CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2008; **?**:1 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

Analysing probabilistically constrained optimism

Michael Lees[1], Brian Logan[1], and Georgios Theodoropoulos[2]

[1] *School of Computer Science*
University of Nottingham UK
{mhl,bsl}@cs.nott.ac.uk
[2] *School of Computer Science*
University of Birmingham, UK

SUMMARY

In previous work we presented the DTRD algorithm, an optimistic synchronisation algorithm for parallel discrete event simulation of multi-agent systems, and showed that it outperforms Time Warp and time windows on range of test cases. DTRD uses a decision theoretic model of rollback to derive an optimal time to delay read event so as to maximise the rate of LVT progression. The algorithm assumes that the inter-arrival times (both virtual and real) of events are normally distributed. In this paper we present a more detailed evaluation of the DTRD algorithm, and specifically how the performance of the algorithm is affected when the inter-arrival times do not follow the assumed distributions. Our analysis suggests that the performance of the algorithm is relatively insensitive to events whose inter-arrival times are not normally distributed. However as the variance of event inter-arrival times increases, its performance degrades to that of Time Warp. The evaluation approach we present is generally applicable, and we sketch how a similar analysis may be performed for two other decision theoretic optimistic synchronisation algorithms.

1. Introduction

A Parallel Discrete Event Simulation (PDES) consists of a series of Logical Processes (LPs) each executing in parallel. Events are generated and processed by the LPs and an LP will typically schedule events on the other LPs in the system. *Synchronisation* ensures that events in a parallel simulation are processed in the correct causal order. One of the challenging problems of PDES is the design and evaluation of synchronisation mechanisms. Synchronisation algorithms can be classified into two categories depending on how they enforce the local causality constraint. *Conservative* algorithms strictly adhere to the local causality constraint by ensuring events are processed only when they are guaranteed not to violate causality. *Optimistic* synchronisation algorithms allow violations of causality but provide a mechanism (rollback) to undo and repeat invalid computation.

Both optimistic and conservative approaches have advantages and disadvantages. Hybrid schemes try to combine the benefits of conservative and optimistic approaches while limiting their negative effects. The performance of such throttling or bounded optimism approaches depends greatly on the assumptions underlying the metric(s) used by the algorithm to bound optimism and the degree to which the algorithm is able to accommodate simulations for which these assumptions do not hold, or hold only partially. For example, the performance of an algorithm and its ability to effectively constrain the optimism of the simulation may be extremely sensitive to the value of the chosen metric. To establish the general utility of a hybrid scheme, it is therefore important to evaluate an algorithm on models that progressively violate the assumptions underlying the metric, to determine, for example, whether its performance degrades gracefully when the assumptions do not hold.

In previous work [1, 2] we presented the Decision Theoretic Read Delay (DTRD) algorithm, a throttling mechanism for parallel discrete event simulation of multi-agent systems (MAS). DTRD uses a decision theoretic model of causality violations (rollbacks) to determine the optimal time to delay an event so as to maximise the rate of local virtual time (LVT) progression. The decision theoretic model used by DTRD assumes that the real and virtual inter-arrival times of events are normally distributed. In [2] we showed that, for event sequences for which this assumption holds, DTRD outperforms Time Warp [3] and a windowing algorithm similar to Moving Time Windows [4] on range of stereotypical multi-agent simulation test cases. In this paper we present a more detailed evaluation of the DTRD algorithm focusing on two key questions: how the algorithm performs as the variance of the input events increases (i.e., as the events become less predictable); and how the algorithm performs when the real and virtual inter-arrival times are not normally distributed.

The remainder of the paper is organised as follows. In section 2 we outline our model of MAS simulation. In section 3 we describe the DTRD algorithm and show how it uses the distribution of inter-arrival times to compute an optimal delay time for an event. In section 4 we present a detailed evaluation of the DTRD algorithm and show how its performance is affected when the variance of the input events increases and when the real inter-arrival times are not normally distributed. In section 5 we discuss some related probabilistic synchronisation algorithms and briefly outline how the evaluation technique presented here can be applied to these algorithms. Finally, in section 6, we conclude and outline plans for future work.

2. Distributed simulations of MAS

We focus primarily on the simulation of *situated agents* [5]. A MAS is situated if the agents are embedded in an environment (a metric space) and sense and act on that environment over time, e.g., robots situated in a physical environment, or characters in a computer game or interactive entertainment situated in a virtual environment. The systems of interest typically involve large numbers (thousands or tens of thousands) of agents in complex environments, e.g., individual-based ecological modelling or simulations of massively multi-player on-line games. From the point of view of simulation, one of the key aspects of situated MAS is the way in which agents interact. Situated agents are typically viewed as repeatedly executing a simple *sense—think—act* cycle. Information obtained from the environment via sensing is used, together with the agent's state, e.g., its beliefs and goals, to choose one or more actions which are then executed, updating the environment. The cycle then repeats. The environment therefore constitutes a key (and in some cases the only) medium for the agents' interactions. The environment may be represented as a simple 'passive' data structure which records the public attributes of objects and agents in the simulation, e.g., the colour, size, shape, position etc., which is updated directly by the agents, or it may be managed by one or more 'environment agents' which are responsible for computing the consequences of the action(s) of the agents and updating the environment accordingly.

The simulation of situated agents presents particular challenges for standard parallel discrete event simulation (PDES) models and techniques as described in, e.g., [6, 7]. In a conventional decentralised event-driven distributed simulation the simulation model is divided into a network of Logical Process (LPs). Each LP maintains its own portion of the simulation state and LPs interact with each other in a small number of well defined ways. The topology of the simulation is determined by the topology of the simulated system and is largely static. In many cases we know the lower bound on the timestamp of an event generated by an LP in response to an input event.

In contrast, a defining characteristic of agents is their autonomy [8]. In a parallel discrete event simulation of a multi-agent system, an agent may spontaneously generate an event at any point without there being a preceding input event. As a result, simulations of MAS typically have zero lookahead [9]. In addition, an agent's interaction with other agents and its environment is hard to predict in advance; indeed discovering how the agents interact with each other and their environment is often a primary goal of the simulation. For example, what a mobile agent can sense is a function of the actions it performed in the past which is in turn a function of what it sensed in the past. This makes it hard to determine an appropriate topology for a MAS simulation a priori, and simulations of MAS typically have a large shared state which is only loosely associated with any particular process [10].

To address these issues, we developed the PDES-MAS framework for the distributed simulation of multi-agent systems [1, 2, 10–14]. In PDES-MAS LPs are divided into two categories, *Agent Logical Processes* (ALPs) and *Communication Logical Processes* (CLPs). ALPs contain the agent models and the private state of the agents. ALPs interact with one or more CLPs which are responsible for maintaining the shared state of the simulation. In PDES-MAS, the sensing and acting phases of the agents' sense–think–act cycle are realised in terms of operations on the shared state. ALPs interact with the shared state by reading and writing *shared state variables* (SSVs): sensing gives rise to read events, and acting gives rise to write events. SSVs are similar to space-time memory [15] and other work on shared state variables in distributed simulation, e.g., [16], and have similar advantages in offering a more natural problem representation and improved performance when state variables must be accessed by distinct logical processes.

PDES-MAS adopts an optimistic synchronisation strategy as this theoretically gives the greatest speedup and avoids the problem of lookahead. With optimistic synchronisation, LPs run asynchronously and each has its own local notion of time within the simulation, referred to as its *Local Virtual Time* (LVT). As LPs execute asynchronously, events may be received out of timestamp order, i.e., with timestamps less than an LP's local virtual time. Events which arrive in the virtual past of an LP trigger a *rollback*. Rollbacks undo invalid computation and recompute the state of the LP, incorporating the new event. In the context of PDES-MAS, rollback occurs when a write event is received by a CLP which invalidates a previous read event by an agent. More precisely, a rollback occurs when a CLP receives a write with timestamp t_w from an ALP a_i to a state variable which has previously been read with timestamp t_r by some ALP a_j, such that $t_w < t_r$ and $a_i \neq a_j$. A disadvantage of optimistic synchronisation is that unconstrained optimism, i.e., where the rate of LVT progression of each LP is unbounded, can result in an excessive number of rollbacks [4], reducing the rate of LVT progression of the simulation as a whole. A major goal of PDES-MAS is therefore the development of synchronisation algorithms which limit rollback in simulations with large shared state.

3. Decision-Theoretic Throttling in DTRD

One approach to reducing rollback is *throttling*, or *bounded optimism*, in which LPs are prohibited from processing events which are likely to be rolled back. In [1, 2] we presented the DTRD algorithm, an adaptive optimistic synchronisation algorithm for PDES-MAS which uses throttling to maximise the rate of LVT progression (defined as virtual time / elapsed time) for each ALP. In conventional optimistic PDES, rollback has generally been considered in terms of late or *straggler* events. This seems intuitive given the overall goal, which is to make the simulation execute as fast as possible. However we can also say that rollbacks result from the premature processing of an event. We say an event e with time stamp t_e is *premature* if another event e' with timestamp $t_{e'} < t_e$ arrives after e in real time. In PDES-MAS only premature read events can give rise to rollbacks. Rolling back only premature reads (as opposed to all events as in [15]) is similar to the query event tagging proposed in [17] and the use of shared variables in [16], and has similar advantages in reducing the frequency and depth of rollback and the state saving overhead. However with unconstrained optimism, the number of rollbacks can still be significant.

DTRD attempts to maximise the rate of LVT progression by delaying the processing of read events which are likely to be premature until after the arrival of a straggler write event. The algorithm calculates the optimal time to delay a read of a state variable based on the history of writes to the variable. DTRD uses a decision theoretic approach to determine the ideal trade off between allowing optimistic execution and preventing rollback. The delay time for a given read event is chosen from a finite set of delay times $\{0, c\frac{1}{n}, c\frac{2}{n}, \dots, c\frac{n-1}{n}\}$, where c is the "rollback cost", i.e., the amount of elapsed time spent rolling back if the read subsequently turns out to have been premature.[*] A delay time of 0 means "commit this event immediately". Each non-zero delay time has an associated "delay cost", namely the amount of real time the read is delayed and hence the ALP which generated the read must spend blocked waiting for the value to be returned by the read. If we prefer delay times with lower cost, we would therefore always choose to "delay for 0" which has zero cost. However, for each delay time we may also have to pay the rollback cost. We assume that the probability of paying the rollback cost is lower for some delay times than others, i.e., the longer we delay, the lower the probability of a straggler write and hence of paying the rollback cost.

This gives us a simple trade-off: delaying for less time costs less (in real time) but has a higher likelihood of incurring a rollback cost. If we know the probability of a rollback occurring for each delay time we can formulate this in decision theoretic terms, and can compute the action (delay time) which maximises expected utility (i.e., minimises expected cost in this case). For simplicity, we assume each action can result in only one of two outcomes: one in which no straggler write arrives after the (real) time $now + d_i$, where d_i is the delay time, and one in which a straggler event does arrive after $now + d_i$. The expected cost of the delay action d_i is therefore:

$$\begin{aligned} &EC(d_i|E) = \\ &\quad P(NoStraggler|E) \times C(NoStraggler) + \\ &\quad P(Straggler|E) \times C(Straggler) \end{aligned}$$

[*] Note that it never makes sense to delay for longer than the rollback cost, as this would cost more than the rollback the delay prevents.

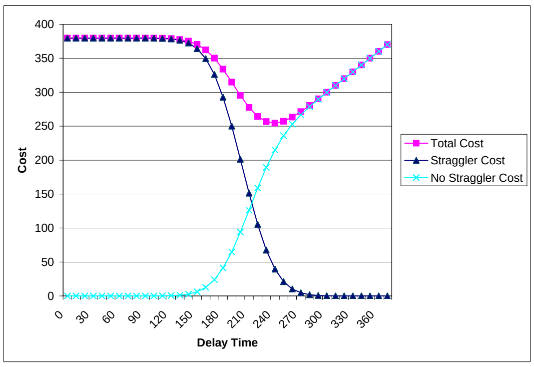

Figure 1. Minimum expected cost for a single read

where E is the history of writes to the state variable. $P(Straggler|E)$ is the probability that a straggler write will arrive after a delay of d_i, given E. $P(NoStraggler|E)$ is the probability that no straggler write will arrive after a delay of d_i given E, and can be calculated as $1 - P(Straggler|E)$. $C(NoStraggler)$ is simply the delay cost for action A_j, i.e., j. $C(Straggler)$ is the sum of the delay cost for action d_j plus the rollback cost c, i.e., $d_i + c$. The optimum action is the one with the lowest expected cost.

As an example, figure 1 shows the expected cost for a given read and history of writes to the variable for varying delay times. As can be seen, as the delay time increases, the expected cost of a straggler write decreases while the expected cost of no straggler increases. The total expected cost is the sum of these two curves, and the optimum delay is that which minimises the total expected cost (in this example approximately 250 msecs). Note that this optimum action is valid only for a specific point in real time. As real time (and the LVT of the ALPs) advances, the probability of encountering a straggler write for a given read event declines. So if we repeat the calculation after, say, one second of real time, the optimum action will be different, typically to delay for less time or to delay for 0.

The number of delay times to consider is determined by the parameter n, which is chosen to given a reasonable number of sample points. The larger the value of n, the closer the delay time chosen will be to the true optimum, however increasing n also increases the computation required to choose a delay time. Choosing the optimum action involves computing the expected cost for each delay time, and can be broken down into two parts: firstly determining the cost of rollback, and secondly determining the probability of rollback. The cost of rollback is calculated by estimating the real time cost of undoing prematurely committed events and replaying the invalid computation. To calculate the probability of rollback the algorithm assumes that the underlying distributions for the both the real and virtual inter-arrival times of write events to a variable v are normally distributed. Using the timestamps of previous write events for v it is possible to determine the mean and standard deviation of these distributions, and so calculate the probability of rollback for a read r, with real and virtual arrival time a_r and t_r. To do this we calculate the probability that the real and virtual timestamps of the next write, a_w and t_w, are such as to cause a rollback, i.e., $a_r + j < a_w$ and $t_r > t_w$ (see [2] for details).

4. Evaluating the DTRD algorithm

In previous work [2] we compared the performance of the DTRD algorithm with that of Time Warp [3] and a windowing algorithm similar to Moving Time Windows [4] on stereotypical cases designed to be representative of the kinds of access patterns found in multi-agent simulations, and on traces from a real agent simulation. The stereotypical cases characterised the agents' interaction with the shared state of the simulation in terms their degree of coupling and their skew. *Coupling* refers to how the ALPs interact with the shared state variables and the resulting potential for causality violations (rollback). *Skew* is the difference in the 'natural' rate of LVT progression between the ALPs in the absence of rollback or any throttling mechanism. A given set of ALPs may vary between high and low degrees of coupling throughout the execution of the simulation. Similarly, depending on the execution environment (system loads etc) the degree of skew may vary throughout the simulation.

In [2] we presented results for the DTRD algorithm on a range of synthetic event traces derived from both fully-coupled and uncoupled systems with varying degrees of skew. The algorithm performed well on these test cases, adapting to differing degrees of both coupling and skew. For example, DTRD has similar performance to Time Warp and outperforms time windows in the uncoupled case and outperforms both algorithms in the fully-coupled case (87% fewer replayed cycles and 30% less CPU time). To evaluate the algorithm's performance in intermediate coupling cases, we also compared the performance of DTRD with that of Time Warp and time windows on traces taken from the Boids [18] agent simulation benchmark. Boids was originally developed as a model of coordinated animal motion such as flocking birds or schools of fish. Each boid has its own local viewpoint of the flock and computes its own motion based on information it collects about other boids within its sensor range. While the Boids testbed is very simple, it captures key characteristics of situated multi-agent systems. We showed that the performance of DTRD was at least as good as Time Warp and time windows in terms of computation time in all the cases investigated, and often significantly better.

While these results are encouraging, the performance of the DTRD algorithm depends on the algorithm's ability to exploit predictability in the arrival time of the next write event. In our previous work, we focused on the structural characteristics of the agent simulation (i.e., coupling and skew), and considered only a limited range of event distributions. In the experiments reported [2], both the real and virtual inter-arrival times were assumed to be normally distributed. The virtual inter-arrival times were assumed to have a mean of 15 and standard deviation of 4, and the mean real inter-arrival time was 50 msecs for the slower agent and $50 \times \mathrm{skew}$ msecs for the faster agent, with a standard deviation for both agents of 0.05 times the mean.[†]

In this paper, we present a more detailed analysis of the performance of DTRD algorithm. We focus on two key questions:

- how the algorithm performs as the variance of the input events increases (i.e., as the events become less predictable); and

[†]The mean real cycle times were chosen as typical of many real agent systems (e.g., 5–20 fps video) and are similar to those used in other agent simulations; for example, the results reported in [19] are for agents with cycle times in the range 95–105 milliseconds.

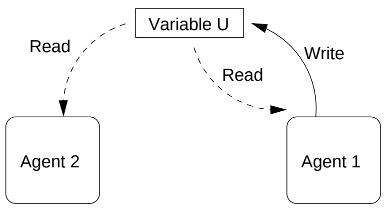

Figure 2. A half-coupled access pattern

- how the algorithm performs when the real and virtual inter-arrival times are not normally distributed.

4.1. Method

To analyse the performance of the algorithm we developed a meta-simulator. The meta-simulator abstracts the ALPs to a list of read and write events. The real and virtual times between events generated by an each ALP are defined by a probability distribution. The meta-simulator proceeds by calculating the next event to be processed. This is done by determining the ALP whose next event has the lowest real arrival time. The virtual time of this event is then calculated, and, if the event causes a rollback, the rolled-back ALP's event trace is reset to the time of the rollback. The total time for the simulation is then the total time required to process all events and any rollbacks.

We also implemented the DTRD algorithm within the meta-simulator. When DTRD is enabled, the optimal delay time for each read event is computed and the read event requeued for processing at *now* + the chosen delay time. The implementation of the DTRD algorithm uses the same method for calculating the cost and probabilities as used in [2]. For comparison purposes, we also implemented a simple version of Time Warp within the meta-simulator, which simply processes each read event immediately, i.e., no delays are applied. The meta-simulator progresses until all events have been executed, measuring the number of replayed events, rollbacks, elapsed time and total delay time applied.

For the experiments, we used a half-coupled test case [2] in which two agents, each running on a separate ALP, access a single shared state variable (see Figure 2). Each agent executes a *sense–think–act* cycle, with read and write events corresponding to the sensing and acting phases of the agent's cycle. At each cycle, agent 2 reads the variable, and agent 1 both reads and writes the variable. In all cases, agent 1 has a lower rate of LVT progression than agent 2. Agent 1 therefore constrains the optimism of agent 2 and determines the rate of global virtual time (GVT) progression of the simulation as a whole. Although the scenario is simplified, this pattern of variable accesses is representative of the kinds of interactions found in highly coupled agent simulations, where agents interact via the shared state. For example, in flocking or predator and prey simulations in which one agent (agent 2) can sense another (agent 1), but the second agent cannot sense the first [18].

We determined the performance of the algorithms both for normally distributed event sequences with different variances and for event sequences which are not normally distributed. We used the total

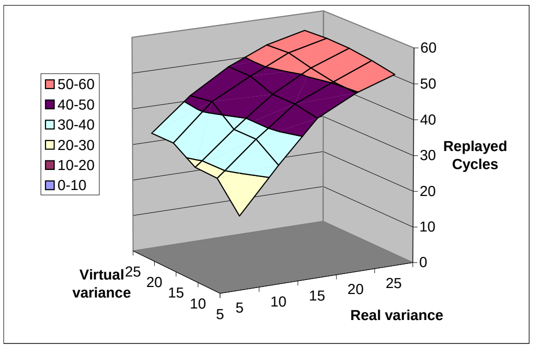

Figure 3. Performance of DTRD with increasing variance

number of cycles repeated by both agents as a result of rollback as our primary performance measure.[‡] This value indicates the frequency and depth of rollbacks within the system. For each algorithm, we report the number of replayed cycles averaged over five input traces from the Boids simulation for each set of experimental parameters.

4.2. Increased Variance of Inter-arrival Times

For the first set of test cases the input events were normally distributed with a fixed mean and increasing standard deviation. The mean real inter-arrival time was taken to be 100 msecs for the faster agent and 200 msecs for the slower agent, and the standard deviation was taken to be 5, 10, 15, 20 and 25% of the mean for each agent. The mean virtual inter-arrival time was 15 for both agents, and the standard deviation was varied from 5 to 25% of the mean. This scenario is intended to be representative of simulations in which the agents have approximately the same virtual cycle time, but different real cycle times (either because of differences in CPU availability or because the amount of CPU required by the 'think' part of the agent cycle is different for each agent). We focus on variation in real cycle times as large differences in virtual cycle times are less common in simulations of situated agents.

This set of test cases is intended to evaluate the performance of the DTRD algorithm when the underlying assumption of normality does hold but increased variance makes it harder to predict the real and virtual timestamps of the next write event. Figure 3 shows the number of replayed cycles for the DTRD algorithm as the variance of the real and virtual timestamps increases. As can be seen the number of replayed cycles increases from 20.6 when the real and virtual variance are 5% of the mean to 55.2 when the real and virtual variance are 25% of the mean. Increases in the variance of real inter-arrival times has a more marked effect on the performance.

[‡]Elapsed time is not informative in this case, as the last event generated by the slower agent determines the minimum execution time.

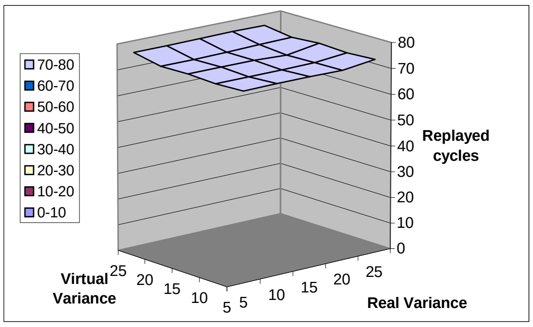

Figure 4. Performance of Time Warp with increasing variance

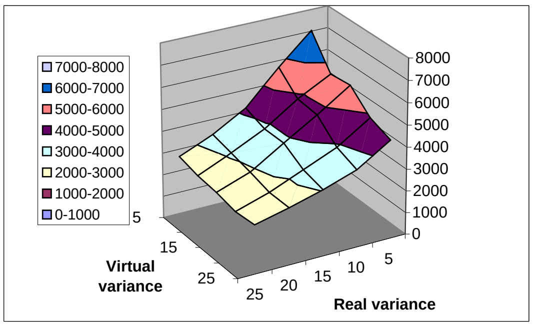

Figure 5. Total delay time for increasing variance

For comparison, Figure 4, shows the number of replayed events for Time Warp as the variance of real and virtual timestamps increases. As can be seen, the unconstrained optimism of Time Warp is relatively insensitive to increases in the variance of the real and virtual inter-arrival times. For example, the number of replayed cycles increases from 73.2 when the real and virtual variance are 5% of the mean to 75.6 when the real and virtual variance are 25% of the mean.

These results indicate that DTRD is able to exploit predictability in the event sequences to reduce the number of rollbacks and replayed cycles. This is consistent with results previously reported in [1, 2]. However, as the variance increases, i.e., as it become more difficult to predict the real and virtual timestamps of next write event, the performance of DTRD degrades until it approximates that of Time Warp.

Figure 5 shows how the average total delay time for all events decreases as the variance increases. When the real and virtual variance is 5% the total delay time is 7041.4 msecs, decreasing to 2236.7 msecs as the real and virtual variance increases to 25%. As the variance increases, the reduction

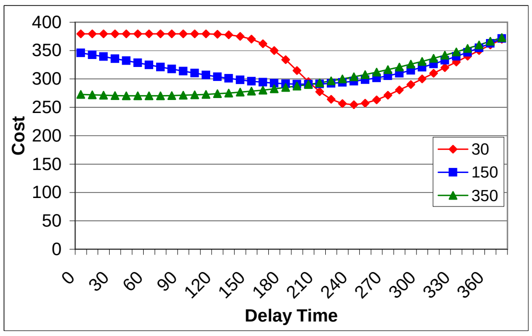

Figure 6. Minimum expected cost for a single read for increasing variance

in expected cost associated with non-zero delays reduces, and eventually disappears entirely. As an example, Figure 6 shows the expected cost computed by the DTRD algorithm for a single read by agent 2 (i.e., for a mean real inter-arrival time of write events of 200 msecs) where the standard deviation of the real inter-arrival time of writes is 30, 150 and 350 msecs. In this example the mean real inter-arrival time of writes was calculated as 202.27 msecs and the mean virtual inter-arrival time is 14.95 with standard deviation 1. The timestamp of the last write is 175 and the timestamp of the read is 205. As can be seen, when the variance is low, delaying until after the expected time of the next write event gives the minimum expected cost. However, as the variance increases, the reduction in expected cost associated with non zero delays becomes smaller until with a standard deviation of 350, there is no expected benefit to be obtained from delaying.

4.3. Non-normally Distributed Inter-arrival Times

A key assumption of the algorithm is that both real and virtual inter-arrival times are normally distributed. In this section we report experiments designed to evaluate the performance of the algorithm when this assumption does not hold.

We generated event sequences in which the real and virtual inter-arrival times were drawn from a triangular and a uniform distribution. The distributions were chosen so as to have the same mean and standard deviation as the normal distributions used for the previous experiments. For the uniform distribution the size of the outcome space was taken to be $\mu \pm (\sqrt{3}\mu \times \sigma_n)$ and for the triangular distribution the end points were $\mu \pm (\sqrt{6}\mu \times \sigma_n)$, where σ_n is the standard deviation of the corresponding normal distribution. For any given read event, the algorithm should therefore choose the same delay time as for normally distributed data, allowing the performance of the algorithm to be directly compared with the normally distributed case.

Figure 7 shows the average, minimum and maximum number of replayed cycles for the normal, triangular and uniform distributions for both DTRD and Time Warp. As might be expected, varying the distribution of the real and virtual times of events has relatively little impact on the performance of Time Warp: for all distributions, the average number of replayed cycles is around 75, though with

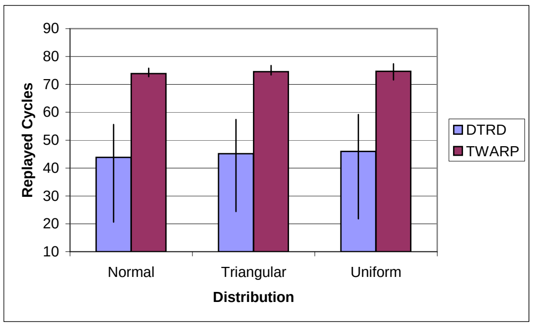

Figure 7. Performance of Time Warp and DTRD for non-normally distributed events

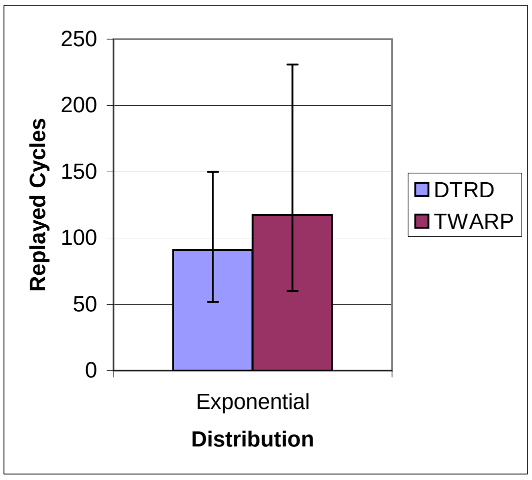

Figure 8. Performance of Time Warp and DTRD for exponentially distributed events

uniformly distributed data, the minimum values are slightly lower and the maximum values slightly higher. More surprisingly, the data also suggest that varying the distribution has relatively little impact on the performance of DTRD. The average number of replayed events increases from 43.9 in the normal case to 46 when the events are uniformly distributed. Similarly, the minimum and maximum values for the normally distributed case are 20.6 and 55.6 and for the uniform case are 21.8 and 59.2. The variation in performance is also similar to that for the normally distributed case, i.e., the number of replayed cycles increases with the variance of the input data.

We also investigated the performance of the DTRD algorithm when the real and virtual inter-arrival times are exponentially distributed. Figure 8 shows the average, minimum and maximum number of replayed cycles for DTRD and Time Warp for events chosen from an exponential distribution with the same mean real and virtual inter-arrival times as in previous cases. As can be seen, the performance of both algorithms is significantly worse than for the normal, triangular or uniform distributions. For example, with Time Warp, the average number of replayed events increases from 73.9 for normally distributed data, to 117.4 in the exponential case. With DTRD the average number of replayed events increases from 43.9 in the normal case to 90.8 in the exponential case. However, even with exponentially distributed data, DTRD still outperforms Time Warp. We believe the decrease in performance of DTRD can be attributed to the increased variance of the exponentially distributed data, rather than to the characteristics of the exponential distribution per se.

In summary, the performance of the algorithm does not appear to degrade significantly when the data are not normally distributed. Indeed the results presented above tend to support the hypothesis that the key factor is the variance of the data rather than the assumption of normality. For all distributions, as the variance increases, DTRD delays fewer events, and, in the limit, its performance degrades gracefully to that of Time Warp. Importantly, in none of the cases investigated did DTRD perform worse on average than Time Warp. This is not to suggest that the algorithm is optimal for all cases; however it does suggest that for a range of input distributions it can successfully exploit predictability in the data to reduce the number of replayed cycles.

5. Related Work

There are a number of probabilistic approaches to constraining optimistic execution in PDES the literature. In this section we look at two other optimistic algorithms, [20] and [21], which apply similar approaches to the DTRD algorithm. We indicate the assumptions made by the algorithms when computing the probability of rollback, and briefly outline how a similar analysis could be applied.

In [20] a decision theoretic algorithm similar to DTRD is used to calculate the optimal amount of CPU time an LP should delay before processing an event. The algorithm calculates the probability of rollback for a particular event and uses previously observed rollback costs to calculate the delay time. To calculate the probability of rollback, the real-virtual time plane is decomposed into spatial regions $R_{ij} = [s_{i-1}, s_i) \times [t_{j-1}, t_j)$, and the number of events and rollbacks within each region are counted. The frequency of events and rollbacks is not uniform over all real-virtual time regions. Assuming distributions of G_{vt} and G_{rt} along the virtual and real axes respectively, the interval boundaries are calculated as

$$s_i = G_{rt}^{-1}\left(\frac{i}{\mu}\right), t_j = G^{-1}vt\left(\frac{i}{v}\right).$$

When considering whether to delay an event, the algorithm determines in which region the event's inter-arrival times lie. The probability of rollback for that event is then the number of rollbacks in the region divided by the total number of events, i.e., the ratio of rollbacks per event.

While the approach presented in [20] does not model the distributions of event times using an explicit distribution, the effectiveness of the algorithm still depends on the distributions used to generate the real and virtual time regions. One may infer that the performance of the algorithm would degrade if the inter-arrival times of events followed a different distribution to the one used to decompose the real and virtual time plane. In [20] the authors indicate that with a uniform decomposition of the real-virtual time plane the algorithms performance is worse. It is not clear how the variance of the real and virtual inter-arrival time distributions would affect this approach. Without the assumption of an underlying distribution, the probability of rollback will not necessarily decrease with increased variance. With high variance, the algorithm may therefore still apply delays. However, with increased variance, we would expect that the delays would be less effective at preventing rollback.

The Minimum Average Cost (MAC) algorithm [21] is another probabilistic synchronisation algorithm which uses a decision theoretic approach to determine delay times. The algorithm is implemented within the ParaSol system, where each LP has an independent input channel for messages from each LP in the system. When the algorithm finds an empty input channel on an LP, k, it determines the amount of real time, w, for which processing on k should be suspended. w is calculated so as to give the optimal trade off between preventing cost of rollback and the cost of delaying.

The MAC algorithm records the real arrival times and timestamps of messages for each channel of an LP. The inter-arrival times are calculated and assumed to be stationary sequences. Given that an input channel is empty at time t, the algorithm calculates the probability that a straggler event will arrive after time $t + w$ as:

$$\begin{aligned}
&P\{S_j(t+w) = 1\} = \\
&\qquad P\{T_{j,M} < y_{i,N} - y_{j,M-1} \times \\
&\qquad R_{j,M} > t + w - x_{j,M-1}\}/ \\
&\qquad P\{R_{j,M} > t - x_{j,M-1}\}
\end{aligned}$$

where $T_{j,M}$ and $R_{j,M}$ are the stationary sequences of virtual and real inter-arrival times, $x_{j,M-1}$ and $y_{j,M-1}$ are the last real and virtual times respectively, and $y_{i,N}$ is the timestamp of the last event to be processed by the LP. The above example only deals with the two source case (i.e., two LPs)—in general the probability of a straggler is calculated as the probability of receiving a straggler on any input channel.

The MAC algorithm makes stronger assumptions about the distribution of inter-arrival times in that it assumes stationarity, i.e., zero variance in the inter-arrival times. With this assumption, for a given mean the probability of receiving a rollback is constant. Therefore if the inter-arrival times were normally distributed, or exponentially distributed with the same mean, the calculated probabilities would be the same. Increasing variance will therefore have no effect on the probability of rollback computed by the algorithm. If there is little variance in the inter-arrival times, the algorithm should see arrival times close to those it predicts. However, with larger variance, the assumption of stationarity would result in many events having arrival times either long before or long after the expected arrival time. Therefore, as the MAC algorithm applies the same delays for event sequences with greater variance, we might expect a degradation in the performance of the algorithm in these cases.

6. Discussion and Further Work

In this paper we have presented a detailed analysis of the DTRD algorithm, and showed how its performance changes when the inter-arrival times of events follow different underlying distributions. We first investigated the effect of increasing the variance of the distribution of inter-arrival times on the performance of the algorithm. As the variance increases, it becomes harder to predict the real and virtual arrival times of a straggler write, and the algorithm executes more optimistically. In the extreme case (standard deviation 25% of mean) the algorithm's performance is similar to that of unconstrained Time Warp. We also investigated the performance of the algorithm when the real and virtual timestamps of the input events are not normally distributed. We showed that if the inter-arrival times follow a different, but comparable distribution (in the sense that the mean and standard deviation can take the same values as a normal distribution), the performance of the algorithm is more or less equivalent to the normal case. For exponentially distributed inter-arrival times, the DTRD algorithm still outperformed unconstrained Time Warp, but both algorithms exhibited a marked reduction in performance.

The behaviour of the algorithm can be explained by considering the probability distribution function of the normal distribution. As the standard deviation increases, the probability of a variate assuming a value in a given range decreases, resulting in the algorithm estimating a lower probability of rollback. The algorithm therefore chooses smaller delay times, as the expected cost of a straggler event is lower while the cost of the next event not being a straggler remains the same.

In our analysis we have assumed a zero overhead cost. In reality the DTRD will incur a real time overhead for calculating the inter-arrival time distributions and for determining an optimal delay time. In previous work [1, 2] we have shown that, at least for some agent simulations, the performance gain outweighs the overhead associated with the algorithm. However, in general, this may not always be the case, particularly for simulations with a high degree of variance in the inter-arrival times. In such cases it may be possible to reduce the overhead of the DTRD algorithm by assuming a delay time of zero, for example, in situations where the standard deviation of inter-arrival times of writes to a variable exceeds a threshold and the cost of rollback is not too high.[§] The algorithm would still need to calculate the mean and standard deviation of the real and virtual inter-arrival times for each shared state variable, however such an optimisation would remove the overhead of considering non-zero delay times in those cases where a zero delay time would ultimately be selected while still being able to exploit predictability in the case of other variables.

In future work we plan to extended our analysis to the algorithms described in section 5, to allow us to compare their performance to that of DTRD, and to investigate how these algorithms are affected when their underlying assumptions do not hold. Another area of interest is investigating whether the performance of DTRD could be improved by adapting the assumed distribution to the observed data as proposed in [22]. We also plan to investigate the performance of the DTRD algorithm for multi-modal distributions. At present a multi-modal distribution will be observed as a unimodal distribution with high variance, however in some circumstances it may be possible to model the different modes of the distribution independently.

Acknowledgements

This work is part of the PDES-MAS project[¶] and was partially supported by EPSRC research grants Nos. GR/R45338/01, EP/C549406/1 and BBSRC grant No. BB/D006619/1.

[§]If the cost of rollback is very high, then even with a low probability of rollback the expected cost of a straggler would still be significant.

[¶]`http://www.cs.bham.ac.uk/research/pdesmas`

REFERENCES

1. Lees M, Logan B, Dan C, Oguara T, Theodoropoulos G. Decision-theoretic throttling for optimistic simulations of multi-agent systems. *Proceedings of the Ninth IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2005)*, Boukerche A, Turner SJ, Roberts D, Theodoropoulos G (eds.), IEEE Press: Montreal, Quebec, Canada, 2005; 171–178.
2. Lees M, Logan B, , Dan C, Oguara T, Theodoropoulos G. Analysing the performance of optimistic synchronisation algorithms in simulations of multi-agent systems. *Proceedings of the Twentieth ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2006)*, Turner SJ, Lüthi J (eds.), IEEE-TCSIM, ACM SIGSIM, SCS, IEEE Computer Society: Singapore, 2006; 37–44.
3. Jefferson DR. Virtual time. *ACM Transactions on Programming Languages and Systems*, vol. 7, 1985; 404–425.
4. Sokol LM, Briscoe DP, Wieland AP. MTW: A strategy for scheduling discrete simulation events for concurrent simulation. *Proceedings of the SCS Multiconference on Distributed Simulation*, SCS Simulation Series, Society for Computer Simulation, 1988; 34–42.
5. Ferber J. *Multi-Agent Systems*. Addison Wesley Longman, 1999.
6. Fujimoto R. Parallel discrete event simulation. *Communications of the ACM* October 1990; **33**(10):31–53.
7. Ferscha A. Parallel and distributed simulation of discrete event systems. *Parallel and Distributed Computing Handbook*, Zomaya AY (ed.). McGraw-Hill, 1996; 1003–1041.
8. Wooldridge M, Jennings NR. Intelligent agents: Theory and practice. *Knowledge Engineering Review* Jun 1995; **10**(2):115–152.
9. Uhrmacher AM, Gugler K. Distributed, parallel simulation of multiple, deliberative agents. *Proceedings of the Fourteenth Workshop on Parallel and Distributed Simulation (PADS'2000)*, IEEE, IEEE Computer Society: Washington DC, USA, 2000; 101–110.
10. Logan B, Theodoropoulos G. The distributed simulation of multi-agent systems. *Proceedings of the IEEE* February 2001; **89**(2):174–186.
11. Lees M, Logan B, Minson R, Oguara T, Theodoropoulos G. Distributed simulation of MAS. *Multi-Agent and Multi-Agent-Based Simulation: Joint Workshop MABS 2004*, Davidsson P, Logan B, Takadama K (eds.), no. 3415 in LNAI, Springer, 2004; 25–36.
12. Oguara T, Chen D, Theodoropoulos G, Logan B, Lees M. An adaptive load management mechanism for distributed simulation of multi-agent systems. *Proceedings of the Ninth IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2005)*, Boukerche A, Turner SJ, Roberts D, Theodoropoulos G (eds.), IEEE Press: Montreal, Quebec, Canada, 2005; 179–186.
13. Ewald R, Dan C, Theodoropoulos G, Lees M, Logan B, , Oguara T, Uhrmacher AM. Performance analysis of shared data access algorithms for distributed simulation of mas. *Proceedings of the Twentieth ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2006)*, Turner SJ, Lüthi J (eds.), IEEE-TCSIM, ACM SIGSIM, SCS, IEEE Computer Society: Singapore, 2006; 29–36.
14. Lees M, Logan B, , Dan C, Oguara T, Theodoropoulos G. Analysing probabilistically constrained optimism. *Proceedings of the Tenth IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2006)*, Alba E, Turner SJ, Roberts D, Taylor SJE (eds.), IEEE Press: Malaga, Spain, 2006; 201–208.
15. Ghosh K, Fujimoto RM. Parallel discrete event simulation using space-time memory. *Proceedings of the International Conference on Parallel Processing*, vol. III, Algorithms & Applications, CRC Press: Boca Raton, FL, 1991; 201–208.
16. Mehl H, Hammes S. Shared variables in distributed simulation. *Proceedings of the Seventh Workshop on Parallel and Distributed Simulation (PADS'93)*, ACM Press: New York, NY, USA, 1993; 68–75.
17. Sokol LM, Weissman JB, Mutchler PA. MTW: An empirical performance study. *Proceedings of the 1991 Winter Simulation Conference*, 1991; 557–563.
18. Reynolds CW. Flocks, herds and schools: A distributed behavioral model. *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM Press, 1987; 25–34.
19. Riley P. MPADES: Middleware for parallel agent discrete event simulation. *RoboCup-2002: The Fifth RoboCup Competitions and Conferences*, Kaminka GA, Lima PU, Rojas R (eds.). No. 2752 in Lecture Notes in Artificial Intelligence, Springer Verlag: Berlin, 2003; 162–178.
20. Ferscha A, Luthi J. Estimating rollback overhead for optimism control in time warp. *Proceedings of 28th Annual Simulation Symposium*, 1995.
21. Mascarenhas E, Knop F, Pasquini R, Rego V. Minimum cost adaptive synchronization: experiments with the ParaSol system. *Modeling and Computer Simulation* 1998; **8**(4):401–430.
22. Ferscha A, Chiola G. Self-adaptive logical processes: the probabilistic distributed simulation protocol. *Proceedings of 27th Annual Simulation Symposium*, IEEE Computer Society Press, 1994.