FATWa : A Testbed for Time Warp in Java

Matthew C. Lowry

Abstract

The Time Warp mechanism is a general scheme for synchronising a distributed computation of message-passing processes. It is particularly well suited to distributed parallel discrete event simulations (PDES). FATWa, an experimental Java-based PDES system using Time Warp which is currently under development, is discussed.

1 Introduction

The Time Warp mechanism is conceptually simple, and provides a elegant and general scheme for synchronising a distributed computation [Jefferson 1985]. Although it places some requirements on the system it is operating on, it has many attractive features. It is an optimistic *lookahead-rollback* scheme that can exploit a significant degree of parallelism and will not encounter deadlock as can happen with alternative *block-resume* conservative synchronisation mechanisms. Also, Time Warp operates transperently to the computation being synchronised, and does not require computation-specific knowledge for its operation. The mechanism is discussed in more detail in Section 2.

Despite its many attractions, Time Warp has been found to perform quite poorly in some circumstances when compared to conservative schemes. Many algorithms have appeared in the literature to address this issue [Fujimoto 1990]. By augmenting the basic mechanism good performance has been achieved, when even compared to conservative schemes that exploit computation-specific knowledge. There are many facets of the operation of the Time Warp mechanism that can be optimised, and for a given facet there has generally been more then one optimisation proposed in the literature. However, an apparantly neglected avenue of research is investigating the interation between the various classes of optimisation that can operate concurrently. This issue is a primary motivation behind the FATWa system which is discussed in Section 3. The system is still at an early stage of development, and directions for future work are discussed in Section 4.

2 Time Warp

The Time Warp mechanism is optimistic in that it allows processes to speculatively execute (*i.e.* process incomming messages) regardless of concern for proper ordering. So if a *straggler* message arrives out of order, a rollback mechanism is engaged. Processes are required to checkpoint their state at regular intervals, as well as retain old messages. This allows them to restore their state and reprocess messages in the correct order with the inclusion of the straggler. To revoke the messages that were sent out as a result of the incorrect computation an "*antimessage*" is sent. The analogy to antimatter here is purposeful; the antimessage is to annihilate the effect of the original incorrect message. If the original has already been processed then the receiver may also need to roll back to annihilate it, causing a cascade of further antimessages. In this way all the effects, direct and indirect, of the original message are revoked.

The issue of ordering messages is solved with the notion of "virtual time", which is essentially the same as Lamport's notion of a distributed system of logical clocks [Lamport 1978]. The idea is to replace real time as the basis for coordination in the system with the much more manageable virtual time. Each process maintains a Lamport clock, using it to give its "local virtual time". Messages are timestamped by the sender

with the local virtual time at which they wish the receiver to process the message, regardless for the real time at which it is received. The minimum value of the set of all local clock times and all in-transit message timestamps defines the extent of progress in the system amdist all the rollback activity. The prompt and accurate determination of the "global virtual time" (GVT) is an important aspect of Time Warp operation.

The rollback/antimessage scheme is completely distributed in its operation as it does not require any halt to computation beyond the process performing a rollback. It does not require knowledge of the contents of messages, and provided processes only interact by message-passing it will correctly synchronise a computation regardless of the execution rates of individual processes and the ordering of message delivery in real time. However the cascade of rollbacks and further antimessages that can be provoked by an antimessage with an early timestamp is a significant source of inefficiency in Time Warp. Another source is the need to continually checkpoint the state of processes, and the significant memory overhead for the saved checkpoints and old messages. There are algorithms that can reduce these costs, and other strategies such as load balancing are available to improve performances.

3 The FATWa Approach

The use of a PDES system as a testbed for Time Warp, as in the case of FATWa, is common. The notion of virtual time is congruent to simulation time in the PDES paradigm, and both paradigms share a common process model. In the introduction it was mentioned that the optimisation algorithms that are required for good Time Warp performance can often operate concurrently. It is inevitable that there will be emergent behaviour from their interaction. This was cited as a primary motivation behind the design of the FATWa system. A second key issue identified for investigation was dynamic load balancing. Many choices in the design of FATWa , including the use of Java as an environment to implement the system, were heavily influence by these two issues. A detailed discussion on the design and its rationale can be found in [Lowry 1999].

To support the inclusion of Time Warp optimisations with a minimum of side-effects to each other and the executing simulation a modular approach has been taken. There are three fundamental classes in FATWa which define a) the base class of simulation processes, b) objects that manage processes on a JVM, and c) an overseer object that resides on a single JVM an monitors the entire simulation. By conforming to the appropriate interface, an algorithm module can be an attached to instances of these classes. The interfaces define methods that allow modules to pass messages between each other, and to allow the host objects to inform the modules of "interesting occurrances", such as the sending of a message. These methods allow Time Warp algorithms to be implemented, since they are precisely the things upon with the operation of the algorithms are based. In [Lowry 1999] it is conjectured that by exploiting the three levels of scope given by the three classes mentioned above (local, "cluster", and global respectively) the vast majority of Time Warp algorithms can be implemented using the plug-in modules. As an example a GVT evaluation algorithm may require that the communication in and out of processes be monitored. Then a single collator must poll each process and collect their accumulated statistics to make the evaluation. This can be achieved by a module attached to the overseer object which sends polling messages to a process module, an instance of which is attached to each simulation process to observe its communication traffic. By attaching the appropriate modules multiple algorithms can operate concurrently and their interaction can be investigated with controlled experimentation.

The dynamic migration of simulation processes between JVMs partaking in a FATWa simulation was achieved in a relatively simple fashion with the Java RMI (remote method invocation) mechanism. The system employs objects that act as RMI servers between JVMs to make cross-machine calls in which a simulation process is referenced as a parameter. Thus the RMI mechanism automatically serialises the process and deserialises it at the destination machine. While migration has been supported in this fashion, the scheme has only been tested with modules that simply migrate processes and random. The issue of migration policies has yet to be investigated.

4 Future Work

Clearly implementation of some migration policies is an immediate avenue for progress. Also work on implementing more Time Warp algorithms is required; to date only a single GVT algorithm, some simple scheduling algorithms, and the random migration module have been implemented. These are the two main foci of the development of FATWa at the current time.

There are other issues that have been identified as interesting topics during development. One is the issue of making checkpoints through the copying of a process' state. In practice this means requiring the system programmer to provide cloning methods for all the process state they define. Furthermore, a vital optimisation known as "lazy cancellation" that can dramatically reduces rollbacks requires that messages can be compared to each other for equality, again requiring programmers to implement a comparator method for all messages passed between processes. A highly desirable and novel development would be to automate these two tasks in an efficient manner. Both tasks must be fast as they are very common in the operation of a Time Warp system. A number of approaches are available, for example using the reflection features of the Java environment to allow the system to identify all the state referred to by a process.

Hopefully a well developed FATWa will be capable of supporting investigation into the issues discussed above and the many others regarding Time Warp that remain to be pursued.

Bibliography

FUJIMOTO, R. M. 1990. Parallel Discrete Event Simulation. *Communications of the ACM 33*, 10 (Oct.), 30–53.

JEFFERSON, D. 1985. Virtual Time. *ACM Transactions on Programming Languages and Systems 7*, 3 (July), 405–425.

LAMPORT, L. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM 21*, 7 (July), 558–565.

LOWRY, M. C. 1999. High Performance Distributed Simulation using Time Warp and Java. Honours' thesis, Department Of Computer Science, The University of Adelaide.