Decision-Theoretic Throttling for Optimisitc Simulations of Multi-Agent Systems

Michael Lees, Brian Logan,
School of Computer Science and Information Technology
University of Nottingham, UK
{mhl,bsl}@cs.nott.ac.uk

Chen Dan, Ton Oguara and Georgios Theodoropoulos
School of Computer Science
University of Birmingham, UK
{cxd,txo,gkt}@cs.bham.ac.uk

Abstract

In this paper we present a throttling mechanism for optimistic simulations of multi-agent systems, which delays read accesses to the shared simulation state that are likely to be rolled back. We develop a decision- theoretic model of rollback and show how this can be used to derive the optimal time to delay a read event so as to minimise the expected overall execution time of the simulation. We briefly describe an implementation of this approach in ASSK*, a distributed simulation kernel developed to investigate synchronisation mechanisms for MAS simulation, and report the results of preliminary experiments to evaluate the effectiveness of our approach.*

1. Introduction

Simulation has traditionally played an important role in multi-agent system (MAS) research and development. It allows a degree of control over experimental conditions and facilitates the replication of results in a way that is difficult or impossible with a prototype or fielded system, freeing the agent designer or researcher to focus on key aspects of a system. As researchers have attempted to simulate larger and more complex MAS, distributed approaches to simulation have become more attractive [2, 16, 8]. Such approaches simplify the integration of heterogeneous agent simulators and exploit the natural parallelism of a MAS, allowing simulation components to be distributed so as to make best use of the available computational resources. However the efficient simulation of multi-agent systems presents particular challenges for parallel discrete event simulation (PDES) models and techniques [6, 7].

In previous work [9, 13] we presented PDES-MAS, a design for an optimistic PDES kernel for the simulation of multi-agent systems. In the PDES-MAS framework LPs are divided into two categories, *Agent Logical Processes* (ALPs) and *Communication Logical Processes* (CLPs). A simulation is made up of at least one ALP and at least one CLP. The ALPs contain the actual models of the agents and their environment. The CLPs are responsible for maintaining the shared state of simulation, with each CLP managing some subset of the shared state.

A defining characteristic of agents is their autonomy. The actions performed by an agent are not solely a function of events in its environment: in the absence of input events, an agent can still produce output events in response to autonomous processes within the agent. As a result, agent simulations have zero lookahead [17]. The PDES-MAS framework therefore utilises an optimistic synchronisation strategy. However unconstrained optimism can result in excessive rollback [12]. 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 PDES-MAS, agents interact with the CLPs by reading and writing state variables. In [10, 11], we showed that rollbacks result from a particular pattern of access to the shared state, in which a CLP receives a late or straggler write with timestamp t_w from an ALP LP_i to a particular state variable which has previously been read with timestamp t_r by some ALP LP_j, such that $t_w < t_r$ and $LP_i \neq LP_j$. In this paper we present a throttling mechanism for the PDES-MAS framework, which delays read accesses to the shared state which are likely to be rolled back. We develop a decision theoretic model of rollback in PDES-MAS and show how this can be used to derive a optimal time to delay a read

event which minimises the expected overall execution time of the simulation. We briefly describe an implementation of this approach in ASSK, a distributed simulation kernel based on the PDES-MAS framework, and report the results of preliminary experiments to evaluate the effectiveness of our approach.

The remainder of the paper is organised as follows. In section 2 we present the decision theoretic throttling model and show how it can be used to compute an optimal delay time for a read access. In sections 3 and 4 we expand on the assumption underlying the model and briefly outline how the probabilities and utilities required by the decision theoretic model are calculated. In section 5 we outline how the model is implemented by a CLP, and in section 6 we present results from a number of simple agent simulations which illustrate the effectiveness of our approach. In section 7 we discuss related work and in 8 we conclude and outline plans for future work.

2. Decision-Theoretic Throttling

In conventional optimistic PDES, rollback has generally been considered in terms of late or *straggler* messages. 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 processing an event prematurely. We say an event e with time stamp t_e is *premature* if another event e' with timestamp $t_{e'} < t_e$ will arrive after e in real time.

Throttling is a means of increasing the performance of optimistic PDES, where performance is taken as the elapsed time required to complete the simulation. The performance increase obtained through using throttling rests upon two assumptions. Firstly, the probability of rollback increases as events are processed further from the global virtual time. Secondly, that the overall performance of the simulation is increased by reducing rollback.[1] The second assumption restated becomes: the real time cost of blocking at an event until the window advances is less than the real time cost of processing the event plus the cost of any subsequent rollback.

We can reformulate the intuition underlying throttling in decision theoretic terms, where the total elapsed time is viewed as a cost to be minimised. A decision to process an event at a particular point in real time will result in one of a number of different outcomes, each of which has an associated probability and cost (in terms of elapsed time). In the PDES-MAS framework the relevant decisions are when to process read events received by a CLP. We assume that agents execute a *sense–think–act* cycle, in which they obtain information from the environment (shared state) and compute an action which changes the environment. Sensing gives rise to read events, and acting gives rise to write events. Write events can always be processed immediately. A write event by an ALP, ALP_i with a time stamp t_w can only be rolled back if a read by ALP_i with timestamp $t_r < t_w$ is rolled back. An ALP which only writes and never reads is not influenced by the other ALPs and can never be rolled back. We can also safely process a read event with timestamp t_r on a shared state variable v if t_r is lower than the LVT of all ALPs other than the one which generated the read, e.g., if all other ALPs have written v with a timestamp greater than t_r. For all other read events, a CLP should delay processing a read if it is likely that the read is premature, i.e., if there is reason to believe that the read is likely to be rolled back by a straggler write.

We assume that the possible delay times are chosen from a finite set of delay times $\{0, 1, 2, \ldots\}$. A CLP can therefore take one of a number of actions: *delay for 0*; *delay for 1*; *delay for 2*; .... where "delay for 0" means "process this event immediately". Each action has an associated cost, namely the amount of real time the read is delayed and hence ALP which generated the read must spend blocked waiting for the value returned by the read. If we prefer actions with lower cost, we would therefore always choose the "delay for 0" action which has zero cost. However, for each action we may have to pay an additional "rollback cost"—the real time spent rolling back if the read subsequently turns out to have been premature. We assume that the probability of paying this rollback cost is lower for some actions than others, i.e., the longer we delay, the lower the probability of a straggler write and hence of paying the rollback cost. Given two reads events, r_1 and r_2 with timestamps t_{r1} and t_{r2} such that $t_{r1} \leq t_{r2}$, a straggler write which rolls back r_1 will also roll back r_2 (if this has been committed). However we may decide to process r_1 if we consider a straggler write with a timestamp $t_w < t_{r1}$ unlikely, and delay processing r_2 if a straggler write with timestamp $< t_{r2}$ is more likely. Note that it never makes sense to delay for longer than the rollback cost. All actions therefore result in the event being processed (since we never delay for an infinite time).

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 action (delay time) we can formulate this in decision theoretic terms, and can compute the action which maximises expected utility (i.e., minimises expected costs in this case).

For simplicity, we assume each action can result in two distinct outcomes: one in which no straggler event arrives after the (real) time $now + d$, where d is the delay time, and

[1] In cases where the execution time of the simulation is bounded by a consistently slower agent which is never rolled back, the time required for the simulation to complete may not be affected by rolling back the faster agents.

one in which a straggler event does arrive after $now + d$.[2] Given the utilities of these two states (in terms of their real time cost), the expected utility, EU of each action, A_j given evidence E is:

$$EU(A_j|E) = \sum_i P(Result_i(A_j)|E) \times U(Result_i(A_j))$$

where $P(Result_i(A_j)|E)$ is probability of the i'th possible outcome of action A_j given evidence E and $U(Result_i(A_j))$ is the utility of this outcome. For each action j (delay time) there are two possible outcomes, giving:

$$EU(A_j|E) = \\ P(NoStraggler|E) \times U(NoStraggler) + \\ P(Straggler|E) \times U(Straggler)$$

$P(NoStraggler|E)$ is the probability that no straggler write will arrive after time $now+j$, where E is the evidence (history of writes to the variable v). $U(NoStraggler)$ is simply the delay time for action A_j, i.e., j. $U(Straggler)$ is the sum of the delay time for action A_j plus the rollback cost c, i.e., $j + c$. The optimum action is the one with the lowest expected cost.

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 the current 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. Whether it is worth recalculating the optimum delay times periodically (as opposed to simply computing the delay once and processing the delayed event when the delay has elapsed) depends on whether we can get better probability estimates with time. For example, if it somehow becomes clear that a "slow" LP has speeded up or stopped writing to a variable, this will change the probability of rollback for reads of the variable.

3. Estimating the Probability of a Straggler Write

Given a read event of a variable v with timestamp t_r and a real time delay, j, the next write to a variable v is a straggler if two conditions are satisfied. Firstly, the write must arrive after $now + j$. We call this the *real-time condition*. Secondly the timestamp of the write, t_w, must be less than the time stamp of the read t_r and greater than or equal to t_v, the timestamp of the write immediately preceding t_r in virtual time, i.e., $t_v \leq t_w < t_r$. We assume reads always occur "before" writes with the same timestamp, so a straggler write must have a timestamp which is strictly less than a read to trigger a rollback. This is called the *virtual-time condition*. We could reformulate the virtual time condition so that a write with a timestamp at *any* point before the timestamp of the read is viewed as a straggler, rather than only timestamps between t_v and t_r. While only writes with timestamps between t_v and t_r are guaranteed to cause a rollback, writes with timestamps less than t_v may indirectly trigger a rollback, by invalidating a prior read. However, for simplicity, we ignore this case. Figure 1 illustrates the values for real times and virtual times which will result in a rollback.

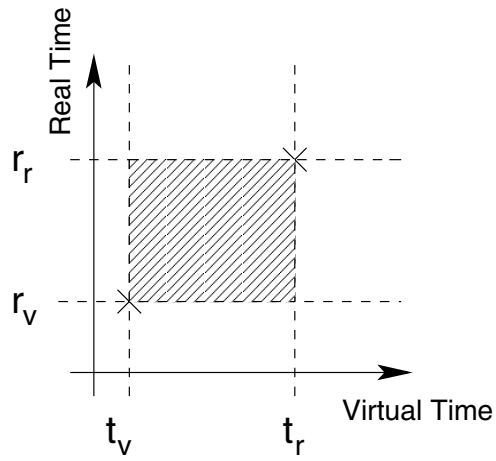

Figure 1. Real and virtual time conditions for rollback

We model the real and virtual time conditions as two probability distributions: P_a, the (real) arrival time of the next write event, and its timestamp, P_t. We are only interested in the arrival times of writes with timestamps less than that of the read we are processing. However, for simplicity, we assume that the two distributions, P_a and P_t, are independent, since the writes may originate from ALPs with differing LVTs, and although in general timestamps increase with increasing arrival times, this is not guaranteed, e.g., if one or more ALPs rolls back.

P_a and P_t can be computed from the arrival time and timestamp of the most recent write to the state variable v and the history of previous writes to v. We assume that Δ_a, the difference between the real arrival times of successive writes to v, is a Poisson process which can be modelled using an exponential distribution. If the real arrival time of the most recent write to v is a_l, then the probability that the next write will arrive after a delay of j is the probability that Δ_a is greater than $now + j - a_l$.

The probability that an exponentially distributed variate

[2] Strictly, this should be "in which at least one straggler write arrives after $now + d$". In this paper, we consider the case in which there is only one straggler.

with arrival rate $\lambda = \frac{1}{\mu_a}$ assumes a value in the interval $[0, z]$ is given by the exponential cumulative distribution function,

$$F(z) = 1 - e^{-\lambda x}$$

where μ_a is the mean real inter arrival time. Note that care must be taken when computing μ_a. If an agent both reads and writes a variable, then delaying a read will increase the real arrival time of subsequent writes and hence the mean real inter-arrival time for the variable. This can result in positive feedback, with delays leading to higher mean inter-arrival times and hence longer delays, since the probability of a straggler write increases as the mean real inter-arrival time increases. We therefore compute a mean real inter-arrival time for each ALP which takes into account only writes from other ALPs.

For P_t, we assume that Δ_t, the difference between the timestamps of successive writes to v is normally distributed. If t_l is the timestamp of the write to v with arrival time a_l, (i.e., the most recent write to v), then the probability that the next write will have a timestamp in the interval $[t_v, t_r)$ is the probability that Δ_t is in the interval $[t_v - t_l, t_r - t_l)$. Note that t_r can be greater than t_l, e.g., $t_v = t_l$, in which case the interval becomes $[0, t_r - t_l)$. For example, given a read with a timestamp of t_{20} and a sequence of writes with timestamps: $t_{10}, t_{15}, t_{17}, t_{30}$ where t_{30} is the timestamp of the most recent write, we compute the probability that the next write to arrive will have a timestamp difference in the interval $[-13, -10)$.

The probability that a standard normal variate assumes a value in the range $[z_1, z_2]$ is given by

$$\Phi(z_1, z_2) = \frac{1}{2}\left[\mathrm{erf}\left(\frac{z_2}{\sqrt{2}}\right) - \mathrm{erf}\left(\frac{z_1}{\sqrt{2}}\right)\right]$$

Neither the normal distribution function Φ or *erf* exists in simple closed form and must be computed numerically. There are however many numerical approximations to the cumulative normal function [1]. These approximations can achieve greater accuracy then numerical integration techniques such as trapezoidal or Simpson's rule.

The probability of a straggler write is therefore

$$P(Straggler|E) = 1 - e^{-\lambda x} \times \Phi_t\left(\frac{(t_v - t_l) - \mu_t}{\sigma_t}, \frac{(t_r - t_l - 1) - \mu_t}{\sigma_t}\right)$$

where Φ_t is the normal distribution function for P_t, and μ_t and σ_t are the mean and standard deviation of the difference between the timestamps of successive writes.[3]

The probability of no straggler is simply

$$P(NoStraggler|E) = 1 - P(Straggler|E)$$

We can therefore compute the probabilities of each of the outcomes for a given delay time, j, and hence the optimum delay action. If too few events have been sent by an ALP to determine the mean and standard deviation (e.g., on startup), we return a delay time of 0.

4. Estimating the Cost of Rollback

Estimating the cost of rollback is hard, due to the difficulty of determining the likelihood of ALPs other than that which generated the read event being rolled back if the read event is processed. We therefore only consider the cost of rolling back the ALP which generated the read, accepting that this may be an underestimate of the true rollback cost.

The rollback cost consists of two separate components: the cost of rolling back the ALP to the timestamp of the straggler (the undo cost) plus the cost of the ALP repeating undone computation.[4] For simplicity, we assume that the ALP's undo cost, u, is constant. In reality, u is dependent on the type of state saving used in the system, e.g., incremental or periodic.

To calculate the cost of replaying rolled back events by the ALP we assume a worst case straggler with timestamp equal to t_v, and compute the real time required for the ALP to advance from t_v to t_r. This is estimated from rate of LVT progression, δLVT, of the ALP which initiated the read, i.e., the real time required for the ALP to advance one unit of virtual time. δLVT is computed by the CLP for each ALP, and is assumed to be simply the timestamp of the last event received from the ALP (which we assume to be t_r) divided by the elapsed time since the start of the simulation. The replay cost also includes any processing done by the ALP between the end of the delay period and before the arrival of the straggler write. We assume this additional replay cost to be equal to $max(a_l + \mu_a - j, 0)$, i.e., the expected arrival time of the straggler write $a_l + \mu_a$ less the delay time, j, if this is positive.

The total rollback cost is therefore:

$$r = u + ((t_r - t_v) \times \delta LVT) + max(a_l + \mu_a - j, 0)$$

5. Implementation

We have implemented the approach outlined above within ASSK [11], a distributed simulation kernel based on the PDES-MAS framework developed to investigate synchronisation mechanisms for MAS simulation. ASSK is a library of C++ classes which use MPI for inter-process

[3] If the standard deviation of difference between successive timestamps, σ_t, is close to zero, we assume stationarity, i.e., that the timestamp of the next write will be $t_l + \mu_t$.

[4] We assume that the CLP's undo cost is marginal, i.e., the cost of undoing the current read. Similarly, we assume that the cost of the ALP replaying the events is significantly greater than the cost of processing the replayed events by the CLP, and ignore the latter in computing the rollback cost.

communication. ASSK does not interface directly with a MAS simulator, rather it takes as input event traces from a MAS simulation. An ASSK simulation consists of one or more agent ALPs and a single shared state LP (SSLP) which maintains the shared state of the simulation. Each ALP processes an event trace from an agent in the original agent simulation and asynchronously sends read and write events from the trace to the shared state LP. Upon receiving an event the SSLP applies the appropriate access to the relevant state variable and generates any necessary responses. If the SSLP receives a straggler write from an ALP, it rolls back the state variable to the timestamp of the write, and triggers rollbacks on all the ALPs which read incorrect value(s) of the variable. This causes the ALPs to delay for a time (representing the undo cost) and then rewind and replay their event traces from the time of the rollback. Although deterministic in that the events generated by the ALPs are entirely determined by the input traces, ASSK provides a flexible framework for synchronisation experiments.

With decision-theoretic throttling turned on, the SSLP checks all incoming reads and computes the expected utilities for 20 delay times, evenly distributed between 0 and three times the mean inter-arrival time, μ_a. If the delay time greater than zero has lowest cost, the SSLP places the read and the selected delay time in the *delay queue*. When the delay time has elapsed, the read is dequeued and processed as normal.

6. Results

To investigate the effectiveness of decision-theoretic throttling, we performed a number of experiments in which we compared the performance of unconstrained optimism (i.e., no throttling) and the decision theoretic algorithm outlined above. The performance measures collected included the total elapsed time, the number of rollbacks, and, in the decision theoretic case, the total delay time.

The experiments used a synthetic event trace, in which two agents access a single shared state variable. 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 1 reads the variable, and agent 2 both reads and writes the variable. 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 can sense another, but the second agent cannot sense the first [15].

We investigated the effect of varying the real and virtual time required by the agents to complete a single *sense–think–act* cycle. The real cycle time of the agents was modelled as an exponential distribution. For agent 1, the mean cycle times were taken to be 80, 160 and 240 milliseconds. In each case, agent 2 has a mean cycle time which is twice that of agent 1, i.e., the mean cycle times for agent 2 were taken to be 160, 320 and 480 milliseconds. For the virtual cycle time, we used a normal distribution with mean 15 and standard deviation 4 for both agents. To investigate the effect of differing rollback costs, we also varied the undo cost from 20 milliseconds to 300 milliseconds.

The experiments were performed on a homogeneous mini-cluster consisting of four nodes (Pentium IV 3.1 GHz, 1GB RAM) running Redhat 7.2 connected via gigabit Ethernet. The results presented below represent an average of 5 runs of 100 agent cycles.

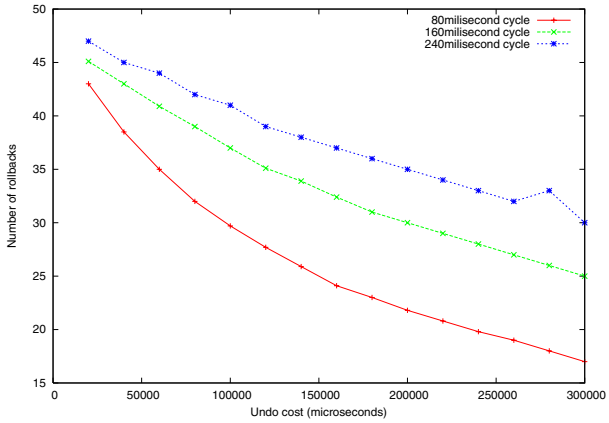

Figure 2. Number of rollbacks (unconstrained)

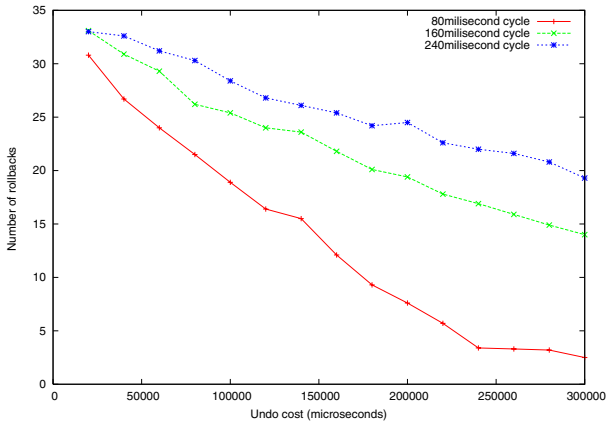

Figure 3. Number of rollbacks (decision theoretic)

Figure 2 shows the number of rollbacks for the unconstrained (purely optimistic) case for each undo cost. As can

be seen, as the undo cost of rollback increases, the number of rollbacks declines, as it takes longer (in real time) for agent 1's LVT to become greater than that of agent 2 following a rollback, so reducing the number of occasions on which agent 1 can be rolled back. Figure 3 shows the corresponding number of rollbacks for the decision theoretic case. As can be seen, for all mean cycle times, the number of rollbacks is lower than in the unconstrained case. For example, in the case in which agent 1 has a cycle time of 80 milliseconds, there are 31 rollbacks when the undo cost is 20 milliseconds (compared with 44 in the unconstrained case), dropping to 2 rollbacks with an undo cost of 300 milliseconds (compared to 17 in the unconstrained case).

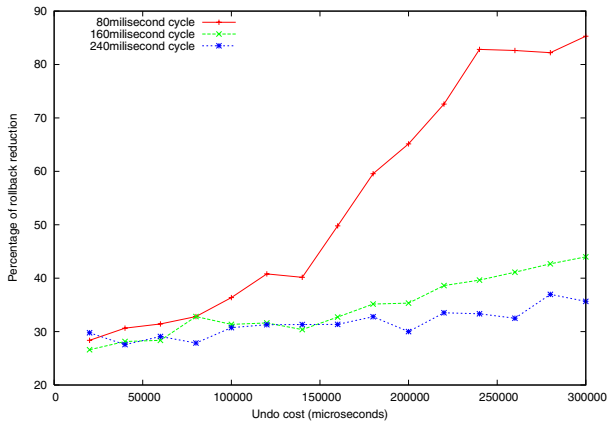

Figure 4. Reduction in rollback

The difference between the two cases can be seen more clearly in Figure 4, which shows the percentage reduction in rollbacks when using the decision theoretic algorithm compared to purely optimistic synchronisation. As can be seen, for each mean real cycle time investigated, the decision theoretic algorithm reduces the number of rollbacks by between 25% and 85%. As the undo cost increases, the algorithm prevents more rollbacks. This effect is most pronounced with a mean cycle time of 80 milliseconds and undo costs greater than 100 milliseconds, as in these cases the cost of delaying a read by agent 1 until the straggler write produced by agent 2 arrives is small relative to the cost of a rollback. With larger mean cycle times, the algorithm allows more rollback to occur as the delay required to prevent a rollback is larger in relation to the rollback cost.

Figure 5 shows the reduction in computation time (elapsed time − delay time) for the simulation when using the decision theoretic algorithm. As can be seen, the reduction in computation time correlates closely with the reduction in rollback shown in Figure 4.

This is more or less what we would expect: the algorithm converts rollbacks (and rollback time) into delay time, injecting sufficient delays to ensure that agent 1 does not advance LVT at a faster rate than agent 2. However while rollback time is essentially 'wasted' CPU, delay time is time that the ALP spends blocked in a read. This time can be used by other processes (e.g., other ALPs or processes) or to facilitate load balancing within the simulation.

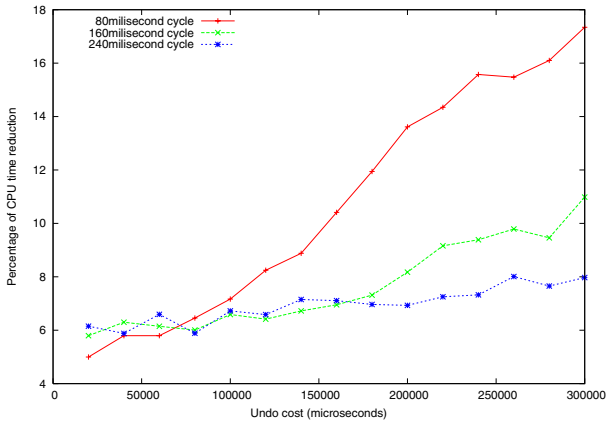

Figure 5. Reduction in computation time

7. Related Work

A number of probabilistic approaches to throttling have been described in the literature. Below we summarise some of this work and briefly indicate its relationship to the approach presented in this paper.

Ferscha and Chiola [4] have proposed a probabilistic protocol which uses timestamps to estimate how likely a particular message arrival is to cause a rollback. Before sending each message the probability of rollback for this event is determined using information about previous arrivals. The time window of the sending LP is then adapted in relation to the previously observed arrivals at the LP. Different methods were used for characterising the arrival process, including average token time increment, linear regression and both normal and exponential distributions. The protocol is similar to the Moving Time Windows protocol [12] except LPs are not required to block when they reach the end of their window. Instead, when the end of the window is reached, the probability of an event triggering a rollback is considered, and, if the event is likely to cause rollback, it is blocked. This has the effect of slowing down an LP after it leaves the window, with the rate at which it slows being determined by the confidence estimate. The Ferscha and Chiola protocol differs from the approach presented here in a number of respects. Firstly the probabilistic blocking is done on the sender side rather than the receiver side. Secondly only virtual time is considered when determining rollback probability. Finally the scheme does not use a decision theoretic model to determine an optimal de-

gree of constraint.

In [5] Ferscha and Luthi present a similar technique based on CPU delay intervals. The optimal delay interval is determined by a probabilistic cost expectation function (PCEF). This involves each LP monitoring the rate of LVT progression per unit of CPU time. Each message has two timestamps: the first corresponding to the virtual time of the event and the second corresponding to the real time at which the message was received. These timestamps are used by a probabilistic decision function which determines the optimal trade off between loss of CPU cycles due to blocking and wasted CPU time due to rollbacks. The probability of rollback is assumed to be $\frac{r}{e}$, where e is the total number of events and r is the total number of rollbacks, and the cost of a rollback is taken to be the average (real) rollback time, including the time required to replay events. This approach was tested using a simulation of stochastic petri nets and was shown to outperform standard Time Warp. The authors state the approach will perform best in simulation models with a high degree of imbalance in LVT progression. There are a number of similarities between this work and our own. It uses a decision theoretic model and both real and virtual time are used in determining rollback probability. However, unlike the work presented in this paper rollback probability is not determined by assuming underlying distributions for virtual and real time arrivals.

In the Probabilistic Adaptive Direct Optimism Control (PADOC) protocol [3], the timestamps of previous messages are used to predict the timestamp of the next message. The number of previous messages used in this prediction can be changed to suit the simulation: using more messages gives a more accurate prediction but also incurs greater overhead. A number of different forecasting methods are investigated in the paper, including three "straightforward" methods: arithmetic mean, exponential smoothing and median approximation, and a more complex method, integrated autoregressive moving average (ARIMA). The results show the ARIMA method outperforms all the straightforward cases and standard Time Warp. This work is the most similar to the work presented here. It does however differ in the method used for determining rollback probability and doesn't make a distinction between read and write events.

Another cost model similar to [5] was proposed by Mascarenhas et al in [14]. The Minimum Average Cost (MAC) algorithm is based on minimising the cost of synchronisation delay and rollback overhead, as in [5]. Using the probability of rollback, they determine if it is better to wait on an empty input channel which may later receive a straggler message. Unlike the work presented here, the MAC approach assumes that the real inter arrival time and timestamp of events are stationary sequences. Based on this assumption, the probability that the next event on a channel will arrive after a given amount of real time and turn out to be a straggler is calculated. The model was tested on three examples of closed queueing systems and was shown to reduce rollback costs by about 25% on average compared to standard Time Warp. However, in some cases the overhead of the adaptive mechanism became so great that any improvement gained was lost through overheads.

In previous work [11], we presented another approach to throttling based on *critical accesses*. The set of critical accesses between two logical processes p_i and p_j is defined as:

$$CA_{ij} = |s_R(p_i) \cap s_W(p_j)| + |s_W(p_i) \cap s_R(p_j)|$$

where $s_R(p_i)$ and $s_W(p_i)$ are the set of reads and writes made by p_i. Reads by an ALPs which share a large number of critical accesses with other ALPs and which have a large difference in LVT are assumed to be more likely to be rolled back and are delayed at the sending ALP. This has the advantage that the degree of optimism can be adjusted to reflect the degree of overlap of the ALP's spheres of influence and their rate of LVT progression. However this approach requires knowledge of the number of critical accesses shared with other ALPs and their LVTs. In the PDES-MAS framework, this information is distributed over the CLPs and ALPs respectively. In contrast, the approach presented in this paper is less discriminating than the critical access approach but requires less information.

8. Discussion and Further Work

In this paper we have presented an adaptive throttling mechanism which uses a decision theoretic model to choose an optimal delay time for read events which are likely to be premature. We outlined the assumptions on which our model is based and briefly described a prototype implementation of the decision theoretic model in the ASSK simulation kernel. The key contributions of this work are:

1. the use of decision theoretic approach to choose the optimal delay times
2. both virtual and real time components of rollback are considered;
3. delays are computed at the receiving LP and information used by the algorithm is available locally, reducing the overhead of data collection;
4. the algorithm exploits the potential for Read/Write optimisation within the PDES-MAS framework;

While previous work has addressed some of these aspects, to the best of our knowledge, the combination of features adopted here is novel.

To investigate the effectiveness of our approach, we conducted a number of experiments using synthetic event traces from simulations of simple multi-agent systems. While preliminary, the results show how the performance of our approach compares to the purely optimistic (unconstrained optimism) case and how the model's behaviour adapts with differing real time costs for rollback. For the simple cases studied, the results are encouraging, showing a five to ten-fold reduction in the number of rollbacks (and time spent rolling back) compared to the unconstrained case.

The results presented in this paper demonstrate 'proof of concept', and indicate the feasibility of applying decision theoretic throttling to simulations of multi-agent systems. We are currently testing the throttling mechanism using event traces which characterise more complex patterns of agent interaction and with event distributions which deviate from the normal and exponential distributions assumed by the algorithm, to establish the sensitivity of the algorithm to the underlying statistical properties of the simulation. In addition, we are evaluating its effectiveness in cases where there are many interacting agents, using event traces from a number of different agent simulations, including SIM_TILEWORLD and SIM_BOIDS.

In future work, we plan to investigate extending our approach to include the possibility of more than one straggler, relaxing the virtual time condition to include any straggler write with a timestamp less than t_r, and utilising alternative methods estimating rollback cost. We also hope to investigate applying the approach to other simulation problems with a large shared state, e.g., mobile networks.

References

[1] M. Abramowiz and I. A. Stegun. *Handbook of Mathematical Functions*. National Bureau of Standards, 1964.

[2] J. Anderson. A generic distributed simulation system for intelligent agent design and evaluation. In *Proceedings of AI, Simulation and Planning In High Autonomy Systems*, 2000.

[3] A. Ferscha. Probabilistic adaptive direct optimism control in time warp. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS '95)*, pages 120–129, 1995.

[4] A. Ferscha and G. Chiola. Self-adaptive logical processes: the probabilistic distributed simulation protocol. In *Proceedings of 27 th Annual Simulation Symposium*. IEEE Computer Society Press, 1994.

[5] A. Ferscha and J. Luthi. Estimating rollback overhead for optimism control in time warp. In *Proceedings of 28 th Annual Simulation Symposium*, 1995.

[6] A. Ferscha and S. K. Tripathi. Parallel and distributed simulation of discrete event systems. Technical Report CS.TR.3336, University of Maryland, 1994.

[7] R. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):31–53, October 1990.

[8] L. Gasser and K. Kakugawa. Mace3j: Fast flexible distributed simulation of large, large-grain multi-agent systems. In *Proceedings of AAMAS-2002*, Bologna, July 2002.

[9] M. Lees, B. Logan, R. Minson, T. Oguara, and G. Theodoropoulos. Distributed simulation of MAS. In *In Proceedings of the Joint Workshop on Multi-Agent and Multi-Agent-Based Simulation*, pages 21–30, 2004. (to appear).

[10] M. Lees, B. Logan, and G. Theodoropoulos. Adaptive optimistic synchronisation for multi-agent simulation. In D. Al-Dabass, editor, *Proceedings of the 17th European Simulation Multiconference (ESM 2003)*, pages 77–82, Delft, 2003. Society for Modelling and Simulation International and Arbeitsgemeinschaft Simulation, Society for Modelling and Simulation International.

[11] M. Lees, B. Logan, and G. Theodoropoulos. Time windows in multi-agent distributed simulation. In *Proceedings of the 5th EUROSIM Congress on Modelling and Simulation (EuroSim'04)*, Paris, September 2004. (to appear).

[12] A. W. L.M. Sokol, D.P. Briscoe. Mtw: A strategy for scheduling discrete simulation events for concurrent execution. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 34–42, San Diego, California, February 1988.

[13] B. Logan and G. Theodoropoulos. The distributed simulation of multi-agent systems. In *Proceedings of the IEEE*, pages 174–185, 2001.

[14] E. Mascarenhas, F. Knop, R. Pasquini, and V. Rego. Minimum cost adaptive synchronization: experiments with the parasol system. *Modeling and Computer Simulation*, 8(4):401–430, 1998.

[15] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34. ACM Press, 1987.

[16] B. Schattenberg and A. Uhrmacher. Planning agents in james. In *Proceedings of IEEE*, 2000.

[17] A. Uhrmacher and K. Gugler. Distributed, parallel simulation of multiple, deliberative agents. In *Proceedings of Parallel and Distributed Simulation Conference (PADS'2000)*, pages 101–110, May 2000.