Experimental Performability Evaluation of Middleware for Large-Scale Distributed Systems*

Luís Soares
University of Minho
Informatics Department
Campus de Gualtar, 4710 Braga, Portugal
losoares@di.uminho.pt

José Pereira
University of Minho
Informatics Department
Campus de Gualtar, 4710 Braga, Portugal
jop@di.uminho.pt

Abstract

We present a tool for experimental evaluation of distributed software that enables performability tests to be incorporated in the agile development of complex middleware systems. The proposed approach combines simulation with the profiling of key components and is achieved by an extension to a standard simulation kernel and reflection, thus leveraging existing simulation models and easing the integration of existing components. The evaluation of database replication middleware in large-scale systems is used to illustrate the approach.

1. Introduction

Middleware plays an increasingly important role in current information systems as an ever increasing share of the complexity of large clusters and grid computing systems is encapsulated in a diversity of standard middleware components. Recently, agile development methodologies [8] have been attracting software engineers to focus on modelling and automated testing of compliance. Unit testing means that development starts by producing auxiliary test components directly from the model which are used, as the software product evolves, to ensure that it matches initial modelling.

This methodology is however hard to apply to middleware, where initial modelling focus is on performability of the system and not on independent correctness of each component. Experimental evaluation of such systems for comparison with analytical model is thus confined to testing with the real systems. However, real tests are costly to setup and run and often depend on the availability of the complete target system and realistic workloads and fault-loads. This is especially difficult when the middleware targets large clusters or grid systems, and precludes incremental development and early testing of individual components. Although often used, toy applications and micro-benchmarks are unable to disclose the subtle interactions of the middleware with application semantics and dynamics (e.g. flow control issues and hot-spots) and with the environment (e.g. fault scenarios) as well as introduce significant probe effect.

In this paper we propose that the agile development of complex middleware with performability concerns can be achieved by running selected components with simulation models of realistic environments, workloads, and fault-loads by means of a centralized simulation kernel [4]. This approach is enabled by a small extension to the Scalable Simulation Framework (SSF) specification that greatly simplifies interfacing real implementations within simulation models while accurately reproducing the timing behavior of real systems. The evaluation of database replication middleware in large-scale systems is used to illustrate the approach.

2. Simulation Framework

In this section, we present the design and implementation of an event-driven simulation framework that allows combining real implementations with simulation models. In an event-driven simulation, time is incremented only by scheduling events with non-zero delays. The challenge when mixing real and simulated components is to ensure that the time actually spent executing real code is accurately reflected in simulation time. Thus this approach goes beyond the simple reuse of real code for simulation models and is able to reproduce timing properties of real systems [4].

In detail, this implies starting a profiling timer whenever real code is entered to account for native execution time. When execution re-enters simulation code, the profiling timer is stopped and the elapsed time used as an offset for all events scheduled. Doing this in an *ad hoc* fashion is however a tedious and error prone task, as it must be performed for every interaction between real and simulation code thus forcing

*Partially funded by FCT, project StrongRep (POSI/CHS/41285/2001).

changes to existing simulation models.

In this paper we propose a simulation kernel that eases mixing simulated and real components by automating the accounting of real time. Our proposal extends the Scalable Simulation Framework (SSF) specification for event-driven simulation [6]. In the SSF interface, *entities* encapsulate passive state. Actions on state are encapsulated by *processes*. Actions on different entities are synchronized by exchanging *events*, routed by binding *outgoing channels* to *incoming channels*. Delays are imposed on individual events, but also on channels and channel bindings, allowing the system to partition the simulation and take advantage of parallel processing. Multiple implementations of the SSF interface are available in C++ [1] and Java [3].

The fundamental extension of the SSF interface in our implementation (MinhaSSF) is very simple and consists in being able to designate selected entities as *real-time entities* by overriding the `isRealTime()` method. A process associated with such entity becomes a *real-time process* and behaves as follows: The profiling clock is started when the process reads an event from an incoming channel and stopped whenever the process writes an event to an outgoing channel. Code executed between writing an event and reading another event is therefore not accounted for. After stopping the profiling clock, the real-time process is not rescheduled until simulation time has advanced by as much as the previous elapsed real delay. The event is actually written to the channel only after simulation time has catch-up.

The second extension to the SSF interface is a pair dynamic proxy classes using Java reflection. These can be used to completely separate implementation from simulation models and APIs. The first can be used to transform an invocation made by real code to a pair of event write and read operations, that transparently interface with simulation code. The second does the reverse operation, listening for events, invoking real code and replying using a second channel.

The role of dynamic proxies is better understood with an example. Consider a real component Client that expects a service implementing a given interface Server. One wants to run the Client implementation using a simulation model of Server, named ServerSim.

Figure 1 shows how this is achieved. During the initialization of a Driver component, a Proxy instance is created for the desired interface Server. The proxy object provides channel end-points that can be connected to the ServerSim to convey invocations and replies. A reference to the proxy object is then provided to the Client instance. This completes the setup phase.

Accounting of real time starts and action of the Driver component is to run the Client instance. This will eventually call into the Proxy object which suspends accounting of real time, as it writes an Event to a channel. It then blocks waiting for the reply. When

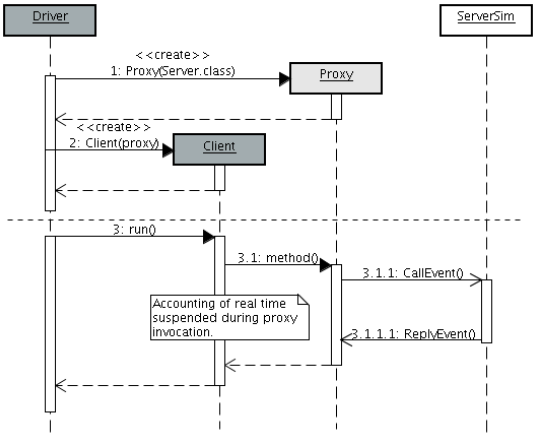

Figure 1. Real code (dark grey) calls a simulated service (white) using a proxy (light grey). Horizontal dotted line marks end of initialization.

it arrives, accounting of time is resumed and control is returned to the Client implementation.

Finally, extensions required for running real time components can be implemented in existing SSF implementations as this boils down to taking into consideration real time processes whenever entering the simulation runtime and minor changes to the scheduler. A key issue is the profiling clock used to measure real-time. It is important that the method used allows a fine-grained measurement of time by one operating system thread even if other concurrent threads are running and can thus preempt the desired thread. This was achieved in the Linux operating system using the perfctr patch.

3. Case Study

In this section we describe the application of the proposed approach to the development of a replicated database system. The replication protocol implemented [10] uses optimistic concurrency control and a group communication protocol for distributed certification. Transactions that are found to conflict or transactions that are executing in replicas that get excluded from the group are aborted to ensure 1-copy serializability. We adopt as performability criterion the probability that a transaction is committed in the distributed environment within a fixed delay of the corresponding centralized multi-processor system. We consider only transient network outages.

The system architecture used for experimenting is depicted in Figure 2. The MinhaSSF runtime underlies all components, which also make use of logging facilities. These are aware of the simulation and when appropriate suspend accounting of real time. On the top of the stack lies a OLTP workload generator based

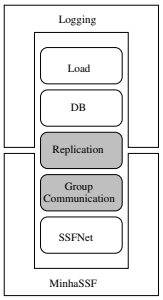

Figure 2. Real code (dark gray) and simulation code (white) interaction in the replication protocol stack.

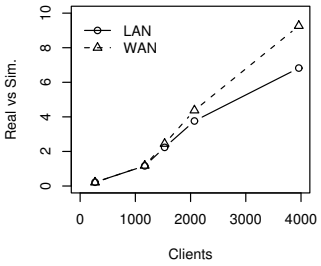

Figure 3. Ratio of time required to run the distributed database simulation vs. simulated time interval (9 servers/increasing number of clients).

on the the industry standard TPC-C [15] benchmark. The generator is comprised of a set of histograms and distribution functions, which models the database, and a SQL parser. The parser analyses the statements and translates them into result sets following to the histograms and distributions. Near the top of the stack lies the database engine model. It recreates a database behavior by modelling the execution of transactions, which consist of a sequence of *reads*, *writes* and *processing* operations, and compete for CPU, storage and locks.

Events in the database are intercepted by the replication protocol for propagation. The replication protocol used [10], relies on an atomic multicast primitive which guarantees ordered delivery of messages to ensure that all replicas accept the same sequence of transactions. This component has already been implemented in real code and is being tested. This is also the case with the group communication toolkit which provides a view synchronous and totally ordered multicast primitive. The network used to connect the entities is simulated and makes use of the SSFNet toolkit [7], which is a previously available and validated network simulation that scales to very large models. This is also the component where faults are injected by simulating either random or bursty packet loss.

There are two points of interaction between simulation and real execution runtimes. The first one lies at the network level. Therefore, whenever the group communication protocol propagates messages, they are sent into the simulated network. Whenever a message arrives from the network, it is enqueued in the list of events to be processed by the group communication layer. The second interaction happens at the database layer. Whenever the replication protocol instructs the database to apply the changes it performs a call to the simulation runtime.

The usefulness of the resulting setup for experimental evaluation of performability, early and often as implementations of replication and group communication protocols evolve, is tightly related to the ratio of time required to run the simulation in a single host to the simulated time interval. Figure 3 shows that a cluster of 9 servers and sufficient client resources to open 4000 client sessions generating up to 12000 tpm (i.e. an additional cluster) can be simulated in less than 8× the simulated time interval. When simulating 3 clusters of 3 servers interconnected by a WAN, which would be substantially harder and more costly to setup in reality, only a small additional delay is incurred.

The experimental evaluation also allowed an analysis of the system when facing packet losses. One was able to assess the processing overhead introduced by retransmission of the lost packets, as well as to discover which transactions were less tolerant to latency variability. These transactions exhibited a higher probability of ending up as aborted.

Portions of and previous versions of the proposed setup have been used to gather results on a variety of components. Namely, on consensus protocols for large clusters [11], optimistic total order protocols [13] and replication protocols [14].

4. Related Work

There is a plethora of tools and methodologies for experimental assessment of dependable distributed systems. In this section we compare them to our approach by considering the aspects relevant for agile development of middleware components with performability concerns.

The centralized simulation approach used in MinhaSSF was first used in the CESIUM framework [4], in which implementations of communication protocols are tested for real-time properties. By running multiple instances of the implementation in a single address space within a discrete-event simulation model of the environment, centralized observation and manipulation of state is allowed with reduced interference. By implementing centralized simulation within a standard simulation interface, MinhaSSF enables that existing simulation models such as SSFNet are leveraged. The automation of management of real and simulated timelines, makes it easy to incrementally replace simulated

components with their implementations as these become available.

The second approach is provided by tools to setup and control distributed tests and benchmarks such as DART [9]. Distributed testing can be performed directly in real testbeds for wide area networks such as PlanetLab [2]. Tools such as Facilita Forecast add to a distributed testing scenario the generation of representative loads, enabling load and stress testing. A large share of the complexity of tools is directly related with the distributed nature of the system under study, namely, in performing consistent global observation of system state and properties while minimizing interference. This is even more involved when one wants to observe internal variables which are relevant for performability, such as the message stability time in a group communication protocol.

Finally, fine grained simulation of computer systems can also be used to create highly realistic although small scale testbeds. Namely, SimOS [12] simulates in detail computer systems allowing the execution of COTS binary-only operating systems and application software with unparallelled observability and lack of interference. On the other hand, the detail means that substantial computing resources are required to run realistic loads and that full implementations are required for testing. This can be improved by directly running operating system and application code in the host processor. This is the approach of FAUMachine [5], which additionally allows for fault injection for dependability evaluation. Nevertheless, full implementations are still required for testing.

5. Discussion

Testing tools and techniques have recently been the subject of renewed attention in the context of agile software development methodologies that advocate testing early and often. Namely, these advocate that the design process should include an executable model and experimental evaluation. Implementation proceeds by incrementally replacing components of the model with implementations, that can be immediately tested within the model. Test harnesses built early in the development process are then to be used during the entire product development life-cycle.

Experimental tools and methodologies must therefore allow the evaluation of the aspects of the model and the implementation that are most relevant for success and incremental transition from modelling to implementation and automation of both tests and analysis. This is normally achieved by using unit testing to evaluate correctness regarding an abstract specification.

This approach is however of limited applicability when performance and dependability are key aspects of the model, as happens with middleware for large scale distributed systems such as large cluster and grid computing infrastructure. A popular approach to evaluate designs, and specifically in the study of performability of very large and complex systems, is the development of simulation models. Namely, the development of the network infrastructure, protocols, and their applications is based on tools such as ns-2 and SSFNet.

In this paper we advocate leveraging such simulation models and tools to enable testing early and often and thus an agile methodology for the development of middleware components. Such components play an increasing important role in the performability aspects of current distributed computing systems. Results obtained show that this is a cost effective approach for a realistic evaluating of very large distributed systems.

References

[1] iSSF homepage. 2003. http://www.crhc.uiuc.edu/~jasonliu/projects/issf/.

[2] Planetlab. http://www.planet-lab.org.

[3] SSF research network. http://www.ssfnet.org/homePage.html.

[4] G. Alvarez and F. Cristian. Applying simulation to the design and performance evaluation of fault-tolerant systems. In *16th Symp. on Reliable Distributed Systems (SRDS'97)*, 1997.

[5] K. Buchacker and V. Sieh. Framework for testing the fault-tolerance of systems including os and network aspects. In *In Proc. High-Assurance System Engineering Symp. (HASE'01)*, 2001.

[6] J. Cowie. *Scalable Simulation Framework API Reference Manual*, 1999.

[7] J. Cowie, H. Liu, J. Liu, D. Nicol, and A. Ogielski. Towards realistic million-node internet simulation. In *Intl. Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, 1999.

[8] B. et. al. Manif. for agile s/w development, 2001.

[9] F. L.-S. Network. Dart: Distributed automated regression testing.

[10] F. Pedone. *The Database State Machine and Group Communication Issues*. PhD thesis, 1999.

[11] J. Pereira and R. Oliveira. The mutable consensus protocol. In *Symp. Reliable Distributed Systems (SRDS)*, 2004.

[12] M. Rosenblum, E. Bugnion, S. Devine, and S. A. Herrod. Using the simos machine simulator to study complex computer systems. *Modeling and Computer Simulation*, 1997.

[13] A. Sousa, J. Pereira, F. Moura, and R.Oliveira. Optimistic total order in wide area networks. In *Symp. Reliable Distributed Systems (SRDS)*, 2002.

[14] A. Sousa, J. Pereira, L. Soares, A. C. Jr., L. Rocha, R. Oliveira, and F. Moura. Testing the dependability and performance of group communication based database replication protocols. In *Intl. Conf. Dependable Systems and Networks (DSN)*, 2005.

[15] T. P. P. C. (TPC). TPC Benchmark^{TM} C standard specification revision 5.0, Feb. 2001.