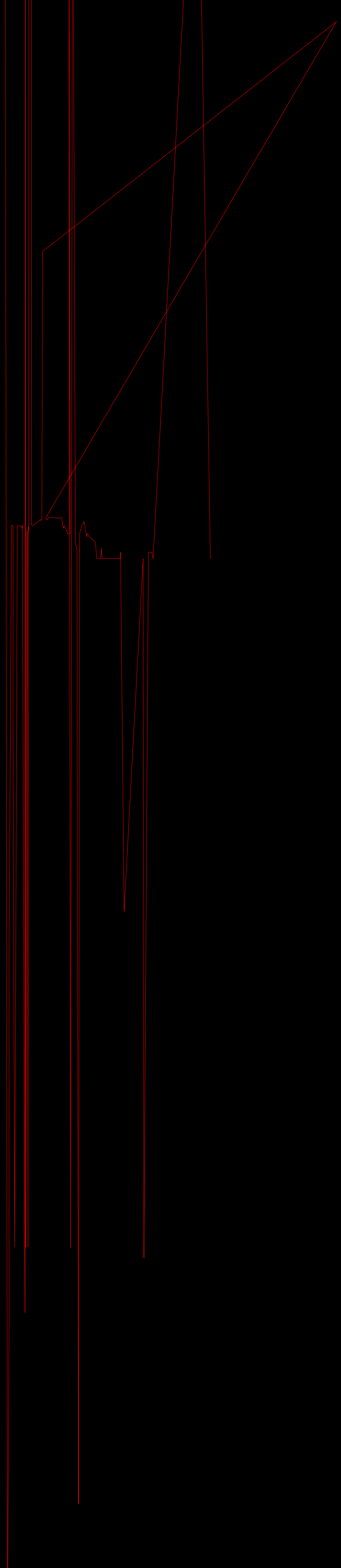
The raw data are stored in a buffer and are written to a set of files when Totem and the CORBA client exit. The raw data files are processed offline to produce the pdfs. The resolution for the pdf calculation is set to 1μsec. The cost of each clock reading, *i.e.*, `gettimeofday()` is about 0.5μsec, which contributes insignificant overhead to the latency.

5.3 Results and Discussion

The measured token circulation time for an idle network is about 205μsec determined by the value at the peak probability density, which corresponds to about 51μsec per token passing time T_0. The value of T_0 is not sensitive to the network traffic because the token message is small (40Bytes) if there are no retransmissions, and it is sent point-to-point separately from normal user traffic. We regard T_0 as a constant in the discussions below.

In the following subsections, we report the measurement results for the end-to-end latency and discuss the de-


Probability Density
node$_1$ runs primary
node$_2$ runs primary
10^0
10^{-1}
10^{-2}
10^{-3}



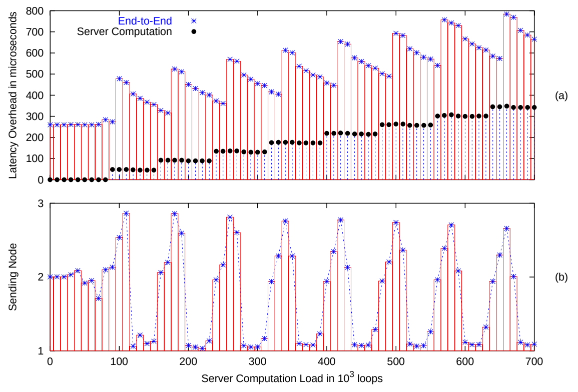


Figure 4. (a) For active replication, the runtime overhead of the end-to-end latency and computation overhead at the server, for different server computation loads. (b) The sending node position for the corresponding measurements given in (a) at different server computation loads.

pending on the position of the primary. For active replication and semi-active replication, where $node_2$ runs the primary, the end-to-end runtime overhead starts at 70% and reduces to 50% as the server computation load increases.

The oscillation of the total runtime overhead is a reflection of the message-send delay pattern. As can be seen from Figure 4(a), the message-send delay can constitute a significant source of the end-to-end runtime overhead. The oscillation amplitude for the runtime overhead, and also the oscillation period, are about one complete token circulation time. Because active replication automatically selects the server replica at the most favorable position to send reply messages, it minimizes the server-side send delay. Thus, with active replication, the client-side send delay dominates the total message-send delay. This is true regardless of the distribution of the "think" time at the client. The client-side send delay, in turn, depends on the selection of the sending node for the different server processing times, as shown in Figure 4(b).

To obtain the sending node information for each reply message, we assign each server replica the node number on which it runs, *e.g.*, the replica that runs on $node_2$ is assigned replica number 2. This number is piggybacked in the reply payload for every reply message that the server replica sends. The client records the number of non-duplicate messages sent by each server replica. The overall sending node position is calculated as the position of the sending node, weighted by the probability that the node transmits the reply message, as shown in Figure 4(b). The oscillation period is about one complete token circulation time (*i.e.*, 205μsec).

6 CONCLUSION AND FUTURE WORK

We have performed in-depth analysis and measurement of the end-to-end latency of a fault-tolerant CORBA infrastructure, with focus on the latency profiles at peak probability densities in terms of different replication styles, the positions of the primary replica on the ring, the client remote invocation patterns and the server processing times. We have shown that different choices of the position of the primary replicas for semi-active replication, for the same server processing load, can differ by a complete token circulation time, which can become significant as the ring size increases. The cost of each message-send is about 270μsec on average. Therefore, for active replication, the presence of duplicate messages can adversely affect the end-to-end latency. By effectively suppressing the duplicates at the sender, active replication exhibits advantages over passive and semi-active replication; in particular, the message-send delay is minimized at the server replica as a result of the automatic selection of the sending node in the most favorable position. Future work includes the characterization of the end-to-end latency in larger-scale networks, on multi-processor machines and with multiple LANs where the Totem multi-ring protocol is used.

References

[1] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal and P. Ciarfella, "The Totem single-ring ordering and membership protocol," *ACM Transactions on Computer Systems* 13, 4 (November 1995), 311-342.

[2] R. K. Budhia, L. E. Moser and P. M. Melliar-Smith, "Performance engineering of the Totem group communication system," *Distributed Systems Engineering* 5, 2 (June 1998), 78-87.

[3] M. Cukier, J. Ren, C. Sabnis, W. H. Sanders, D. E. Bakken, M. E. Berman, D. A. Karr and R. Schantz, "AQuA: An adaptive architecture that provides dependable distributed objects," *Proceedings of the IEEE 17th Symposium on Reliable Distributed Systems*, West Lafayette, IN (October 1998), 245-253.

[4] P. Felber, R. Guerraoui and A. Schiper, "The implementation of a CORBA object group service," *Theory and Practice of Object Systems* 4, 2 (1998), 93-105.

[5] H. Karl, M. Werner and L. Kuttner, "Experimental investigation of message latencies in the Totem protocol in the presence of faults," *IEE Proceedings–Software* 145, 6 (December 1998), 219-227.

[6] F. Kuhns, C. O'Ryan, D. C. Schmidt, O. Othman and J. Parsons, "The design and performance of a Pluggable Protocols Framework for Object Request Broker middleware," *Proceedings of the IFIP Sixth International Workshop on Protocols for High-Speed Networks*, Salem, MA (August 1999), 81-98.

[7] S. Landis and S. Maffeis, "Building reliable distributed systems with CORBA," *Theory and Practice of Object Systems* 3, 1 (1997), 31-43.

[8] C. Marchetti, M. Mecella, A. Virgillito and R. Baldoni, "An interoperable replication logic for CORBA systems," *Proceedings of the International Symposium on Distributed Objects and Applications*, Antwerp, Belgium (September 2000), 7-16.

[9] L. McVoy and C. Staelin, "lmbench: Portable tools for performance analysis," *Proceedings of the Winter 1996 USENIX Conference*, San Diego, CA (January 1996), 279-284.

[10] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia and C. A. Lingley-Papadopoulos, "Totem: A fault-tolerant multicast group communication system," *Communications of the ACM* 39, 4 (April 1996), 54-63.

[11] L. E. Moser, P. M. Melliar-Smith and P. Narasimhan, "Consistent object replication in the Eternal system," *Theory and Practice of Object Systems* 4, 2 (1998), 81-92.

[12] P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, "Message packing as a performance enhancement strategy with application to the Totem protocols," *Proceedings of IEEE Global Telecommunications Conference GLOBECOM'96* 1, London, UK (November 1996), 649-653.

[13] P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, "Strong replica consistency for fault-tolerant CORBA applications," *Proceedings of the IEEE 6th Workshop on Object-Oriented Real-Time Dependable Systems*, Rome, Italy (January 2001), 10-17.

[14] B. Natarajan, A. Gokhale, S. Yajnik and D. C. Schmidt, "DOORS: Towards high-performance fault-tolerant CORBA," *Proceedings of the International Symposium on Distributed Objects and Applications*, Antwerp, Belgium (September 2000), 39-48.

[15] Object Management Group, The Common Object Request Broker: Architecture and Specification (2.4 edition). OMG Technical Committee Document (formal/2001-02-33) (February 2001).

[16] Object Management Group. Fault Tolerant CORBA (final adopted specification). OMG Technical Committee Document (ptc/2000-04-04) (April 2000).

[17] D. Powell, "Delta-4: A Generic Architecture for Dependable Distributed Computing," Springer-Verlag, 1991.

[18] E. Thomopoulos, L. E. Moser and P. M. Melliar-Smith, "Analyzing and measuring the latency of the Totem multicast protocols," *Computer Networks: The International Journal of Computer and Telecommunications Networking* 31, 1-2 (1999), 59-78.

[19] Vertel Corporation, e*ORB C++ User Guide, December 2000.

[20] W. Zhao, L. E. Moser and P. M. Melliar-Smith, "Design and implementation of a pluggable Fault Tolerant CORBA infrastructure," *Proceedings of the International Conference on Parallel and Distributed Systems*, Fort Lauderdale, FL (April 2002).

Wenbing Zhao is completing the PhD in Electrical and Computer Engineering at the University of California, Santa Barbara. He received the Bachelor of Science (1990) and Master of Science (1993) degrees in Physics from Peking (Beijing) University, and the Master of Science degree in Electrical and Computer Engineering (1998) from the University of California, Santa Barbara. He has published nearly 20 journal publications in the fields of superconducting materials and quantum optics, and nearly 10 conference and journal publications in the fields of fault tolerance and distributed systems.

Louise E. Moser is a professor in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. She has served as an associate editor for *IEEE Transactions on Computers* and an area editor for *Computer* magazine in the area of networks. Her research interests include distributed systems, computer networks and software engineering. She received a Ph.D. in Mathematics from the University of Wisconsin, Madison.

P. M. Melliar-Smith is a professor in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. His research interests span the areas of high-speed communication networks and protocols, distributed systems and fault tolerance. He received a Ph.D. in Computer Science from the University of Cambridge, England.