Highly Available Trading System: Experiments with CORBA

Xavier Défago, Karim R. Mazouni, André Schiper
Département d'Informatique, École Polytechnique Fédérale
CH-1015 Lausanne, Switzerland.
Tel +41 21 693 4240. Fax: +41 21 693 6770
email: {Xavier.Defago,Karim.Mazouni,Andre.Schiper}@epfl.ch

Abstract

The Swiss Exchange system (SWX system) was the first stock exchange system in service to be fully computerised. For high availability, the trading system is built as a replicated service based on Isis. For portability reasons, the SWX team has considered basing the next version of the trading system on CORBA. Despite the numerous advantages of a CORBA based solution, it was necessary not only to meet the functionality requirements of the system, but also to evaluate the performance of the chosen middleware.

This paper describes a model that simulates the communication behavior of the trading system on Iona's Orbix and OrbixTalk. We have evaluated this model in order to illustrate some of the performance limitations of this communication infrastructure. We have also studied the fault-tolerance of the system and found that a key aspect in such a system is the state transfer. We have extended our model to support different schemes for state transfer. Measurements have shown us the conditions under which a state transfer could be performed in the background, concurrently with the normal operations.

Keywords

CORBA, event service, IP-multicast, performance prediction, state transfer, high availability

1 INTRODUCTION

Back in 1996, the Swiss Exchange system (SWX system) was the first stock exchange system in service to be fully computerised (Piantoni *et al*. 1997). The architecture of the system is based on two main components, the *exchange system* (ES) and the *trading system* (TS) (see Figure 1). The exchange system is unique and runs at the SWX headquarters in Zurich. The TS is run by SWX clients, and acts like a cache, extended with local information. For higher availability, the TS is built as a replicated service based on Isis (Birman *et al*. 1993). After one year of activity, the proprietary aspect of Isis, as well as some scalability problems, have led SWX to consider basing the next version of the TS on CORBA (OMG 1995). In addition, it was observed that the current system makes no use of the virtual synchrony provided by Isis.

In a collaboration with SWX, we have developed a model of the TS based on Iona's Orbix (IONA 1996*a*), and evaluated the performance of the communications. Our role was twofold: 1) to investigate the performance of Orbix, and 2) to improve the fault-tolerance of the system by working on the state transfer mechanism.

Concerning 1), one of the most critical part of the TS regarding the performance and scalability of communication is the broadcasting of quotes to the traders. For scalability reasons, we have built this mechanism on top of OrbixTalk (IONA 1996*b*), an implementation of the event service (OMG 1996) based on IP-multicast and a negative acknowledgement protocol.

Concerning 2), since the TS is a replicated service, an important problem is the recovery of a replica after a crash. When a crashed replica recovers, it needs to update its state according to the current state of the system. This is done by transferring the state of an up-to-date replica to the recovering replica. The mechanism that Isis provides for implementing the state transfer was found to be much too costly. Furthermore, the Isis mechanism for state transfer blocks the whole replicated service until the state has been transferred. As a result, in the current implementation the state transfer has to be done off-line, during the night. In the new TS, one of the requirements is to be able to perform an on-line state transfer. We designed an adequate mechanism that is compatible with this requirement, and evaluated its performance.

The rest of the paper is structured as follows. In Section 2, we give an quick overview of the Swiss Exchange System, with a clear emphasis on the TS. In Section 3, we present the environment in which our measures have been performed. We also show the architecture of the TS in this environment. In Section 4, we answer some important questions about the performance limits of the system. In Section 5, we illustrate the problems that we encountered when designing the on-line state transfer mechanism. Finally, Section 6 concludes the paper.

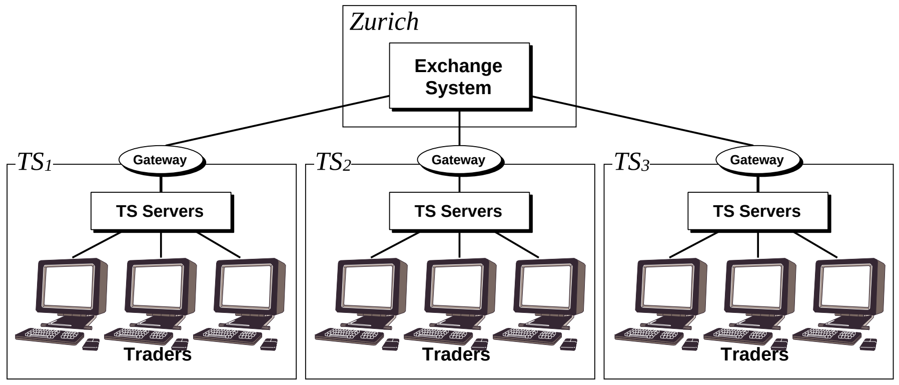

Figure 1 Architecture of the Swiss Stock Exchange.

2 THE SWISS STOCK EXCHANGE

The architecture of the SWX system is described in (Piantoni *et al*. 1997). We give here a brief overview of this architecture (see Figure 1). The Exchange System (ES) is the core of the SWX system, and is based in Zurich. It allows trading against a central order book. Market changes are interactively forwarded to all members.

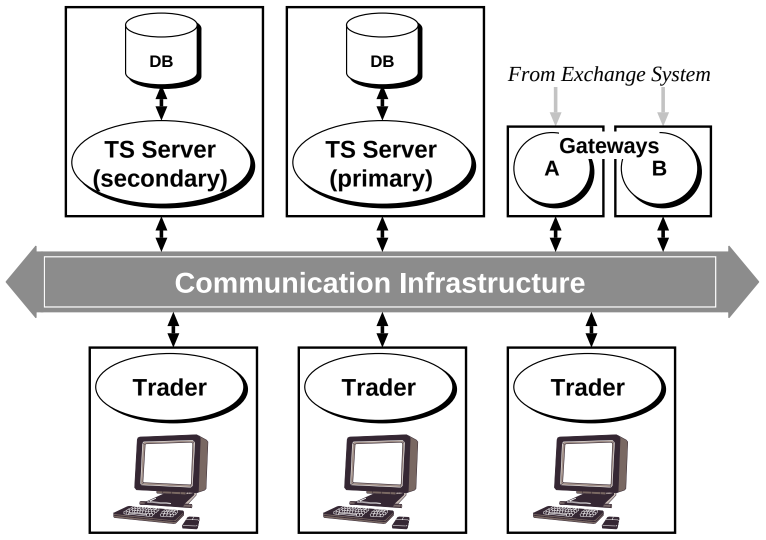

Figure 2 Architecture of the Trading System.

The SWX clients receive information issued by the ES, through a (possibly replicated) gateway. Each SWX client runs a TS which caches the information coming from the gateways. The TS is an intermediary between the traders and the ES, and allows as well traders to interact with each other.

The architecture of the TS is represented in Figure 2. The communication infrastructure makes it possible for the different components of the TS to communicate with each other. The information coming from the ES is received by the gateways, which forward the information to the primary TS server. The primary TS server informs the secondary TS server, stores the information on stable storage, and broadcasts it to the traders.

3 CONTEXT OF OUR EXPERIMENTS

3.1 Trading System Model

We have modelled the different entities of the TS as objects, represented in Figure 3. The *traders* represent the different objects that receive information from the TS server and issue information to the TS server. The TS server is replicated. The *primary server* is the active part of the TS. First, it receives messages from the Exchange System: information coming from the ES is generated in our model by the *source* (see Figure 3, arrow 1). Second, it broadcasts the relevant information changes to the traders (see Figure 3, arrow 2), and it keeps information on stable storage. The *secondary server* has a passive role. Similar to the primary, it keeps persistent information, but it does not broadcast any information to the traders. Arrow 3 in Figure 3 represents the state transfer and is discussed in Section 5.

3.2 Hardware and Software Environment

We have implemented our model using Orbix and OrbixTalk. One of the main reason to choose OrbixTalk, was the IP-multicast on which it is based, as well as the negative acknowledgement protocol that it implements. By combining these two characteristics, we have found the system to be scalable in the context of a local area network (LAN). The role of the negative acknowledgement protocol that OrbixTalk implements is to increase the reliability of the communication. OrbixTalk retransmits messages in a best-effort manner; a receiver which detects that a message could not be delivered is notified by an exception.

The environment in which the measures have been performed is a cluster of workstations. The cluster consists of Sun's SPARC stations (Sparc-20 50 MHz and Ultra Sparc-1 167 MHz) running Solaris 2.5.1, and interconnected with a 10 Mbits Ethernet.

The measures that we present in Section 4 and Section 5 were performed with OrbixTalk 1.04, and Orbix 2.1MT. Earlier versions of OrbixTalk had very limited

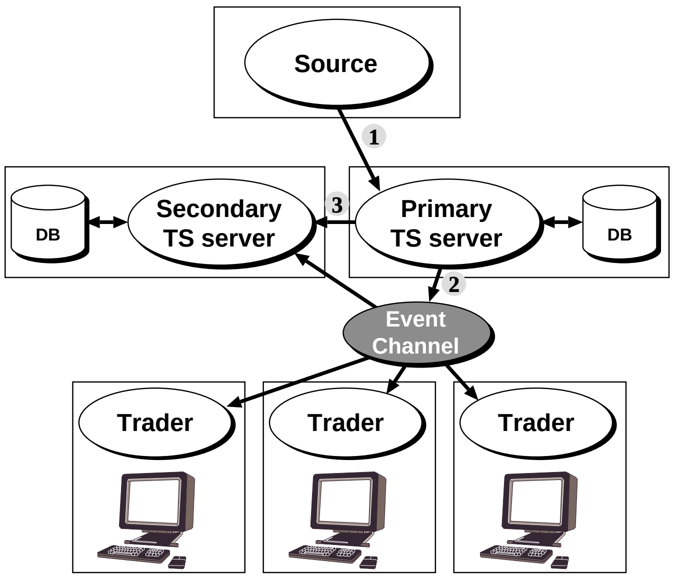

Figure 3 Model of the Trading System.

performance with respect to our needs. With Orbix 1.04, the performance is satisfactory.

4 INFORMATION FLOW BETWEEN THE PRIMARY SERVER AND THE TRADERS

One of the most critical performance aspect of the TS is the flow of information between the primary server and the traders. Since the expected throughput of messages emitted by the primary server is more than 200 Kb/s and the number of traders can be over a hundred, the scalability of the communication is of great importance: these figures are expected to increase in the future.

The main reason for building our prototype was to answer questions related to the design of the TS.

- *Is OrbixTalk scalable with respect to the number of traders?*
- *What is the effect of a slow trader on the overall system?*
- *What throughput can be reached using an OrbixTalk event channel?*
- *What about using two OrbixTalk event channels in parallel?*

4.1 Measures

We have considered a source (Figure 3) that emits messages at a given *target throughput*. These messages are received by the primary TS server, which emits them to the traders through an OrbixTalk event channel. A dozen of traders where receiving those messages. We measured the rate at which messages where actually emitted by the primary, and the rate at which they were received by the traders. Our results show the *instantaneous throughput*, which is the throughput achieved at every single instant in time.

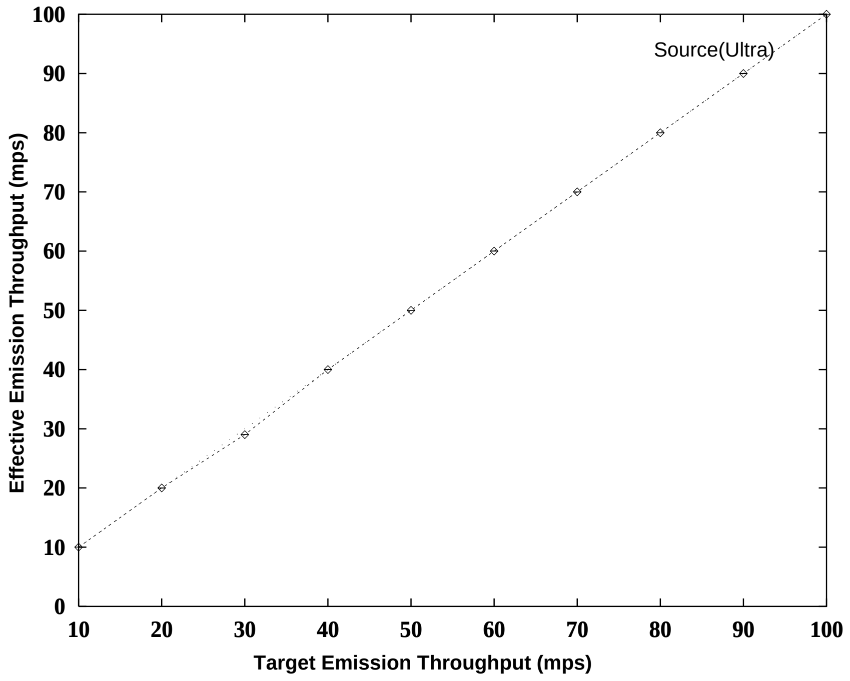

Figure 4 Effective emission throughput.

The size of the messages is 8 Kb. Associated with each incoming message, the traders perform a 5 ms processing. The primary TS server was run on an UltraSparc workstation. The traders run on a group of Sparc-20, and UltraSparc workstations. There is exactly one trader on each workstation and a total of 12 traders.

We measured the effective emission throughput with increasing emission rates. These measures are illustrated in Figure 4. We see that the source can emit messages at any required rate between 10 and 100 messages per second. Measures performed under the same conditions, with earlier versions of OrbixTalk could not go beyond 70 messages per second.

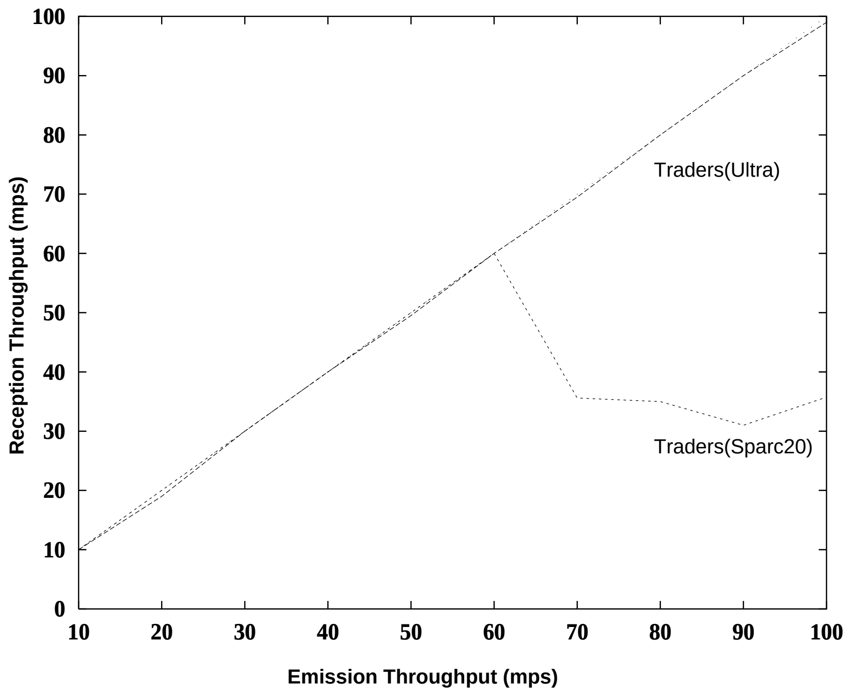

Figure 5 Emission vs. reception throughput.

Figure 5 represents the evolution of the reception throughput, as the emission throughput increases. When the traders are not overloaded, the reception throughput is equal to the emission throughput. Nevertheless, Figure 5 shows that some traders cannot cope with a rate higher than 60 messages per second (the traders that run on a Sparc-20 workstation). In that case, the reception throughput is lower than the emission throughput. As a result, an increasing number of messages get stored into buffers, and eventually, messages have to be dropped. Figure 5 shows different reception throughput for the two types of workstations. This shows that the actual bottleneck is not the network, but rather the software layers that handle the reception of messages.

Figure 6 represents the temporal evolution of the instantaneous throughput at a target emission rate of 70 messages per second. Figure 6 shows a strong disparity between platforms. A superposition of the curves of both the source and the fast traders (Ultra-1) in Figure 5, shows that the fast traders can cope with a rate of 70 messages per second. However, the traders that run on the Sparc-20 workstations can clearly not cope with a rate of 70 messages per second. The temporal behavior of the reception throughput gives a good indication of what may happen here. The traders receive messages at a rate they cannot handle, i.e. the reception buffers overflow. When this

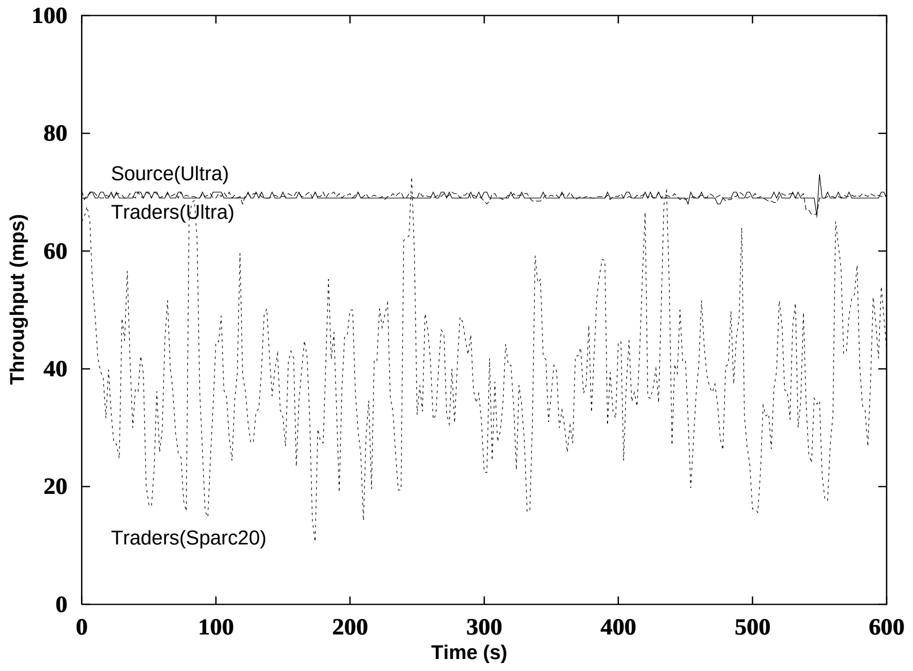

Figure 6 Emission throughput of 70 msg/s.

situation arises, OrbixTalk drops a bunch of messages to have a chance to catch up with the reception rate. This behavior confers a 'saw-like' aspect to the curve.

Initially, we thought that the emission of messages would be a bottleneck in the TS. We were surprised to see that the actual bottleneck lies in the reception of messages. This is probably due to the higher cost of *unmarshaling* of parameters compared to the cost of *marshaling* of parameters.

To sum up, the performance of the system is limited by the performance of the traders (the bottleneck is located at the reception). We observe a great disparity between traders, depending on the computational performance of the machine on which they are running.

4.2 Discussion

The analysis of the measures that we have gathered from our experiments allows us to answer our questions. These questions about performance are quite fundamental when it comes to consider building a demanding application on a new platform. Knowing the limits of such a platform is almost as important as the capabilities since it determines whether the non-functional requirements of the application can be reached.

- *Is OrbixTalk scalable with respect to the number of traders?*
  Among the measures that we have performed on OrbixTalk (these measures are not presented here), we have observed that the number of traders does not slow down the emission of messages.
- *What is the effect of a slow trader on the overall system?*
  One of the most interesting aspect of the measures presented in Figure 6 is the fact that slow traders (Sparc-20) do not affect the whole system; the fast traders (Ultra-1) and the source can still sustain the desired rate, although the slow traders cannot. This is confirmed by Figure 5.
- *What throughput can be reached using an OrbixTalk event channel?*
  We have found this question to greatly depend on the CPU power of the workstations. For traders running on Sparc-20, the limit is clearly lower than for traders running on Ultra-1. Indeed, Figure 5 show that slow traders (Sparc-20) stall when the system throughput exceeds 60 messages per second. On the other hand, fast traders (Ultra-1) can still cope with a throughput of 100 messages per second.
- *What about using two OrbixTalk event channels in parallel?*
  In other measures that are not presented here, we have tested other communication architectures, among which two event channels in parallel to see if this could increase the maximum throughput from the primary server to the traders. We have found the maximum throughput not to depend on the number of event channels. In other words, two event channels do not allow for a higher emission throughput than a single one since they compete for the same resources (network, CPU).

5 ON-LINE RECOVERY

In the current TS based on Isis, the state transfer is performed off-line. In other words, when a server crashes, it can only be restarted during the night, when the stock exchange is closed. This leads to two major problems. 1) the reliability is reduced since a second failure in the same day cannot be masked, and 2) it makes it impossible to guarantee 24 hours a day availability as may be required in the future (Piantoni *et al*. 1997).

The reason that led to such a limitation on the Isis based TS has to do with the mechanism that Isis provides for state transfer. During a state transfer Isis requires the whole group to block until the state transfer is finished. Although such a limitation is acceptable for systems with small states, the TS cannot allow it. The state to transfer is quite large (30 Mb or more, depending on the down-time). We have performed further measurements to answer the following question.

- *Is it possible to implement a non-blocking on-line state transfer?*
  A non-blocking state transfer means that, during the state transfer, the primary is not blocked and can still provide its normal service.

To answer this question, we have tried to find the conditions under which a non-blocking state transfer is possible.

5.1 Measures

We have implemented a mechanism that allows us to measure the impact of the state transfer on the normal execution of the TS. Messages are sent to the primary at a constant rate by the source, using CORBA method invocations (Figure 3, arrow 1). The messages are relayed to the traders by the primary, through an event channel (Figure 3, arrow 2 to the traders). In our experiment, the secondary starts after 5 minutes and requests a state transfer from the primary, with a CORBA method invocation (Figure 3, arrow 3). At the same time, the secondary begins to receive messages from the primary, through the event channel (Figure 3, arrow 2 to the secondary).

The tests have been made with the following parameters: the messages are emitted by the source at a rate of 50 messages per second. The state to transfer from the primary to the secondary has a size of 20 Mb. The source, the primary, and the secondary run on UltraSparc workstations. The traders run on Sparc-20, and UltraSparc workstations. There is exactly one trader on each workstation in the cluster, and a total of 10 traders.

In Figure 7, the state is transferred in one block of 20 Mb, with a single Orbix invocation. This transfer starts after about 300 s and is responsible of the black-out period shown in Figure 7. Figure 7 clearly shows that this solution leads to instabilities once the transfer is finished (the ‘saw-like’ aspect of the curves). With a larger state, the resulting instabilities become even stronger. This behavior poses at least two major problems, 1) the unavailability of the system during the state transfer breaks some of the requirements of the TS (Piantoni *et al*. 1997), and 2) if the network is slightly more loaded, the traders may stall and never be able to catch up.

In order to circumvent this problem, we have modified the mechanism for state transfer, to allow the transfer of the state in smaller messages. Further measures where taken, where the 20 Mb state was transferred in 640 chunks of 32 Kb each (i.e., 640 Orbix invocations). Surprisingly, the resulting measures showed exactly the same behavior than a state transfer in one block (see Figure 7).

In Figure 8, the state is also cut into 640 chunks of 32 Kb (for a total of 20 Mb). But, unlike the previous measures, the chunks are sent one after another, with a pause interval of 50 ms between the emission of two chunks. As shown in Figure 8, this solution does not hinder the normal execution, and the flow of messages to the traders is not influenced by the state transfer.

To summarise, our measures show that it is not enough to implement state transfer in a separate thread, but it is also important to split the state into smaller chunks and add a pause between the emission of two chunks.

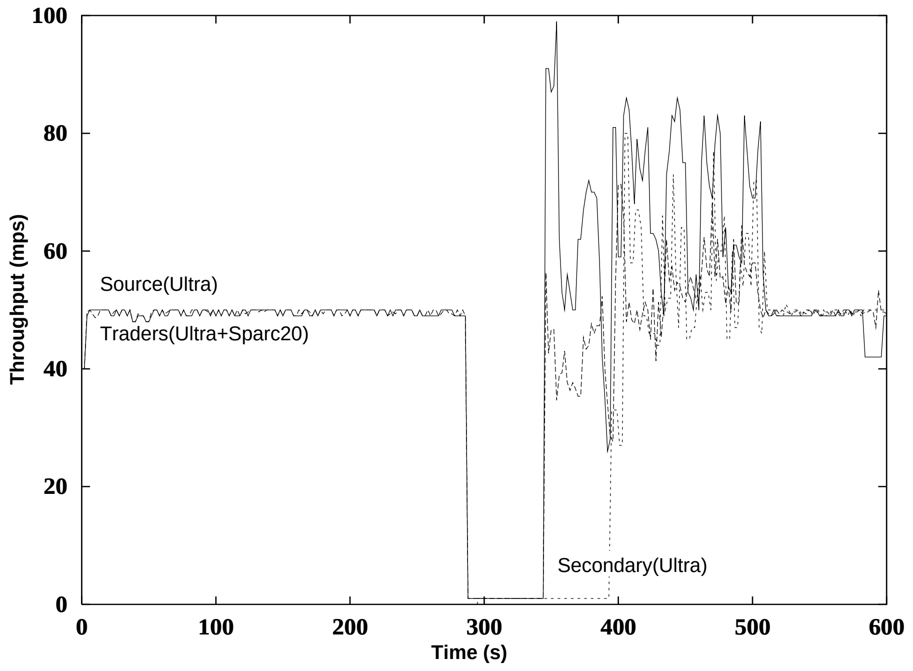

Figure 7 State transfer (1 × 20 Mb).

5.2 Discussion

This simulation of a state transfer gives us invaluable information on the behavior that we can expect from Orbix and OrbixTalk. The problem of a state transfer (or the transfer of a large amount of information) is a common problem in practice, but often neglected. In highly-available distributed applications, a state transfer is needed whenever a secondary server recovers from a crash.

In the context of the TS, the only way to allow an online recovery of the secondary server is to implement a non-blocking state transfer. In other words, if the primary server is blocked for the whole duration of the state transfer, the state transfer has to be made off-line. Among other things, such a restriction makes it impossible to consider running the service 24 hours a day. It also reduces the number of failures tolerated by the system during a single day.

- *Is it possible to implement a non-blocking on-line state transfer?*
  Considering the measures made in Section 5.1, the answer is yes. However, this is only possible under the following conditions.

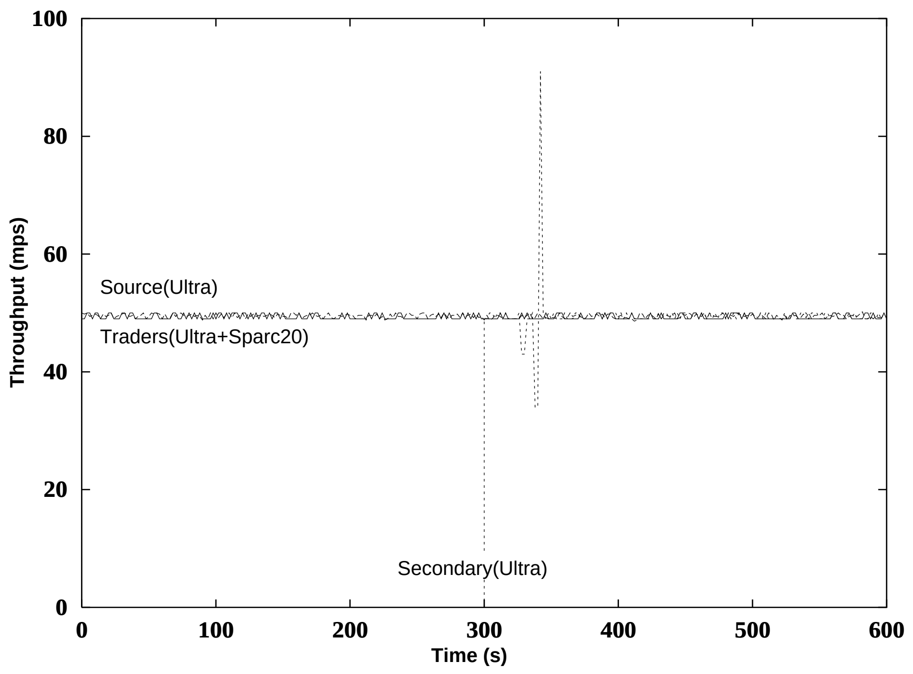

Figure 8 State transfer (640 × 32 Kb).

- The state transfer mechanism is decoupled from the servers; i.e., the state transfer is performed by another thread.
- The state is cut into smaller chunks, with a pause between the emission of each chunk.

6 CONCLUSION

The experiments described in the paper have enabled us to answer some important questions with respect to using a CORBA based communication infrastructure for the TS.

In Section 4, we have investigated the performance aspect of OrbixTalk with respect to the emission of information to the traders. We have seen this performance to greatly depend on the CPU power of the involved machines, rather than the network capabilities. The development of our model has shown us that i) most of the overhead comes from the unmarshaling of parameters, ii) the performance of OrbixTalk has improved between versions, iii) decent performance could only be reached by modifying undocumented parameters, iv) slow receivers do not slow down the whole system.

In Section 5, we have investigated the state transfer aspect of the TS. We have shown that, with an adequate mechanism, a non-blocking state transfer is possible, even if the state to transfer is large. This not only requires to decouple the state transfer from the normal service, but also to split the state into smaller chunks and introduce a pause between the emission of two consecutive chunks. To sum up, the experiments have shown us the problems raised by requirements such as responsiveness, throughput, or scalability, in particular in the context of state transfer. A system with a blocking state transfer such as Isis, is not realistic for applications with a large state. This is not only a practical problem, but it also comes in contradiction with the high availability that such a system is supposed to provide.

ACKNOWLEDGEMENTS

We would like to thank the SWX laboratory and more specifically Rico Piantoni, Constantin Stancescu, and Benoît Jardin.

REFERENCES

Birman, K. and van Renesse, R. (1993), *Reliable Distributed Computing with the Isis Toolkit*, IEEE Computer Society Press.

IONA (1996*a*), *Orbix Programming Guide*, IONA Technologies Ltd.

IONA (1996*b*), *OrbixTalk Programming Guide*, IONA Technologies Ltd.

Object Management Group (1995), *The Common Object Request Broker: Architecture and Specification*, OMG. Revision 2.0.

Object Management Group (1996), *CORBAservices: Common Object Services Specification*, OMG.

Piantoni, R. and Stancescu, C. (1997), Implementing the Swiss Exchange trading system, *in* ‘Proceedings of the 27th International Symposium on Fault-Tolerant Computing’, IEEE Computer Society Press, Seattle, Washington, USA, pp. 309–313.

BIOGRAPHIES

Xavier Défago graduated from the EPFL (Federal Institute of Technology in Lausanne) in 1995. After working for a year on parallel operating systems at NEC C&C Central Research Labs. in Kawazaki (Japan), he joins Prof. André Schiper’s Operating Systems laboratory as a research assistant and a PhD student. His research interests are in the areas of fault-tolerant distributed systems, distributed algorithms, operating systems principles, and real-time systems.

Karim Riad Mazouni received a PhD (EPFL, 1996), an MSc. (Univ. of Paris, 1990) and an engineer’s degree (Univ. of Algiers, 1988) in Computer Science. He spends several years at the EPFL, teaching operating systems, and doing research on fault-tolerant and object-oriented distributed systems. At that time, his research focused on support for replicated objects invocation. In January 1998, he joins the Union Bank of Switzerland (UBS) as a system architect. His current activities at UBS focus on the design of object-oriented distributed services (such as a corporate-wide directory service) that will implement the core functionality of the company's distributed computing platform.

André Schiper has been professor of Computer Science at the EPFL (Federal Institute of Technology in Lausanne) since 1985, leading the Operating Systems laboratory. He has worked on the implementation of communication primitives for the Isis system, developed at Cornell University. His research interests are in the areas of fault-tolerant distributed systems and group communication. In the context of CORBA, his group is currently working on the specification and the implementation of a replication service.

André Schiper is a member of the ESPRIT Basic Research Network of Excellence in Distributed Computing Systems Architectures (CaberNet).