Transaction Abort Rate Reduction with Prioritized Atomic Multicast Protocols

Emili Miedes and Francesc D. Muñoz-Escoí

Instituto Tecnológico de Informática
Universitat Politècnica de València
Valencia, SPAIN

{emiedes,fmunyoz}@iti.upv.es

Technical Report ITI-SIDI-2012/008

Transaction Abort Rate Reduction with Prioritized Atomic Multicast Protocols

Emili Miedes and Francesc D. Muñoz-Escoí

Instituto Tecnológico de Informática
Universitat Politècnica de València
Valencia, SPAIN

Technical Report ITI-SIDI-2012/008

e-mail: {emiedes,fmunyoz}@iti.upv.es

July 5, 2012

Abstract

Prioritized atomic multicast is a variant of the well-known atomic multicast problem that consists in delivering messages according to the total order guarantee while ensuring that the priorities of the messages are considered, this is, messages with higher priorities are delivered first. That service can be used in multiple applications. This paper shows that prioritized atomic multicast protocols may reduce the transaction abort rates in applications that use a replicated database system. Such reduction depends on the message sending rate.

1 Introduction

An *atomic multicast* message delivery protocol, also known as *total order* protocol is a basic group communication building block that can be used to design and build complex distributed applications. Such a protocol enables an application to send messages to a set of nodes such that they are delivered in the same order by each node. Atomic multicast has been studied for more than thirty years, during which a large amount of results has been produced [2, 3]. Some of these services offer an additional feature that enables users to prioritize the delivery of certain messages over others [15, 14, 13].

Our research group has also produced some results [8, 10, 9, 11, 7] related to prioritized atomic multicast, that are briefly described in Section 2. Specifically, in [10] an experimental study shows the effectiveness of the *prioritization techniques* proposed in [8], reducing the abort rates of transactions being served by a replicated database system whose schema had different integrity constraints. To this end, our system prioritized the delivery of writesets belonging to transactions that do not violate such constraints. In this paper a complementary experimental study is performed to analyze the impact that the sending rate has on the *behavior* of the prioritization techniques.

The rest of the paper is structured as follows. Section 2 reviews our previous work regarding prioritization. Section 3 describes the system model being assumed. Section 4 presents the experimental study that has been performed to analyze the impact the sending rate has in the evaluation of the application constraints. The paper is concluded in Section 5.

2 Prioritized Atomic Multicast

Starting from the review of total order protocols given in [3], several ways to *transform* a regular atomic multicast protocol into a *prioritized* one were identified in [8]. This generated a basic set of *prioritization techniques*.

An analysis of the effectiveness of these techniques was given in [10]. That paper shows that a prioritized atomic multicast protocol may reduce the overall transaction abort rate when the underlying database manages integrity constraints. To this end, a proper assignment of the priorities of the messages that propagate the transaction writesets can minimize the number of transactions that violate those constraints.

A second experimental study was presented in [9], assessing how *expensive* these techniques are. To this end, different regular total order protocols and their corresponding prioritized versions were compared, measuring the overhead imposed by these techniques in terms of processor time and main memory. The results showed that prioritization techniques do not impose significant overheads.

Finally, a switching infrastructure was designed [11] and evaluated [7], allowing the dynamic (i.e., at run-time) exchange of total-order multicast protocols (either prioritized or not). Thus, an application may choose at any time which is the most appropriate protocol to be used according to the current system state (amount of concurrent senders, sending rate, target delivery time, ...). The study also proved that the switching mechanism does not block nor slow down the flow of messages delivered to the application.

3 System Model

The system considered is composed of a set of physical nodes. In each node, a process is run. Processes communicate through message passing by means of a *fair lossy channel* (i.e., a channel that may lose some messages, but not all the messages; moreover, it does not produce new spurious messages, does not duplicate messages, and does not change their contents).

Each node has a multilayer structure. The user level is represented by a distributed client application that uses the services offered by a group communication system (GCS), that is composed of one or more group communication protocols (GCP). The GCP providing atomic multicast is placed on top of a reliable message transport.

The system is partially synchronous [4]. Although several definitions exist on partial synchrony, it is considered that on the one hand, processes run on different physical nodes and the drift between two different processors is not known. On the other hand, the time needed to transmit a message from one node to another is bounded but the bound is not known. In practice, the system does not need more synchrony than that offered by a conventional network which offers a reasonably bounded message delivery time.

Processes can fail due to several reasons (for instance, hardware failures, software bugs or human misoperation). Processes are also subject to network failures that keep them from sending or receiving messages. Network partitions may also occur. Nevertheless, since this work focuses on prioritization techniques, these issues will not be addressed here since prioritization is unrelated to fault managing. An implementation of these techniques may rely on some mechanisms (like failure detectors, membership services, message stability criteria, etc.) regularly used by the GCS in order to deal with failures.

4 Experimental Study

This section presents the experimental work that has been done in order to analyze the impact the sending rate has on the evaluation of the application constraints. First, the testbed, the parameters and the methodology are described. Then, the results are explained.

4.1 Testbed

The study uses a test application that relies on the services of a total order protocol which uses a reliable transport layer that was implemented on top of the JBoss Netty 3.2.4 networking library [5]. Netty is a library that offers asynchronous event-driven abstractions for using I/O resources. Netty allowed us to build a reliable, stream oriented, TCP-like message transport layer used by the group communication protocols to unicast and broadcast messages.

The experiments have been conducted in a system of four nodes with an Intel Pentium D 925 processor at 3.0 GHz and 2 GB of RAM, running Debian GNU/Linux 4.0 and Sun JDK 1.5.0. The nodes are con-

nected by means of a 22-port 100/1000Mbps DLINK DGS-11224T switch that keeps the nodes isolated from any other node, so no other network traffic can influence the results.

4.2 Test Application

The `BalanceTest2` test application being used for these tests is very similar to the `BalanceTest` application developed in [10]. It simulates a system that keeps track of the overall amount of money being processed by all investment brokers of a stock trade enterprise. Each broker runs its own instance of the application, operating on the stock exchange on behalf of the stock owners and a potentially large number of investors.

When a broker performs some operation, the application attempts to apply the requested updates to the global balance. If the operation implies the purchase of shares, the application checks whether it can be performed, considering the price of the purchase and the current global balance. The application rejects an operation when the price of the purchase exceeds the global balance.

As there are several brokers working at various sites buying and selling shares concurrently, the global balance is incessantly updated. In order to ensure that the current value of the global balance is consistent among all nodes of the application, a total order protocol is needed. It is used by all nodes to multicast the updates so that all brokers see the same sequence of operations and apply the same sequence of updates to the global balance. That way, consistency among all nodes at each moment is achieved.

Each node creates and broadcasts a number of messages, each one representing a stock trading operation that may update the current balance. Each update carries an integer value. Positive and negative values represent selling and buying operations of stock trading, respectively. The values range from -1500 to 1000. The actual value assigned to each message is generated at random.

All messages are multicast to all nodes using a total order protocol, so all messages are delivered by all nodes in the same order. Nodes apply messages in their delivery order. To apply a message means to update the local copy of the global balance, as kept by each node.

Each message carries a second integer value which represents its priority. In real-life stock trading, these priorities are determined by considering a large number of factors, such as the market situation, recent evolutions of shares, some long-term trends, risk analyses, expected benefits, etc. To simplify the test process, the priority of each operation is uniquely determined by its type (purchase or sale), as follows. Given the value v of an operation, its priority p is computed as $p = 1000 - v$. Thus, a sale update of the global balance with a value of 1000 obtains the priority value 0, and a purchase update with a value of -1500 obtains priority 2500. Since priority management in the modified total order protocols is implemented according to a *lower value = higher priority* rule, the priority of the first update is higher than that of the second one. So, positive updates (from sales) are prioritized over negative updates (from purchases).

This system implemented an integrity constraint for discarding updates that would overdraw the balance. For each negative update request, the presumptive new balance is computed. If it is greater or equal to zero, then the update is applied. Otherwise, the update is discarded. Thus, the global balance is prevented from ever being in the red.

4.3 Test Methodology

The expected behavior of a `BalanceTest2` execution is different for the conventional and the prioritized protocol versions. For the former, the nodes apply approximately 2/5 of positive (sale) updates and 3/5 of negative (purchase) updates. For the latter (prioritized) version, positive updates (i.e., sales transactions) are prioritized, as already stated. This means that the balance is more likely to increase than to decrease, thus less purchase transactions will be discarded.

To test the proposed prioritization techniques, different protocols are compared. For each protocol, the sending rate at which each node broadcasts messages is varied, as discussed in Section 4.4. For each case, `BalanceTest2` is executed, recording the number of updates each node discarded. Then, the percentage of messages that a node has discarded is computed.

For obtaining reliable results, each execution of `BalanceTest2` has been repeated a statistically relevant number of times; i.e., until the standard deviation was lower than 1.5% of the mean.

4.4 Parameters

This section describes the values of the test parameters. First, a group of *fixed* parameters is presented, whose values are the same for all tests, and then a group of *variable* parameters.

Each `BalanceTest2` instance is run in a physical node. Each instance creates a sequence of messages, as described above, and sends them by a rate that is constant during all the test. Each instance is configured to receive 10000 messages. Each message is tagged with a priority value ranging between -1500 and 1000, as explained above. The initial balance value is set to 0.

The variable parameters are the protocol type and the sending rate.

Three non-prioritized total order protocols and a prioritized version for each have been compared. The *UB* protocol is an implementation of the UB sequencer-based total order algorithm proposed by [6]. UB stands for *Unicast-Broadcast*. The *TR* protocol implements a token ring-based algorithm, inspired in to the ones of [12] and [1]. Finally, the *CH* protocol is an implementation of the causal history algorithm from [3]. The corresponding prioritized versions are *UB_PRIO*, *TR_PRIO* and *CH_PRIO*.

These tests have been executed using different sending rates: 40, 60, 80, 120 and 140 messages sent per second and per node.

4.5 Results

For each set of executions of a test with a given protocol and sending rate, a mean percentage of discarded messages and the standard deviation *of those percentages* are collected.

The *mean percentages* are shown in Figure 1 and depicted in Figure 2 while the *standard deviation of the percentages* is shown in Figure 3.

	UB	UB_PRIO	TR	TR_PRIO	CH	CH_PRIO
40	25.93	25.80	26.21	26.45	26.41	26.07
60	25.78	25.34	26.06	25.89	25.67	25.30
80	25.72	24.93	25.94	25.73	25.89	25.15
120	25.56	24.65	25.73	24.90	25.87	25.17
140	25.85	24.28	25.61	24.60	25.95	24.56

Figure 1: Percentages of discarded messages (means)

4.6 Discussion

The results presented in Figures 1, 2 and 3 show that the sending rate has an impact on the evaluation of the integrity constraints defined in the test application.

At a glance, it can be seen that the percentage of discarded messages decreases as the sending rate increases. When the sending rate is low, all the protocols can total order the messages very quickly. For instance, in *UB* and *UB_PRIO*, the incoming messages are received by the sequencer and sequenced immediately. This means that *UB_PRIO* has no chance to reorder the incoming messages according to their priorities. A similar situation happens in *TR_PRIO* and *CH_PRIO*. As the sending rate increases, more and more messages are queued in the *incoming* data structures and the prioritized protocols have a higher chance to reorder messages. This means that positive update messages have the chance to advance other negative update messages and therefore, the shared balance has a higher chance to have a value able to accept such negative updates when they are finally delivered.

5 Conclusion

In distributed systems that use replicated databases where integrity constraints are defined, prioritized atomic multicast protocols may be used in order *to reduce the transaction abort rate*. The experimental study performed in [10] showed that this goal is feasible.

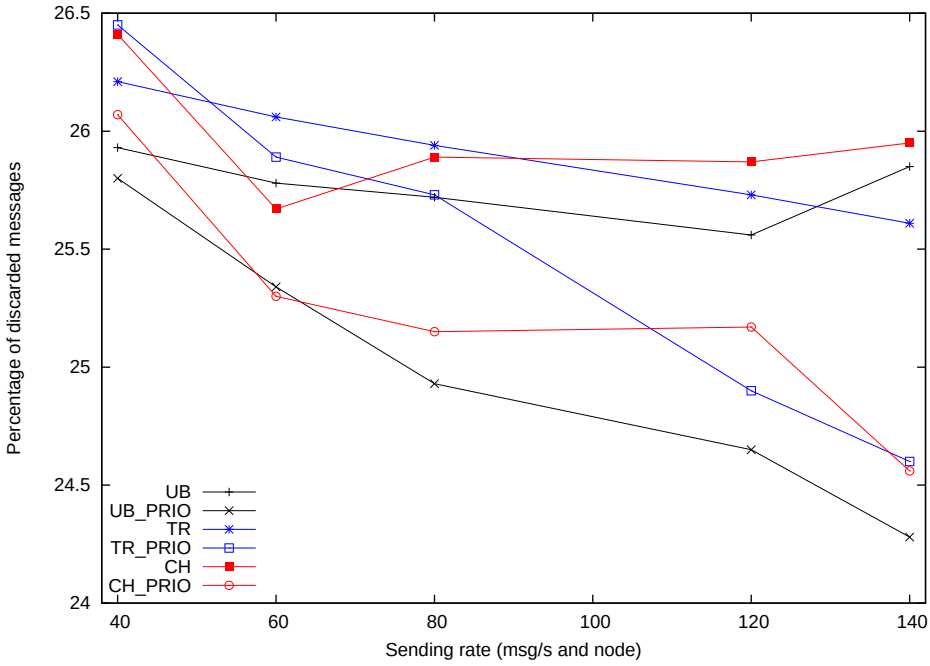

Figure 2: Percentages of discarded messages (means)

	UB	UB_PRIO	TR	TR_PRIO	CH	CH_PRIO
40	0.23	0.36	0.36	0.19	0.39	0.39
60	0.22	0.28	0.21	0.35	0.13	0.21
80	0.06	0.24	0.25	0.21	0.21	0.09
120	0.10	0.30	0.11	0.21	0.22	0.16
140	0.22	0.21	0.13	0.17	0.32	0.32

Figure 3: Percentages of discarded messages (standard deviations)

The reordering achieved by these prioritization protocols strongly depends on the overall length of the message queue being used by the prioritizing component of such protocols. That length directly depends on the message sending rate being supported. The results presented in this paper have shown that an improvement of 5% is achievable (abort rate values of 0.258 vs 0.245) at 560 msg/sec when the non-prioritized and prioritized variants are compared. Note that prioritization does not introduce any advantage at low sending rates (i.e., with global values below 40 msg/sec) and that this 5% improvement is a direct consequence of the higher sending rates and their effects on the reordering queue length.

At low sending rates other complementary approaches are needed in order to guarantee a minimal length in the reordering queue being used by the prioritizing component of the atomic multicast protocol. One option is to temporarily block message broadcasting until the sending queue is large enough (in the privilege-based or sequencer-based multicast protocols) or to temporarily block delivery in the receiving queue (in the causal-history protocols), but such approaches will increase the message propagation and delivery time being perceived by the application users.

References

[1] Yair Amir, Claudiu Danilov, and Jonathan Robert Stanton. A low latency, loss tolerant architecture and protocol for wide area group communication. In *Intnl. Conf. on Depend. Syst. and Netw. (DSN)*,

pages 327–336, Washington, DC, USA, 2000. IEEE-CS.

[2] Gregory Chockler, Idit Keidar, and Roman Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001.

[3] Xavier Défago, André Schiper, and Péter Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421, 2004.

[4] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, January 1987.

[5] JBoss. JBoss Netty, 2012. http://www.jboss.org/netty.

[6] M. Frans Kaashoek and Andrew S. Tanenbaum. An evaluation of the Amoeba group communication system. In *Intnl. Conf. on Distrib. Comput. Syst. (ICDCS)*, pages 436–448, Washington, DC, USA, 1996. IEEE-CS.

[7] Emili Miedes. *Prioritized Atomic Multicast Protocols*. PhD thesis, Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València, 2012.

[8] Emili Miedes and Francesc D. Muñoz-Escoí. Managing Priorities in Atomic Multicast Protocols. In *Intnl. Conf. on Avail., Reliab. and Security (ARES)*, pages 514–519, Barcelona, Spain, March 2008.

[9] Emili Miedes and Francesc D. Muñoz-Escoí. On the cost of prioritized atomic multicast protocols. In *Intnl. Symp. on Distrib. Obj., Middleware and Appl. (DOA)*, volume 5870 of *Lect. Notes Comput. Sc.*, pages 585–599. Springer, Vilamoura, Portugal, November 2009.

[10] Emili Miedes, Francesc D. Muñoz-Escoí, and Hendrik Decker. Reducing Transaction Abort Rates with Prioritized Atomic Multicast Protocols. In *Intnl. Euro. Conf. on Paral. and Distrib. Comput. (Euro-Par)*, volume 5168 of *Lect. Notes Comput. Sc.*, pages 394–403. Springer, Las Palmas de Gran Canaria, Spain, August 2008.

[11] Emili Miedes and Francesc D. Muñoz-Escoí. Dynamic switching of total-order broadcast protocols. In *Intnl. Conf. on Paral. and Distrib. Proces. Tech. and Appl. (PDPTA)*, pages 457–463, Las Vegas, Nevada, USA, July 2010. CSREA Press.

[12] Louise E. Moser, P. Michael Melliar-Smith, Deborah A. Agarwal, R.K. Budhia, and C.A. Lingley-Papadopoulos. Totem: a fault-tolerant multicast group communication system. *Commun. ACM*, 39(4):54–63, April 1996.

[13] Akihito Nakamura and Makoto Takizawa. Priority-based total and semi-total ordering broadcast protocols. In *Intnl. Conf. on Distrib. Comput. Syst. (ICDCS)*, pages 178–185, Yokohama, Japan, June 1992.

[14] Luís Rodrigues, Paulo Veríssimo, and Antonio Casimiro. Priority-based totally ordered multicast. In *Wshop. on Alg. and Arch. for Real-Time Control (AARTC)*, Ostend, Belgium, May 1995.

[15] Alan Tully and Santosh K. Shrivastava. Preventing state divergence in replicated distributed programs. In *Intnl. Symp. on Reliab. Distrib. Syst. (SRDS)*, pages 104–113, Huntsville, Alabama, USA, oct 1990.