Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

TWO PROCESSOR TIME WARP ANALYSIS: A UNIFYING APPROACH

L. KLeinrock
R. Feldterman

June 1992
CSD-920034

Two Processor Time Warp Analysis: A Unifying Approach

Leonard Kleinrock and Robert E. Felderman*

January 1, 1992

Abstract

We present a new model and its exact analysis for the problem of two processors running the Time Warp distributed simulation protocol. The model creates a unifying framework for previous work in this area and additionally provides some clear insight into the operation of systems synchronized by rollback.

Keywords: Discrete Event Simulation, Time Warp, Parallel Processing, Distributed Processing, Optimistic Simulation, Rollback, Speedup, Performance Analysis, Markov Chain.

1 Introduction: Discrete Event Simulation

The systems which we are able to create become larger and more complex every day. We have moved beyond a point where one is able to predict the performance of a large system, be it a complex computer network or a super-sonic airplane, by purely analytical means. It is now necessary to simulate the operation of proposed systems in order to better understand their behavior. Additionally, simulation is a useful tool to examine events unlikely to occur in the "real world", such as a nuclear attack. As the size of a simulation increases it demands more computing time. Naturally then, one would like to utilize the recent advances in parallel computing technology to speed up the execution of simulations. Unfortunately, it is a non-trivial task to efficiently implement a parallel simulation system, though several techniques have been developed to do so. This paper presents an analytical model of the performance of one distributed simulation algorithm, Time Warp (TW) [1]. We analyze the operation of TW when run on two processors and obtain exact results for important performance measures; most notably, the speedup of two processors over execution on a single processor. To better understand Time Warp and our underlying model of it, we first discuss parallel simulation in general.

Parallel Discrete Event Simulation (PDES) is generally accomplished by partitioning the simulation into logical processes (LP) which simulate some physical process in the system. Each LP maintains an independent local clock indicating how far forward in simulation time it has progressed. Processes interact by sending and receiving timestamped messages. Each process operates autonomously by receiving messages, performing internal computation and sending messages. A process will terminate once its local clock, the time of receipt of the message currently being processed, has reached some user specified value. Certain LPs perform operations only in response to messages (the messages carry the work), while others perform internal computations regardless of whether any messages have arrived. For example, an LP which is simulating a single server queue would only perform an operation in response to the arrival of a message (customer). On

*This work was supported by the Defense Advanced Research Projects Agency under Contract MDA 903-87-C0663, Parallel Systems Laboratory. To appear in the International Journal of Computer Simulation, late 1992.

the other hand, an LP which simulates a customer arrival process would be able to operate without receiving any messages at all, it just keeps generating customers. Nicol [2] discusses these two types of logical processes in more detail.

Each LP could be placed on its own processor, and one might hope that we could then gain speedup proportional to the number of processors used. Unfortunately, this is often not the case as the system being simulated may have only limited parallelism [3]. Also, the PDES algorithms themselves limit parallelism in their attempts to prevent the simulation from deadlocking and to ensure correctness. Several competing techniques have been developed to address deadlocking and correctness [4]-[5]. The algorithm of interest for this paper is Time Warp [1] which uses a rollback mechanism invoked only when needed for synchronization. The essential problem to address when designing an algorithm for distributed simulation is to maintain causality between events. In the physical system, event A might have a direct causal effect on event B. When these two events are executed on two separate processors, it is non-trivial to efficiently make sure that event A actually occurs before B in real time. Time Warp maintains this causality by restoring a previous state and re-executing any operations it finds to have violated causality. The next section describes the algorithm in more detail.

1.1 Time Warp

The basic idea behind Time Warp is to allow each LP to advance forward as fast as it can without regard to the operation of the other LPs in the system. A TW process will choose the message with the minimum timestamp in its input queue; set its local clock to the time on that message; process the message; then find the next smallest message in the queue, etc. It is possible that a "straggler" message could arrive with a timestamp less than the local clock time of the LP. When this happens, the process is forced to "roll back" to a time before the timestamp of the arriving message This is able to be accomplished because the system periodically saves the state of the LP. Any effects of having advanced too far (i.e. erroneous messages) are canceled through an elegant technique called anti-messages [1]. Any possible gain from the aggressive behavior of the Time Warp mechanism does not come without a cost. One of these costs is the overhead associated with the aforementioned state saving. There are two performance tradeoffs to keep in mind when choosing the frequency of state saving. If we save state very often, we pay a large time penalty in real time for all the data saving operations. If we choose to save state less often, we run the risk of having to roll back much further into the simulation time past than the time of the message causing the rollback, thus paying the real time cost of re-executing correct events. Lin and Lazowska [6] address exactly this issue and find an optimum state saving interval based on certain assumptions about the arrival of messages and state saving costs etc. We don't examine this tradeoff in our work. Rather, we force each processor to save state after the execution of every event so as to keep the model tractable. Another overhead of state saving is the space required to save the history of the LP. Fortunately, we do not need to keep all state information back to the beginning of the run. A concept called Global Virtual Time (GVT) [1] allows the system to periodically throw away obsolete information. GVT is defined as the minimum of all the local LP clocks and the timestamps of all messages in transit. Since nothing in the system has a timestamp less than GVT, no process could ever be forced to roll back to a time prior to GVT. Obviously GVT is a very difficult measure to obtain, since we cannot take a "global" snapshot of this distributed system [7]. Algorithms have been developed to calculate a lower bound on GVT [8] which can be used as an estimate to free up memory space.

1.2 Previous Work

Our research focuses on the analysis of the average case behavior of Time Warp when run on two processors. We will solve explicitly for the average rate of progress of simulation time per unit real time of a two processor TW system. Using this measure we find the speedup of a two processor TW system over the execution of the processes on a single processor. We further solve for interesting performance measures like

the average state buffer occupancy. Very little work has appeared in the literature which discusses average case behavior of TW. Lavenberg et al. [9] and Mitra and Mitrani [10] have examined models similar to ours, and we will address their relationship to this work in Section 6. Madisetti [11] [12] provides bounds on the performance of a two processor system where the processors have different speeds of processing and move at constant rates. Madisetti extends his model to multiple processors, something we do not address in this work. Lin and Lazowska [13] have examined Time Warp and conservative methods by appealing to critical path analysis. They have also examined TW itself [6] [14] [15] to better understand the state saving overhead, rollback mechanisms and processor scheduling when running the TW algorithm. Though their work provides important insights, it generates different types of results than ours. Nicol [2] has provided bounds on the performance of parallel "self-initiating" models where the processors schedule their own state recalculation. Finally, an overview of the results of this paper appeared in an earlier work [16].

The next section introduces our model for Time Warp. Section 3 provides its exact solution. In Section 4 we examine some performance measures including speedup. In Section 5 we take limits on various parameters, and we discuss the relationship of this work to that of Lavenberg et. al and Mitra and Mitrani in Section 6. In Section 7 we examine a restricted version of the model in detail. Finally, in Sections 8 and 9 we provide some concluding remarks and notes on future research directions.

2 A Model for Time Warp on Two Processors

Assume we have a job that is partitioned into two processes, each of which is executed on a separate processor. As these processes are executed, we consider that they advance along the integers on the x-axis in discrete steps, each beginning at $x = 0$ at time $t = 0$. Each process independently makes jumps forward on the axis where the (integer) size of the jump is geometrically distributed with mean $1/\beta_i$ $(i = 1, 2)$. The amount of real time between jumps is a geometrically distributed number of time slots with parameter α_i $(i = 1, 2)$. After process i makes an advance along the axis, it will send a message to the other process with probability q_i $(i = 1, 2)$. Upon receiving a message from the other (sending) process, this (receiving) process will do one of the following:

1: If its position along the x-axis is equal to or behind the sending process, it ignores the message.

2: If it is ahead of the sending process, it will immediately move back (i.e., "rollback") along the x-axis to the current position of the sending process.

Let

$F(t)$ = the position (on the x-axis) of the First process (process one) at time t

and

$S(t)$ = the position of the Second process (process two) at time t.

Further, let

$$D(t) = F(t) - S(t)$$

$D(t) = 0$ whenever Case 2, a rollback, occurs. $D(t)$ is a Markov process whose behavior we are interested in studying. From our assumptions that $F(0) = S(0) = 0$, we have $D(0) = 0$. Clearly, $D(t)$ can take on any integer value (i.e., it certainly can go negative, see Figure 1). [! ! **INSERT Figure 1 HERE** ! !] We will solve for

$$p_k = \lim_{t\to\infty} P[D(t) = k] \quad k = 0, 1, 2, \ldots$$
$$n_k = \lim_{t\to\infty} P[D(t) = -k] \quad k = 1, 2, 3, \ldots$$

namely, the equilibrium probability for the Markov chain $D(t)$. Moreover, we will find the speedup with which the computation proceeds when using two processors relative to the use of a single processor.

This is a simple model of the Time Warp distributed simulation algorithm where two processors are both working on a simulation job in an effort to speed it up. They both proceed independently until such time as one (behind) process transmits a message in the "past" of the other (ahead) process. This causes the faster process to "rollback" to the point where the slower process is located, after which they advance independently again until the next rollback.

The interpretation of the model is that the position of a process on the axis is the value of the local clock of that process. The amount of real time to execute a particular event is modelled by the geometric distribution of time slots between jumps. The geometrically distributed jumps along the axis indicate the increase in the virtual timestamp from one event to the next. Messages passed between processors (with probability q_i) have virtual time stamps equal to the virtual time of the sending process. The messages convey only synchronization information; they do not carry any work. We assume that each LP is always able to perform work regardless of whether any messages have arrived. The messages are used to synchronize the processors and may be thought of as carrying state information only, not extra work.

Our model assumes that states are stored after every event, otherwise a rollback would not necessarily send the processor back to the time of the tardy message; rather it might have to rollback to a much earlier time, namely, that of the last saved state. Another implicit assumption is that each process always schedules events for itself. We assume that communication between processors incurs no delay from transmission to reception. Finally, our model states that messages that arrive in the virtual-time future are ignored. They are not queued, nor do they require work in the future. Messages (at any time) are only used for synchronization. We are mainly interested in the cost of the synchronization itself, not communication. For a model which takes into account the queueing of message see our extensions to this work [17].

It should be easy to see that this system will always make progress. Even if each processor always sends a message to the other ($q_1 = q_2 = 1$), the Global Virtual Time of the system will always advance. For example, imagine that processor one (P_1) is at point x_1 on the axis (virtual time) and processor two (P_2) is at x_2 where $x_1 < x_2$ so that P_1 is behind P_2. By the definition of our model, when a processor completes an event it moves by at least one step and possibly sends a message. Therefore, P_2 can only send a message to P_1 at a time which is greater than or equal to $x_2 + 1$. Therefore, P_2's actions at the present time can not roll back P_1 or GVT. P_1, on the other hand, precisely determines the value of GVT and the rate at which it advances at the present moment. Since P_1 will advance at least one step when it advances, GVT will make progress and the system will make progress. It should be noted here that the two-processor system is attractive since it will not suffer from "cascading rollbacks" [18] where a processor that gets rolled back causes another to roll back and so on.

3 Discrete Time, Discrete State Analysis

In this section we provide the exact solution for the discrete time, discrete state model introduced in Section 2. Although, as we proceed, the equations may look formidable, the analysis is quite straightforward. First, we provide some definitions.

$$
\begin{aligned}
\alpha_i &= \mathrm{P}[i^{th} \text{ processor advances in a time slot}] \\
\overline{\alpha}_i &= 1 - \alpha_i \\
A_1 &= \alpha_1\overline{\alpha}_2 \text{ (Only processor 1 advances in a time slot)} \\
A_2 &= \alpha_2\overline{\alpha}_1 \text{ (Only processor 2 advances in a time slot)} \\
A_3 &= \alpha_1\alpha_2 \text{ (Both processors advance in a time slot)} \\
A_4 &= \overline{\alpha}_1\overline{\alpha}_2 \text{ (Neither processor advances in a time slot)} \\
f_j &= \mathrm{P}[\text{Processor 1 advances } j \text{ units} \mid \text{it advances}] \\
&= \beta_1\overline{\beta}_1^{j-1}(j > 0) \quad (\overline{\beta}_1 = 1 - \beta_1)
\end{aligned}
$$

$$
\begin{aligned}
g_j &= \text{P[Processor 2 advances } j \text{ units} \mid \text{it advances]} \\
&= \beta_2 \overline{\beta}_2^{\,j-1} (j > 0) \quad (\overline{\beta}_2 = 1 - \beta_2) \\
\pi &= \text{P[Procs. 1 and 2 advance the same dist.} \mid \text{both advance]} \\
&= \frac{\beta_1 \beta_2}{1 - \overline{\beta}_1 \overline{\beta}_2} \\
q_i &= \text{P[} i^{th} \text{ processor sends a message after advancing]} \\
\overline{q}_i &= 1 - q_i
\end{aligned}
$$

Since the transitions in our system are quite complex (there are an infinite number of transitions into and out of each state) we choose to show the state diagram only for a simplified version of our system where $\beta_1 = \beta_2 = 1$ in Figure 2. [! ! **INSERT Figure 2 HERE** ! !] This is the case where the processors only make jumps of a single step (from k to $k+1$).

The steady-state balance equations for our completely general system (no restrictions on β_i) are:

$$
\begin{aligned}
[A_1 + A_2 + A_3(1 - \pi + q_2\pi)]\, p_k &= A_1 \left(\sum_{i=0}^{k-1} p_i f_{k-i} + \sum_{i=1}^{\infty} n_i f_{k+i} \right) \\
&\quad + A_2 \overline{q}_2 \sum_{i=k+1}^{\infty} p_i g_{i-k} \\
&\quad + A_3 \overline{q}_2 \sum_{i=k+1}^{\infty} p_i \sum_{j=1}^{\infty} f_j g_{j+i-k} \\
&\quad + A_3 \overline{q}_2 \sum_{i=0}^{k-1} p_i \sum_{j=1}^{\infty} f_{j+k-i} g_j \\
&\quad + A_3 \overline{q}_2 \sum_{i=1}^{\infty} n_i \sum_{j=1}^{\infty} f_{j+k+i} g_j \quad k \geq 1
\end{aligned} \tag{1}
$$

$$
\begin{aligned}
[A_1 + A_2 + A_3(1 - \pi + q_1\pi)]\, n_k &= A_2 \left(\sum_{i=0}^{k-1} n_i g_{k-i} + \sum_{i=1}^{\infty} p_i g_{k+i} \right) \\
&\quad + A_1 \overline{q}_1 \sum_{i=k+1}^{\infty} n_i f_{i-k} \\
&\quad + A_3 \overline{q}_1 \sum_{i=k+1}^{\infty} n_i \sum_{j=1}^{\infty} g_j f_{j+i-k} \\
&\quad + A_3 \overline{q}_1 \sum_{i=0}^{k-1} n_i \sum_{j=1}^{\infty} g_{j+k-i} f_j \\
&\quad + A_3 \overline{q}_1 \sum_{i=1}^{\infty} p_i \sum_{j=1}^{\infty} g_{j+k+i} f_j \quad k \geq 1
\end{aligned} \tag{2}
$$

$$
p_0 = 1 - \sum_{i=1}^{\infty} p_i - \sum_{i=1}^{\infty} n_i
$$

The above equations will have a steady-state solution only if $q_1, q_2 > 0$ and $\alpha_1, \alpha_2 > 0$. These are reasonable limitations. The alphas must be greater than zero or else each processor will take an infinite amount of time to process an event. Each of the q_is must be greater than zero so that each processor has some positive probability of being rolled back when it gets ahead.

We define the following z-transforms

$$P(z) = \sum_{k=1}^{\infty} p_k z^k, \quad Q(z) = \sum_{k=1}^{\infty} n_k z^k, \quad \text{and } F(z) = \sum_{k=1}^{\infty} f_k z^k = \frac{\beta_1 z}{(1 - \overline{\beta}_1 z)}.$$

Using Equation 1 we solve for $P(z)$ by multiplying each term by z^k and summing from $k = 1$ to ∞. Since this equation has so many summations, we list the solution for each term separately and omit the derivations. (The derivations may be found in [19].) Also, since $P(z)$ and $Q(z)$ are symmetric with respect to (α_1, α_2),(β_1, β_2) and (q_1, q_2), we only need to solve explicitly for $P(z)$.

$$\begin{aligned}
\sum_{k=1}^{\infty} (A_1 + A_2 + A_3 - A_3\overline{q}_2\pi)\, p_k z^k &= \left(A_1 + A_2 + A_3 - A_3\overline{q}_2 \frac{\beta_1\beta_2}{1 - \overline{\beta}_1\overline{\beta}_2}\right) P(z) \\
\sum_{k=1}^{\infty} z^k A_1 \sum_{i=0}^{k-1} p_i f_{k-i} &= A_1 F(z)\,(P(z) + p_0) \\
A_1 \sum_{k=1}^{\infty} z^k \sum_{i=1}^{\infty} n_i f_{k+i} &= A_1 F(z) Q(\overline{\beta}_1) \\
A_2\overline{q}_2 \sum_{k=1}^{\infty} z^k \sum_{i=k+1}^{\infty} p_i g_{i-k} &= \frac{A_2\overline{q}_2\beta_2}{z - \overline{\beta}_2}\left(P(z) - \frac{zP(\overline{\beta}_2)}{\overline{\beta}_2}\right) \\
A_3\overline{q}_2 \sum_{k=1}^{\infty} z^k \sum_{i=k+1}^{\infty} p_i \sum_{j=1}^{\infty} f_j g_{j+i-k} &= \frac{A_3\overline{q}_2\beta_1\beta_2\overline{\beta}_2}{(1 - \overline{\beta}_1\overline{\beta}_2)(z - \overline{\beta}_2)}\left(P(z) - \frac{zP(\overline{\beta}_2)}{\overline{\beta}_2}\right) \\
A_3\overline{q}_2 \sum_{k=1}^{\infty} z^k \sum_{i=0}^{k-1} p_i \sum_{j=1}^{\infty} f_{j+k-i} g_j &= \frac{A_3\overline{q}_2\overline{\beta}_1\beta_2 F(z)}{(1 - \overline{\beta}_1\overline{\beta}_2)}\,(P(z) + p_0) \\
A_3\overline{q}_2 \sum_{k=1}^{\infty} z^k \sum_{i=1}^{\infty} n_i \sum_{j=1}^{\infty} g_j f_{j+k+i} &= \frac{A_3\overline{q}_2\overline{\beta}_1\beta_2 F(z) Q(\overline{\beta}_1)}{1 - \overline{\beta}_1\overline{\beta}_2}
\end{aligned}$$

Finally, by combining terms, we arrive at the following equation which defines $P(z)$.

$$\begin{aligned}
(A_1 + A_2 + A_3 - A_3\overline{q}_2\pi)\, P(z) = {} & A_1 F(z)\,(P(z) + p_0 + Q(\overline{\beta}_1)) \\
& + \frac{A_2\overline{q}_2\beta_2}{z - \overline{\beta}_2}\left(P(z) - \frac{zP(\overline{\beta}_2)}{\overline{\beta}_2}\right) \\
& + \frac{A_3\overline{q}_2\beta_1\beta_2\overline{\beta}_2}{(1 - \overline{\beta}_1\overline{\beta}_2)(z - \overline{\beta}_2)}\left(P(z) - \frac{zP(\overline{\beta}_2)}{\overline{\beta}_2}\right) \\
& + \frac{A_3\overline{q}_2\overline{\beta}_1\beta_2 F(z)}{(1 - \overline{\beta}_1\overline{\beta}_2)}\,(P(z) + p_0 + Q(\overline{\beta}_1))
\end{aligned}$$

We simplify this equation by solving for $P(z) = N(z)/D(z)$, a ratio of two polynomials, explicitly.

$$\begin{aligned} N(z) &= z\left(\beta_2\left(A_3\beta_1\overline{\beta}_2 + A_2\left(1-\overline{\beta}_1\overline{\beta}_2\right)\right)\overline{q}_2\left(1-\overline{\beta}_1 z\right)P(\overline{\beta}_2)\right. \\ &\quad \left. -\beta_1\overline{\beta}_2\left(A_1\left(1-\overline{\beta}_1\overline{\beta}_2\right)+A_3\overline{\beta}_1\beta_2\overline{q}_2\right)\left(z-\overline{\beta}_2\right)\left(p_0+Q(\overline{\beta}_1)\right)\right) \end{aligned} \tag{3}$$

$$\begin{aligned} D(z) &= \overline{\beta}_2\left(1-\overline{\beta}_1\overline{\beta}_2\right)\left[\left(A_1+(A_2+A_3)\overline{\beta}_1\right)z^2\right. \\ &\quad -\left(A_1\beta_1\overline{\beta}_2+(A_1+A_2+A_3)\left(1+\overline{\beta}_1\overline{\beta}_2\right)+\beta_2\overline{q}_2\left(A_2\overline{\beta}_1-A_3\beta_1\right)\right)z \\ &\quad \left.+\overline{\beta}_2(A_1+A_2+A_3)+A_2\beta_2\overline{q}_2\right] \end{aligned} \tag{4}$$

We simplify $D(z)$ to

$$D(z) = \overline{\beta}_2\left(1-\overline{\beta}_1\overline{\beta}_2\right)a_p\left(z-r_1\right)\left(z-r_2\right) \tag{5}$$

where r_1 and r_2 are the roots of the quadratic expression in $D(z)$.

Using the quadratic equation we solve for the roots (r_1, r_2) as given below.

$$(r_1, r_2) = \frac{-b_p \pm \sqrt{{b_p}^2 - 4a_pc_p}}{2a_p}$$

where

$$\begin{aligned} a_p &= A_1 + \overline{\beta}_1(A_2+A_3) \\ b_p &= -((A_1+A_2+A_3)(1+\overline{\beta}_1\overline{\beta}_2) + A_1\beta_1\overline{\beta}_2 + \beta_2\overline{q}_2(A_2\overline{\beta}_1 - A_3\beta_1)) \\ c_p &= \overline{\beta}_2(A_1+A_2+A_3) + A_2\beta_2\overline{q}_2 \end{aligned}$$

Since $Q(z)$ is symmetric to $P(z)$, we proceed to find the roots, (s_1, s_2) of the denominator of $Q(z)$ to get

$$(s_1, s_2) = \frac{-b_n \pm \sqrt{{b_n}^2 - 4a_nc_n}}{2a_n}$$

where

$$\begin{aligned} a_n &= A_2 + \overline{\beta}_2(A_1+A_3) \\ b_n &= -((A_1+A_2+A_3)(1+\overline{\beta}_1\overline{\beta}_2) + A_2\beta_2\overline{\beta}_1 + \beta_1\overline{q}_1(A_1\overline{\beta}_2 - A_3\beta_2)) \\ c_n &= \overline{\beta}_1(A_1+A_2+A_3) + A_1\beta_1\overline{q}_1 \end{aligned}$$

In Section 3.1 we show that r_1 and r_2 are real, $r_1 \geq 1$ and $0 \leq r_2 \leq 1$. Since $P(z) \triangleq \sum_{i=1}^{\infty} p_i z^i$ is the transform of a probability distribution, we know that $P(z)$ must be analytic whenever $\mid z \mid \leq 1$. Therefore, the numerator of $P(z)$ must equal zero when $z = r_2$ since $r_2 \leq 1$. Plugging r_2 into Equation 3 and setting it equal to zero we solve for $P(\overline{\beta}_2)$

$$P(\overline{\beta}_2) = \frac{\beta_1\overline{\beta}_2 D_p(r_2-\overline{\beta}_2)\left(p_0+Q(\overline{\beta}_1)\right)}{\beta_2\overline{q}_2\left(A_3\beta_1\overline{\beta}_2 + A_2(1-\overline{\beta}_1\overline{\beta}_2)\right)(1-\overline{\beta}_1 r_2)}.$$

D_p and D_n are constants given below. We then substitute $P(\overline{\beta}_2)$ back into $N(z)$ (Equation 3) resulting in the following equation.

$$N(z) = \frac{\beta_1\overline{\beta}_2\left(1-\overline{\beta}_1\overline{\beta}_2\right)D_p z\left(r_2-z\right)\left(p_0+Q(\overline{\beta}_1)\right)}{1-\overline{\beta}_1 r_2}$$

Therefore,

$$P(z) = \frac{\beta_1 D_p z \left(p_0 + Q(\overline{\beta}_1)\right)}{a_p \left(1 - \overline{\beta}_1 r_2\right)(r_1 - z)} \tag{6}$$

and by symmetry

$$Q(z) = \frac{\beta_2 D_n z \left(p_0 + P(\overline{\beta}_2)\right)}{a_n \left(1 - \overline{\beta}_2 s_2\right)(s_1 - z)} \tag{7}$$

where

$$D_p = A_1 \left(1 - \overline{\beta}_1 \overline{\beta}_2\right) + A_3 \overline{\beta}_1 \beta_2 \overline{q}_2 \tag{8}$$

$$D_n = A_2 \left(1 - \overline{\beta}_1 \overline{\beta}_2\right) + A_3 \beta_1 \overline{\beta}_2 \overline{q}_1. \tag{9}$$

We solve for the unknown constants $P(\overline{\beta}_2)$ and $Q(\overline{\beta}_1)$ by solving Equations 6 and 7 simultaneously with z replaced by $\overline{\beta}_1$ and $\overline{\beta}_2$ respectively.

$$P(\overline{\beta}_2) = K_p \left(p_0 + Q(\overline{\beta}_1)\right)$$
$$Q(\overline{\beta}_1) = K_n \left(p_0 + P(\overline{\beta}_2)\right)$$

Solving them simultaneously yields.

$$P(\overline{\beta}_2) = \frac{p_0 K_p (1 + K_n)}{1 - K_p K_n}$$
$$Q(\overline{\beta}_1) = \frac{p_0 K_n (1 + K_p)}{1 - K_p K_n}$$

where

$$K_p = \frac{\beta_1 \overline{\beta}_2 D_p}{a_p \left(r_1 - \overline{\beta}_2\right)\left(1 - \overline{\beta}_1 r_2\right)} \tag{10}$$

$$K_n = \frac{\overline{\beta}_1 \beta_2 D_n}{a_n \left(s_1 - \overline{\beta}_1\right)\left(1 - \overline{\beta}_2 s_2\right)} \tag{11}$$

Substituting these values into Equations 6 and 7 and simplifying we find

$$P(z) = \frac{z p_0 C_p}{r_1 - z} \tag{12}$$

$$Q(z) = \frac{z p_0 C_n}{s_1 - z} \tag{13}$$

where

$$C_p = \frac{\beta_1 D_p (1 + K_n)}{(1 - K_n K_p) a_p \left(1 - \overline{\beta}_1 r_2\right)} \tag{14}$$

$$C_n = \frac{\beta_2 D_n (1 + K_p)}{(1 - K_n K_p) a_n \left(1 - \overline{\beta}_2 s_2\right)} \tag{15}$$

By conservation of probability we know that $P(1) + Q(1) + p_0 = 1$. Therefore,

$$p_0 = \frac{1}{1 + \left(\frac{C_p}{r_1 - 1}\right) + \left(\frac{C_n}{s_1 - 1}\right)} \tag{16}$$

Finally, we invert the z-transforms to get our final answer.

$$p_k = C_p p_0 \left(\frac{1}{r_1}\right)^k \quad k \geq 1 \tag{17}$$

$$n_k = C_n p_0 \left(\frac{1}{s_1}\right)^k \quad k \geq 1 \tag{18}$$

3.1 Root Locus

In this section we show that r_1 and r_2 are real, $r_1 \geq 1$ and $0 \leq r_2 \leq 1$. To show that the roots r_1 and r_2 are real, we must show that the quantity under the square root is greater than or equal to zero, or that

$$b_p^2 - 4a_p c_p \geq 0$$

Substituting in the values for a_p, b_p, c_p and simplifying, we find that the roots will be real if the following inequality is satisfied.

$$\begin{aligned}&(\beta_2\alpha_1 - \beta_1\alpha_2)^2 + {\beta_2}^2{\alpha_2}^2{q_2}^2(\overline{\beta}_1 - \alpha_1)^2 \\ &+ 2\beta_2\alpha_2 q_2\left(\beta_2\alpha_1\overline{\alpha}_1 + \beta_1\left(\alpha_1(2 - \beta_2 - \alpha_2) + \overline{\beta}_1\alpha_2\right)\right) \geq 0\end{aligned} \tag{19}$$

Since all the factors on the left-hand side of Equation 19 are non-negative, the inequality must hold. Therefore, both r_1 and r_2 are real roots.

We now show that $r_1 \geq 1$. Assuming it is true, we require

$$\begin{aligned} r_1 = \frac{-b_p + \sqrt{{b_p}^2 - 4a_p c_p}}{2a_p} &\geq 1 \\ -b_p + \sqrt{{b_p}^2 - 4a_p c_p} &\geq 2a_p \\ \sqrt{{b_p}^2 - 4a_p c_p} &\geq 2a_p + b_p \\ {b_p}^2 - 4a_p c_p &\geq 4{a_p}^2 + 4a_p b_p + {b_p}^2 \\ 0 &\geq a_p + b_p + c_p \end{aligned}$$

Substituting in the values for a_p, b_p and c_p we arrive at the condition

$$0 \geq -\beta_1\beta_2\alpha_2 q_2 \tag{20}$$

Since all the terms on the right-hand side of Equation 20 are non-negative, the inequality holds.

To show that $r_2 \leq 1$ we need to prove the following

$$\begin{aligned} r_2 = \frac{-b_p - \sqrt{{b_p}^2 - 4a_p c_p}}{2a_p} &\leq 1 \\ -b_p - \sqrt{{b_p}^2 - 4a_p c_p} &\leq 2a_p \\ -b_p - 2a_p &\leq \sqrt{{b_p}^2 - 4a_p c_p} \\ {b_p}^2 + 4a_p b_p + 4{a_p}^2 &\leq {b_p}^2 - 4a_p c_p \\ a_p + b_p + c_p &\leq 0 \end{aligned}$$

which was shown above to be true.

Finally, we show that $r_2 \geq 0$. We require

$$\begin{aligned} r_2 = \frac{-b_p - \sqrt{{b_p}^2 - 4a_pc_p}}{2a_p} &\geq 0 \\ -b_p - \sqrt{{b_p}^2 - 4a_pc_p} &\geq 0 \\ b_p + \sqrt{{b_p}^2 - 4a_pc_p} &\leq 0 \text{ (multiply by } -1) \\ \sqrt{{b_p}^2 - 4a_pc_p} &\leq -b_p \\ -4a_pc_p &\leq 0 \end{aligned} \tag{21}$$

Since a_p and c_p are non-negative, the inequality in Equation 21 holds, thus proving that $r_2 \geq 0$. Similar proofs follow for (s_1, s_2), the roots of $Q(z)$.

4 Performance Measures

With the complete solution to the Markov chain in hand (Equations 16, 17 and 18), we calculate several interesting performance measures. The first is $\overline{K}_1$ which is defined as the average distance processor one is ahead of processor two, given that processor one is ahead. This measure is useful in determining the number of states that will need to be saved on average as we will see below.

$$\begin{aligned} \overline{K}_1 &= \frac{\sum_{k=1}^{\infty} kp_k}{\sum_{k=1}^{\infty} p_k} \\ &= \frac{r_1}{r_1 - 1} \end{aligned}$$

We find a symmetric value for processor two.

$$\begin{aligned} \overline{K}_2 &= \frac{\sum_{k=1}^{\infty} kn_k}{\sum_{k=1}^{\infty} n_k} \\ &= \frac{s_1}{s_1 - 1} \end{aligned}$$

Since we know the expected size of a state jump at processor one is $1/\beta_1$, we find that the expected number of buffers needed for state saving when processor one is ahead is

$$\begin{aligned} \text{E[Buffers needed when Proc. 1 is ahead]} &= \frac{\overline{K}_1}{\frac{1}{\beta_1}} \\ &= \frac{r_1\beta_1}{r_1 - 1} \end{aligned} \tag{22}$$

and for processor two

$$\begin{aligned} \text{E[Buffers needed when Proc. 2 is ahead]} &= \frac{\overline{K}_2}{\frac{1}{\beta_2}} \\ &= \frac{s_1\beta_2}{s_1 - 1} \end{aligned} \tag{23}$$

Another useful measure, $\Theta_{1,b}$, is the probability that processor one needs more than b buffers for storing state.

$$\begin{aligned}
\Theta_{1,b} &= \text{P[Proc. 1 needs} > b \text{ buffers]} \\
&= \sum_{i=b+1}^{\infty} \text{P[Proc. 1 is using } i \text{ buffers]} \\
&= \sum_{i=b+1}^{\infty} \sum_{k=i}^{\infty} \text{P[Proc. 1 is using } i \text{ buffers} \mid \text{Proc. 1 is } k \text{ units ahead}]\ p_k \\
&= \sum_{i=b+1}^{\infty} \sum_{k=i}^{\infty} \text{P[Sum of } i \text{ geometric random variables} = k]\ p_k \\
&= \sum_{i=b+1}^{\infty} \sum_{k=i}^{\infty} \binom{k-1}{i-1} \beta_1^i \overline{\beta}_1^{k-i} C_p p_0 \left(\frac{1}{r_1}\right)^k \\
&= C_p p_0 \left(\frac{\beta_1}{r_1 - 1}\right) \left(\frac{\beta_1}{r_1 - \overline{\beta}_1}\right)^b
\end{aligned} \tag{24}$$

If $\beta_1 = 1$ (single step state jumps), then $\Theta_{1,b}$ reduces to the following expression.

$$\Theta_{1,b} = \sum_{k=b+1}^{\infty} p_k = \frac{C_p p_0}{r_1{}^b (r_1 - 1)} \tag{25}$$

A similar (symmetric) value, $\Theta_{2,b}$, can be found for processor two. The quantity of most interest though is speedup, and we calculate its value in the next section.

4.1 Speedup

Using the formulae for p_k and n_k we calculate the speedup S when using two processors versus using only one. S is the rate of progress when using two processors (R_2) divided by the rate of progress when using only one processor (R_1). The rate of forward progress for one processor (obviously not running Time Warp) is defined as the average rate of virtual time progress per real time step of the two processes defined earlier by the real time and virtual time geometric distributions (Section 2). Since process one completes events (on average) every $1/\alpha_1$ seconds and process two does so every $1/\alpha_2$ seconds and they make jumps in virtual time of distance $1/\beta_1$ and $1/\beta_2$ respectively, then the average rate of virtual time progress would be

$$R_1 \triangleq \frac{\frac{\alpha_1}{\beta_1} + \frac{\alpha_2}{\beta_2}}{2} = \frac{\alpha_1\beta_2 + \alpha_2\beta_1}{2\beta_1\beta_2}$$

The average rate of forward progress for two processors is the expected "unfettered" rate of progress (without rollbacks) per time step minus the (rollback-distance-weighted) expected rollback rate per time step for the two processors. The first three terms (positive terms) in the following expression give the forward rate while the negative terms give the rollback rate. The negative terms are derived by noting that when a process advances f units and causes the other to rollback r units, the net progress is given by the difference $(f-r)$.

$$\begin{aligned}
R_2 &= \frac{A_1}{\beta_1}+\frac{A_2}{\beta_2}+A_3\left(\frac{1}{\beta_1}+\frac{1}{\beta_2}\right) \\
&\quad -A_3q_2p_0\sum_{i=2}^{\infty}f_i\sum_{j=1}^{i-1}g_j(i-j)-A_3q_1p_0\sum_{i=2}^{\infty}g_i\sum_{j=1}^{i-1}f_j(i-j) \\
&\quad -A_2q_2\sum_{k=1}^{\infty}p_k\sum_{i=1}^{k-1}ig_{k-i}-A_1q_1\sum_{k=1}^{\infty}n_k\sum_{i=1}^{k-1}if_{k-i} \\
&\quad -A_3q_2\sum_{k=1}^{\infty}p_k\sum_{i=1}^{\infty}f_i\sum_{j=1}^{k+i-1}jg_{k+i-j}-A_3q_1\sum_{k=1}^{\infty}n_k\sum_{i=1}^{\infty}g_i\sum_{j=1}^{k+i-1}jf_{k+i-j} \\
&\quad -A_3q_1\sum_{k=1}^{\infty}p_k\sum_{i=1}^{\infty}f_i\sum_{j=1}^{\infty}jg_{k+i+j}-A_3q_2\sum_{k=1}^{\infty}n_k\sum_{i=1}^{\infty}g_i\sum_{j=1}^{\infty}jf_{k+i+j}
\end{aligned}$$

As with the $P(z)$ calculation we list the solution for each term separately, but since each pair of terms above is symmetric with respect to f and g we only need to derive the closed-form solution for one of the two. The derivations can be found in [19].

$$\begin{aligned}
A_3q_2p_0\sum_{i=2}^{\infty}f_i\sum_{j=1}^{i-1}g_j(i-j) &= \frac{A_3\overline{\beta}_1\beta_2q_2p_0}{\beta_1(1-\overline{\beta}_1\overline{\beta}_2)} \\
A_2q_2\sum_{k=1}^{\infty}p_k\sum_{i=1}^{k-1}ig_{k-i} &= \frac{A_2q_2\beta_2C_pp_0r_1}{(r_1-\overline{\beta}_2)(r_1-1)^2} \\
A_3q_2\sum_{k=1}^{\infty}p_k\sum_{i=1}^{\infty}f_i\sum_{j=1}^{k+i-1}jg_{k+i-j} &= \frac{A_3q_2\beta_2C_pp_0}{(1-\overline{\beta}_1\overline{\beta}_2)(r_1-1)^2}\left(\frac{(r_1-\overline{\beta}_1)}{\beta_1}+\frac{\beta_1\overline{\beta}_2r_1}{(r_1-\overline{\beta}_2)}\right) \\
A_3q_1\sum_{k=1}^{\infty}p_k\sum_{i=1}^{\infty}f_i\sum_{j=1}^{\infty}jg_{k+i+j} &= \frac{A_3q_1\beta_1\overline{\beta}_2^2C_pp_0}{\beta_2(1-\overline{\beta}_1\overline{\beta}_2)(r_1-\overline{\beta}_2)}
\end{aligned}$$

Finally, combining all the terms together we find the formula for speedup.

$$
\begin{aligned}
S = \quad & \left(\frac{2\beta_1\beta_2}{\alpha_1\beta_2+\alpha_2\beta_1}\right)\left[\frac{A_1}{\beta_1}+\frac{A_2}{\beta_2}+A_3\left(\frac{1}{\beta_1}+\frac{1}{\beta_2}\right)\right. \\
& -\frac{A_3\overline{\beta}_1\beta_2 q_2 p_0}{\beta_1(1-\overline{\beta}_1\overline{\beta}_2)}-\frac{A_3\overline{\beta}_2\beta_1 q_1 p_0}{\beta_2(1-\overline{\beta}_1\overline{\beta}_2)} \\
& -\frac{A_2 q_2\beta_2 C_p p_0 r_1}{(r_1-\overline{\beta}_2)(r_1-1)^2}-\frac{A_1 q_1\beta_1 C_n p_0 s_1}{(s_1-\overline{\beta}_1)(s_1-1)^2} \\
& -\frac{A_3 q_2\beta_2 C_p p_0}{(1-\overline{\beta}_1\overline{\beta}_2)(r_1-1)^2}\left(\frac{(r_1-\overline{\beta}_1)}{\beta_1}+\frac{\beta_1\overline{\beta}_2 r_1}{(r_1-\overline{\beta}_2)}\right) \\
& -\frac{A_3 q_1\beta_1 C_n p_0}{(1-\overline{\beta}_1\overline{\beta}_2)(s_1-1)^2}\left(\frac{(s_1-\overline{\beta}_2)}{\beta_2}+\frac{\beta_2\overline{\beta}_1 s_1}{(s_1-\overline{\beta}_1)}\right) \\
& \left.-\frac{A_3 q_1\beta_1\overline{\beta}_2^2 C_p p_0}{\beta_2(1-\overline{\beta}_1\overline{\beta}_2)(r_1-\overline{\beta}_2)}-\frac{A_3 q_2\beta_2\overline{\beta}_1^2 C_n p_0}{\beta_1(1-\overline{\beta}_1\overline{\beta}_2)(s_1-\overline{\beta}_1)}\right]
\end{aligned}
\tag{26}
$$

5 Limiting Behavior

Before examining the results of the previous section, we explore what sorts of models arise when taking limits on the α and β parameters. The next three subsections will present the results for $(\alpha_1,\alpha_2) \to 0$ and $(\beta_1,\beta_2) \to 0$. By taking these limits we transform the geometric distributions into exponential distributions, thus moving from discrete time and state to continuous time and state.

5.1 Continuous Time, Discrete State

We transform our model into a continuous time, discrete state (CD) model by taking the limit as α_1 and $\alpha_2 \to 0$ while keeping the ratio $\frac{\alpha_1}{\alpha_2}$ constant and defining $\frac{A_1}{A_1+A_2} = a$. We can take the limit either on the p_k equations or on our formula for speedup. The final result for speedup is given below.

$$
S = 2\left(1-\frac{p_0}{\frac{a}{\beta_1}+\frac{\overline{a}}{\beta_2}}\left(\frac{C_p\overline{a}q_2\beta_2 r_1}{(r_1-\overline{\beta}_2)(r_1-1)^2}+\frac{C_n a q_1\beta_1 s_1}{(s_1-\overline{\beta}_1)(s_1-1)^2}\right)\right) \tag{27}
$$

where

$$
p_0 = \frac{1}{1+\left(\frac{C_p}{r_1-1}\right)+\left(\frac{C_n}{s_1-1}\right)}
$$

$$
\begin{aligned}
C_p &= \frac{a\beta_1(1-\overline{\beta}_1\overline{\beta}_2)(1+K_n)}{(1-K_pK_n)(1-\overline{\beta}_1 r_2)(1-\beta_1\overline{a})} \\
C_n &= \frac{\overline{a}\beta_2(1-\overline{\beta}_1\overline{\beta}_2)(1+K_p)}{(1-K_pK_n)(1-\overline{\beta}_2 s_2)(1-\beta_2 a)}
\end{aligned}
$$

$$
\begin{aligned}
K_p &= \frac{a\beta_1\overline{\beta}_2(1-\overline{\beta}_1\overline{\beta}_2)}{(1-\overline{\beta}_1 r_2)(1-\beta_1\overline{a})(r_1-\overline{\beta}_2)} \\
K_n &= \frac{\overline{a}\beta_2\overline{\beta}_1(1-\overline{\beta}_1\overline{\beta}_2)}{(1-\overline{\beta}_2 s_2)(1-\beta_2 a)(s_1-\overline{\beta}_1)}
\end{aligned}
$$

$$(r_1, r_2) = \frac{(1+\overline{\beta}_2(1-\beta_1\overline{a})+\overline{\beta}_1\beta_2\overline{aq}_2)}{2(1-\beta_1\overline{a})} \pm \frac{\sqrt{(1+\overline{\beta}_2(1-\beta_1\overline{a})+\overline{\beta}_1\beta_2\overline{aq}_2)^2 - 4(1-\beta_1\overline{a})(\overline{\beta}_2+\beta_2\overline{aq}_2)}}{2(1-\beta_1\overline{a})}$$

$$(s_1, s_2) = \frac{(1+\overline{\beta}_1(1-\beta_2 a)+\overline{\beta}_2\beta_1 a\overline{q}_1)}{2(1-\beta_2 a)} \pm \frac{\sqrt{(1+\overline{\beta}_1(1-\beta_2 a)+\overline{\beta}_2\beta_1 a\overline{q}_1)^2 - 4(1-\beta_2 a)(\overline{\beta}_1+\beta_1 a\overline{q}_1)}}{2(1-\beta_2 a)}$$

5.2 Discrete Time, Continuous State

We create a discrete time, continuous state (DC) model by taking the limit as β_1 and $\beta_2 \to 0$ while keeping $\frac{\beta_1}{\beta_2} = b$. We find the value for speedup by taking limits on the speedup formula calculated for the discrete time, discrete state model. The resulting formula is given below. We first substitute $\beta_2 = \beta_1/b$, then take the limit as $\beta_1 \to 0$. When taking the limit, we were often confronted with functions of the form 0/0 and were forced to use l'Hospital's rule repeatedly. Any terms of the form F' are a shorthand notation for $\partial F/\partial\beta_1$. We find

$$\begin{aligned} S = & \frac{2(A_1+A_3+A_2 b+A_3 b)}{\alpha_1+\alpha_2 b} - \frac{2A_3 p_0\left(b^2 q_1+q_2\right)}{(1+b)(\alpha_1+\alpha_2 b)} \\ & - \frac{2p_0 C_p'\left(A_2(1+b)q_2 + A_3\left((1+b)q_2(1+r_1') + b\left(b^2 q_1+q_2\right)r_1'^2\right)\right)}{(1+b)(\alpha_1+\alpha_2 b)r_1'^2(1+br_1')} \\ & - \frac{2p_0 C_q'\left(A_1(1+b)q_1 + A_3\left((1+b)q_1(1+bs_1') + \left(b^2 q_1+q_2\right)s_1'^2\right)\right)}{(1+b)(\alpha_1+\alpha_2 b)s_1'^2(1+s_1')} \end{aligned} \tag{28}$$

where

$$p_0 = \frac{r_1' s_1'}{C_n' r_1' + C_p' s_1' + r_1' s_1'}$$

$$C_p' = \frac{(1+K_n)D_p'}{(1-K_p K_n)(1-r_2')(A_1+A_2+A_3)}$$

$$C_n' = \frac{(1+K_p)D_n'}{(1-K_p K_n)(1-bs_2')(A_1+A_2+A_3)}$$

$$K_p = \frac{D_p'}{(1-r_2')\left(\frac{1}{b}+r_1'\right)(A_1+A_2+A_3)}$$

$$K_n = \frac{D_n'}{(1-bs_2')(1+s_1')(A_1+A_2+A_3)}$$

$$D_p' = \frac{A_1 + A_1 b + A_3 \bar{q}_2}{b}$$

$$D_n' = \frac{A_2 + A_2 b + A_3 b \bar{q}_1}{b}$$

$$\begin{aligned}(r_1', r_2') \;=\; & \left(\frac{1}{2b\,(A_1 + A_2 + A_3)}\right)\Big\{ -A_1 - A_3 + b(A_2 + A_3) - A_2 q_2 \\ & \pm \Big[(A_1 + A_3 - b(A_2 + A_3)(1 - 2q_2) + A_2 q_2)^2 \\ & + 4bq_2\bar{q}_2\,(A_2 + A_3)\,(A_2 + b(A_2 + A_3))\Big]^{\frac{1}{2}} \Big\}\end{aligned}$$

$$\begin{aligned}(s_1', s_2') \;=\; & \left(\frac{1}{2b\,(A_1 + A_2 + A_3)}\right)\Big\{ A_1 + A_3 - b(A_2 + A_3) - A_1 b q_1 \\ & \pm \Big[(A_1 + A_3 - b(A_2 + A_3))^2 \\ & + 2bq_1\left({A_1}^2 + 4A_1A_2 + 6A_1A_3 + 4A_2A_3 + 4{A_3}^2\right. \\ & \left. + 2A_1A_2b + 2A_1A_3b + {A_1}^2 bq_1\right)\Big]^{\frac{1}{2}} \Big\}\end{aligned}$$

5.3 Continuous Time, Continuous State

Finally, we solve a continuous time, continuous state (CC) model by taking limits on both α_i and β_i. This can be done either by going first to the CD (α_i) or DC (β_i) model from DD, and then finishing by taking limits on the other variable. The final equation for speedup is given below.

$$S = 2\left(1 - \frac{\bar{a}q_2 p_0 {C_p}'}{(1 + br_1')\,(a + \bar{a}b)\,{r_1}'^2} - \frac{aq_1 p_0 {C_n}'}{(1 + s_1')\,(a + \bar{a}b)\,{s_1}'^2}\right) \tag{29}$$

where

$$p_0 = \frac{r_1' s_1'}{{C_n}' r_1' + {C_p}' s_1' + r_1' s_1'}$$

$$C_p' = \frac{a\,(1 + b)\,(1 + K_n)}{b\,(1 - K_p K_n)\,(1 - r_2')}$$

$$C_n' = \frac{\bar{a}\,(1 + b)\,(1 + K_p)}{b\,(1 - K_p K_n)\,(1 - bs_2')}$$

$$K_p = \frac{a\,(1 + b)}{(1 + br_1')\,(1 - r_2')}$$

$$K_n = \frac{\bar{a}\,(1 + b)}{b\,(1 + s_1')\,(1 - bs_2')}$$

$$(r_1', r_2') = \frac{-1 + \overline{a}b + \overline{a}\overline{q}_2 \pm \sqrt{1 + 2\overline{a}b + \overline{a}^2b^2 - 2\overline{a}\overline{q}_2 - 4\overline{a}b\overline{q}_2 + 2\overline{a}^2b\overline{q}_2 + \overline{a}^2\overline{q}_2^2}}{2b}$$

$$(s_1', s_2') = \frac{a - b + ab\overline{q}_1 \pm \sqrt{a^2 + 2ab + b^2 - 4ab\overline{q}_1 + 2a^2b\overline{q}_1 - 2ab^2\overline{q}_1 + a^2b^2\overline{q}_1^2}}{2b}$$

6 Previous Work on 2-Processor Models

There has been some similar work on two processor Time Warp models. Lavenberg, Muntz and Samadi [9] used a continuous time, continuous state model to solve for the speedup (S_{lms}) of two processors over one processor. Their work resulted in an approximation for speedup that was valid only for $0 \leq q_i \leq 0.05$, where q_i is the probability that processor i will send a message to the other processor. Our result for this CC case is exact, has no restrictions on any of the parameters and therefore subsumes their work. In fact, we can compare our results directly for a simplified case where $a = 1/2$ (same processing rate for both processors), $b = 1$ (same average jump in virtual time for both) and $q_1 = q_2 = q$ (same probability of sending a message), which we refer to as the symmetric, balanced case. Lavenberg et al. derive the following approximation for speedup in this case.

$$S_{lms} \approx 2 - \sqrt{2q}$$

Our equation for speedup in this restricted case is exactly

$$S = \frac{2\left(\sqrt{8+q} - \sqrt{q}\right)}{\sqrt{8+q} + \sqrt{q}}$$

If we expand this formula using a power series about the point ($q = 0$) and list only the first few terms, we see the essential difference between our result and Lavenberg et al.

$$S \approx 2 - \sqrt{2q} + \frac{q}{2} - \frac{\sqrt{2}q^{\frac{3}{2}}}{16} + O(q^{\frac{4}{2}}) \quad .$$

This clearly shows that the Lavenberg et al. result matches ours in the first two terms. Figure 3 shows the Lavenberg et al. result and our result compared to simulation with 99% confidence intervals. [! ! **INSERT Figure 3 HERE** ! !]

Mitra and Mitrani [10] also solve a two processor model but use a discrete time, continuous state approach. They solve for the distribution of the separation between the two processors and the rate of progress of the two. In the definition of their model, a processor sends a message (with probability q_i) **before** advancing. Our model has a processor send a message **after** advancing. This difference between the two models disappears in the calculation of the average rate of progress. Their solution allows a general continuous distribution for the state jumps (virtual time), but requires (deterministic) single steps for the discrete time. In each time slot both processors always advance forward in virtual time some arbitrary distance. In our model this is equivalent to setting $\alpha_1 = \alpha_2 = 1$. Since our analysis only supports an exponential distribution for state changes, but their analysis doesn't have a distribution on time, neither model subsumes the other.

Finally, the DD and CD models have not appeared in the literature, although an early version of this work has been published by Kleinrock [20]. It is a simplified version of the CD model where $\beta_1 = \beta_2 = 1$ (which is single step state jumps). Figure 4 shows how all of this work fits together. The work discussed in this paper covers the shaded region. [! ! **INSERT Figure 4 HERE** ! !]

7 Results for a Restricted Model

In order to better understand our results, we examine a restricted version of the CD model (i.e. the model analyzed in [20]). In this less general model we eliminate two variables by forcing each processor to advance **exactly** one virtual time unit each time it advances ($\beta_1 = \beta_2 = 1$). Again, we define q_i as the interaction parameter; the probability that processor i sends a message to the other processor. We also define a as the ratio $\frac{\lambda_1}{\lambda_1+\lambda_2}$ where λ_i is the rate for the continuous time distribution for processor i (rate at which messages are processed). The parameter a can be thought of as a measure of "load balancing". When $a = 1/2$ the load is balanced.

The solution for this simplified system is given below.

$$S = 2\left(1 - p_0\left(\frac{\bar{a}q_2}{(r_1-1)^2} + \frac{aq_1}{(s_1-1)^2}\right)\right)$$

$$p_0 = \frac{1}{1 + \left(\frac{1}{r_1-1}\right) + \left(\frac{1}{s_1-1}\right)}$$

$$r_1 = \frac{1+\sqrt{1-4a\bar{a}q_2}}{2a}$$

$$s_1 = \frac{1+\sqrt{1-4a\bar{a}q_1}}{2\bar{a}}$$

The equations above indicate that speedup reaches a maximum value of two when $q_1 = q_2 = 0$ (no interaction). Since neither processor hinders the other, we can exploit the full potential of each processor. In general one might assume that the speedup would simply be twice the speed of the slowest processor. In fact, the system does a little bit worse. Even though one processor might be faster than the other, it is possible (stochastically) that the slower processor gets ahead of the faster one. At this point it is possible that the faster processor could cause the slower one to rollback. Overall therefore, the speedup is less than twice the speed of the slower processor on average.

Figure 5 shows the speedup for the symmetric case where $q_1 = q_2 = q$, [! ! **INSERT Figure 5 HERE** ! !] though it does not show the discontinuity in the function S at $q = 0$. For $q = 0$, $S = 2$ for all a and so S is discontinuous for all $a \neq 1/2$. This is not shown in the figure. For $q = 0$ no messages are sent, therefore no rollbacks will occur, and it is clear that $S = 2$. For $q > 0$ as $a \to 0$ or $a \to 1$ ($\lambda_1 \to 0$ or $\lambda_2 \to 0$), then the speedup goes to zero as shown in the figure. This occurs because one process moves extremely slowly (compared to the equivalent single process) and it will eventually drag the faster process back to its lagging position. The TW system moves at less than twice the speed of the slowest processor, while the equivalent single processor moves at the average rate $\frac{\lambda_1+\lambda_2}{2}$. It is clear that load balancing is extremely important since good speedup only occurs near $a = 1/2$. Notice that the interaction parameter is important when a is near $1/2$.

Figure 6 shows the speedup for the balanced case where $\lambda_1 = \lambda_2$. [! ! **INSERT Figure 6 HERE** ! !] Note that the speedup is 2 for $q_1 = q_2 = 0$ and goes to 4/3 for $q_1 = q_2 = 1$ Specifically, it never goes below one. We always get speedup with two processors as long as $a = 1/2$.

Figure 7 shows the speedup for the extremely simplified symmetric, balanced case where $q_1 = q_2 = q$ and $\lambda_1 = \lambda_2 = \lambda$. [! ! **INSERT Figure 7 HERE** ! !] For this special case the formula for speedup is

$$S = \frac{4}{2+\sqrt{q}}$$

Note for $q = 0$ that $S = 2$ and for $q = 1$ we have $S = 4/3$. We can see this last result intuitively. Since each process always sends a message to the other after it advances, then the time for both processes to advance one unit is equal to the maximum of two exponential delays at rate λ which is 3/2 times $1/\lambda$. Thus, the rate of progress for each process is simply $\frac{2\lambda}{3}$. Since both are moving at this rate, the sum equals $\frac{4\lambda}{3}$ while $R_1 = \lambda$ which yields $S = 4/3$ for $q = 1$. The curve shown in Figure 7 is the "spine" of the surface plotted in Figure 5 and is the "45 degree" line ($q_1 = q_2$) of the surface plotted in Figure 6.

7.1 Optimality Proofs

Using the simple model described above, we prove several results about optimality with respect to the parameters of the system. We first show that the speedup is monotonically decreasing in both q_1 and q_2, the interaction parameters (i.e. q_1 and q_2 should be as small as possible). We do this by showing that $\frac{\partial S}{\partial q_1}$ is negative. If we differentiate S with respect to q_1 we arrive at the following formula

$$\frac{\partial S}{\partial q_1} = \Phi(q_1, q_2, a)\left(-(-1+2a)^2 - 2a\bar{a}q_1 + (1-2a)\sqrt{1-4a\bar{a}\bar{q}_1}\right)$$

where $\Phi(q_1, q_2, a)$ is a non-negative function of q_1,q_2,a and is given below

$$\Phi(q_1, q_2, a) = \frac{128a^3\bar{a}^3q_2}{(-1+2\bar{a}-f(\bar{q}_1))^2 f(\bar{q}_1)\left(-1+4a\bar{a}-f(\bar{q}_1)-\sqrt{f(\bar{q}_2)}-f(\bar{q}_1)\sqrt{f(\bar{q}_2)}\right)^2}$$

where

$$f(x) = \sqrt{1-4a\bar{a}x}$$

In order to show that $\frac{\partial S}{\partial q_1}$ is negative, we must show that

$$-(-1+2a)^2 - 2a\bar{a}q_1 + (1-2a)\sqrt{1-4a\bar{a}\bar{q}_1} \leq 0 \tag{30}$$

When $a \geq \frac{1}{2}$ Equation 30 is trivially satisfied. Our concern is in the range $0 \leq a < \frac{1}{2}$, in which case our condition becomes

$$\begin{aligned}
-(-1+2a)^2 - 2a\bar{a}q_1 &\leq -(1-2a)\sqrt{1-4a\bar{a}\bar{q}_1} < 0 \\
\left(-(-1+2a)^2 - 2a\bar{a}q_1\right)^2 &\geq \left(-(1-2a)\sqrt{1-4a\bar{a}\bar{q}_1}\right)^2 \\
4a^2{q_1}^2 - 8a^3{q_1}^2 + 4a^4{q_1}^2 &\geq 0 \\
4a^2\bar{a}^2{q_1}^2 &\geq 0
\end{aligned}$$

which is trivially true. A similar (symmetric) proof for q_2 is omitted here.

Optimization with respect to a is a little more difficult. When we differentiate S with respect to a we get such a complicated formula that it is prohibitive to solve for the optimum value of a. Fortunately, by plotting S versus a, q_1 and q_2 (Figures 5 and 6) we see that S is unimodal and that the optimum value of a is 1/2 ($\lambda_1 = \lambda_2$). When we plug this value ($a = 1/2$) into $\frac{\partial S}{\partial a}$ we see that the result is 0.

$$\frac{\partial S}{\partial a}_{|a=\frac{1}{2}} = \frac{2\left(-\left(\left(1-\bar{q}_1\right)q_2\right) + q_1\left(1-\bar{q}_2\right)\right)}{\left(1-\bar{q}_1\right)\left(1-\bar{q}_2\right)} = 0$$

To show that this is a maximum we must show that the second derivative is negative at $a = 1/2$.

$$\frac{\partial^2 S}{\partial {q_1}^2}_{|a=\frac{1}{2}} = \frac{-8\left(\sqrt{q_1}+\sqrt{q_2}\right)\left(q_1+\sqrt{q_1}\sqrt{q_2}+2q_1\sqrt{q_2}+q_2+2\sqrt{q_1}q_2+q_1q_2\right)}{\sqrt{q_1}\left(\sqrt{q_1}+\sqrt{q_2}+\sqrt{q_1}\sqrt{q_2}\right)^2\sqrt{q_2}} \tag{31}$$

Equation 31 is clearly negative since the numerator is negative and the denominator is positive. For the more general case, where the processors are not restricted to single step advances, we find from analyzing plots of speedup that the above result ($a = 1/2$ ($\lambda_1 = \lambda_2$) for optimal performance) generalizes to

$$\frac{\lambda_1}{\beta_1} = \frac{\lambda_2}{\beta_2} \text{ or } b = \frac{a}{1-a} \tag{32}$$

meaning that the average "unfettered" rate of progress in virtual time for each processor should be the same. For a fixed value of a the best performance can be found when Equation 32 is true, and overall best performance is found at $a = 1/2$ with Equation 32 holding true.

We have not seen this result before in the literature since the two processor models haven't been general enough. It says that for optimum performance we would like to place tasks on processors such that the average "independent" rate of progress in virtual time is the same for both processors. Ideally we want this to be true while also having each processor execute events at the same rate ($a = 1/2$). This result is generally applicable to systems consisting of more than two processors. The intuition is that if every processor tends to move forward in virtual time at the same rate as the others, then the processors will remain nearly synchronized without suffering a large penalty for rollbacks.

7.2 Adding a Cost for State Saving

One simple way of examining how state saving overhead affects the performance of the system is to modify the value of R_1, the rate of progress on a single processor. We introduce a parameter c ($c \geq 1$) that indicates how much faster events are executed without state saving. If $c = 2$ an event completes twice as fast on average without state saving. Since our model requires that each processor save its state after every event, we can think of each event as taking longer to complete in the TW system than in the single processor system where no state saving is required. We note here that this is actually an upper bound on the cost for state saving in this two processor system. Lin and Lazowska [6] have shown that to achieve minimal state saving costs, TW should save state less often than after every event. This result depends on certain system parameters, most notably the cost for state saving. We make no attempt to optimize the frequency of state saving, nonetheless this simple model provides some interesting results as shown below.

By examining the CD model with the single step restriction (as above) we arrive at the following value for R_1

$$R_1 = \frac{c(\lambda_1 + \lambda_2)}{2}$$

For this model we find that the new formula for speedup is simply $1/c$ times the old value. Let us examine a very simple case in detail. If we look at the symmetric, balanced case, the updated formula for speedup is

$$S = \frac{4}{c(2+\sqrt{q})}$$

It is easy to see that as $c \to \infty$ speedup will go to zero. For $c \geq 2$ Time Warp with two processors is always worse than running on one processor without TW. Conversely, for $c \leq 4/3$ TW always wins out. The interesting range is $4/3 \leq c \leq 2$. In this range, certain values of q will yield speedup, while others won't. We are most concerned with the boundary where $S = 1$ which is the transition from areas where TW on two processors helps to where it hurts. Setting $S = 1$ and solving for q we find the necessary condition for two processors running TW to be faster than the single (non-TW) processor.

$$q \leq \frac{4(2-c)^2}{c^2}$$

Figure 8 shows the regions in the $c - q$ plane where TW on two processors is effective and where it is not. Thus, if we know the values of both c and q for our symmetric, balanced system we can immediately tell

whether the application will run faster under Time Warp on two processors. [! ! **INSERT Figure 8 HERE** ! !]

8 Conclusions

In this paper we have created a model for two processor Time Warp execution and provided the results of its exact solution. The model is general enough to subsume the work of Lavenberg, Muntz and Samadi [9] and to partially subsume the work of Mitra and Mitrani [10]. Further, we examined a simplified version of our model and showed for optimal performance that the processors should send as few messages as possible. Further, q (the interaction parameter) has a large effect on speedup for when the load is balanced and speedup changes rapidly when q is near zero. Tasks should be placed on processors such that the average "independent" rate of progress in virtual time is the same for both processors to achieve good speedup. Ideally we want this to be true while also having each processor process events at the same rate ($\lambda_1 = \lambda_2$). Finally, we addressed the cost of state saving by using a very simple extension to the model, and examined its effect on performance. Small state saving costs or infrequent message interactions indicate that TW is effective in gaining speedup.

9 Extensions and Future Work

There are several avenues to follow for future work. One is to add message queueing to our model. Currently any message that arrives in the future is ignored. There are many simulation models where the messages actually carry some work. Another addition would be to charge some cost for rollback. In the present model, rollbacks are free and therefore, there is no penalty for speculative computing. We have exact solutions for models that address these concerns and they will appear in a future work [17]. The most important area to address is the extension of the model to accommodate more than two processors. Certainly an exact Markov chain analysis, which uses the difference in virtual time between the processors as its state variable, will quickly become intractable. We have therefore opted for bounds and approximations, this work is reported in [21].

10 Acknowledgements

We would like to thank the anonomous reviewer who took the time to carefully check our equations, thus discovering several errors.

References

[1] David R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

[2] David M. Nicol. Parallel self-initiating discrete-event simulations. *Transactions on Modelling and Computer Simulation*, 1(1):24–50, January 1991.

[3] D.B. Wagner and E.D. Lazowska. Parallel simulation of queueing networks: Limitations and potentials. In *Proceedings of 1989 ACM SIGMETRICS and PERFORMANCE '89*, volume 17,1, pages 146–155, May 1989.

[4] Jayadev Misra. Distributed discrete-event simulation. *Computing Surveys*, 18(1):39–65, March 1986.

[5] J. Kent Peacock, J.W. Wong, and Eric G. Manning. Distributed simulation using a network of processors. *Computer Networks*, 3(1):44–56, 1979.

[6] Yi-Bing Lin and Edward D. Lazowska. Reducing the state saving overhead for time warp parallel simulation. Technical Report 90-02-03, Department of Computer Science and Engineering, University of Washington, February 1990.

[7] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–564, July 1978.

[8] Steven Bellenot. Global virtual time algorithms. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22,1, pages 122–127. Society for Computer Simulation, January 1990.

[9] Steven Lavenberg, Richard Muntz, and Behrokh Samadi. Performance analysis of a rollback method for distributed simulation. In *Performance '83*, pages 117–132. North-Holland, 1983.

[10] Debasis Mitra and I. Mitrani. Analysis and optimum performance of two message-passing parallel processors synchronized by rollback. In *Performance '84*, pages 35–50. North-Holland, 1984.

[11] Vijay Madisetti, Jean Walrand, and David Messerschmitt. Synchronization in message-passing computers: Models, algorithms and analysis. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22,1, pages 35–48. Society for Computer Simulation, January 1990.

[12] Vijay Krishna Madisetti. Self synchronizing concurrent computing systems. Technical Report UCB/ERL M89/122, Electronics Research Laboratory, College of Engineering University of California Berkeley, CA 94720, October 1989.

[13] Yi-Bing Lin and Edward D. Lazowska. Optimality considerations for "time warp" parallel simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22,1, pages 29–34. Society for Computer Simulation, January 1990.

[14] Yi-Bing Lin and Edward D. Lazowska. A study of time warp rollback mechanisms. *ACM Transaction on Modeling and Computer Simulation*, 1(1):51–72, January 1991.

[15] Yi-Bing Lin and Edward D. Lazowska. Effect of process scheduling in parallel simulation. *International Journal in Computer Simulation*, 2(1):107–121, 1992.

[16] Robert E. Felderman and Leonard Kleinrock. Two processor time warp analysis: Some results on a unifying approach. In *Proceedings of the 5th Workshop on Parallel and Distributed Simulation(PADS91)*, volume 23,1, pages 3–10. Society for Computer Simulation, January 1991.

[17] Robert E. Felderman and Leonard Kleinrock. Two processor time warp analysis: Capturing the effects of message queueing and rollback/state saving costs. Technical Report 920035, University of California, Los Angeles, Computer Science Department, June 1992. Submitted to *ACM Transactions on Modelling and Computer Simulation (Nov 1990)*.

[18] B. Lubachevsky, A. Shwartz, and A. Weiss. Rollback sometimes works... if filtered. In *Proceedings of the 1989 Winter Simulation Conference*, pages 630–639, December 1989.

[19] Robert E. Felderman. Performance analysis of distributed processing synchronization algorithms. Technical Report 910019, University of California, Los Angeles, Computer Science Department, June 1991. Ph.D. Dissertation.

[20] Leonard Kleinrock. On distributed systems performance. In *Proceedings of the 7th ITC Specialist Seminar, Adelaide, Australia.* ITC, September 1989. (Also published in "Computer Networks and ISDN Systems" vol. 20, no.1-5, pp. 206-215, December 1990.).

[21] Robert E. Felderman and Leonard Kleinrock. Bounds and approximations for self-initiating distributed simulation without lookahead. Submitted to *ACM Transactions on Modelling and Computer Simulation*, September 1991.

List of Figures

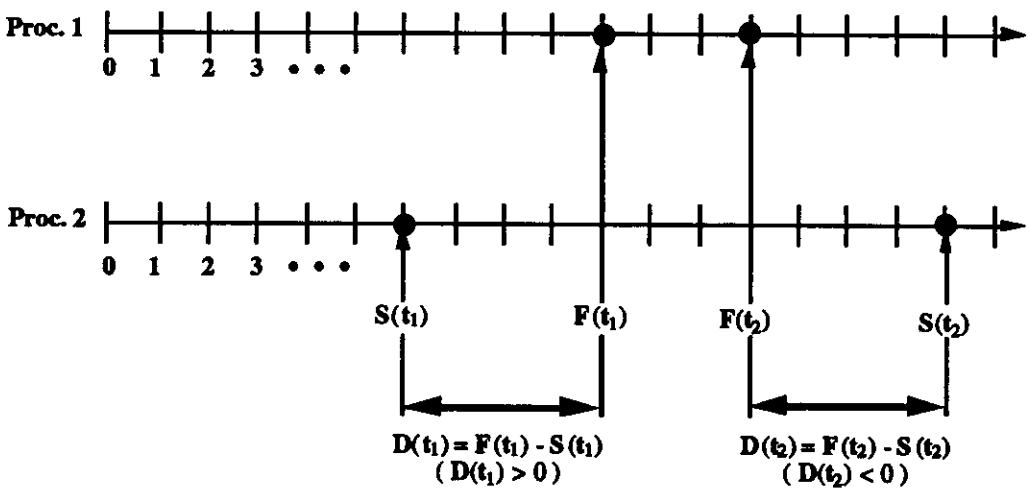

Figure 1: The states of two processors at times t_1 and t_2.

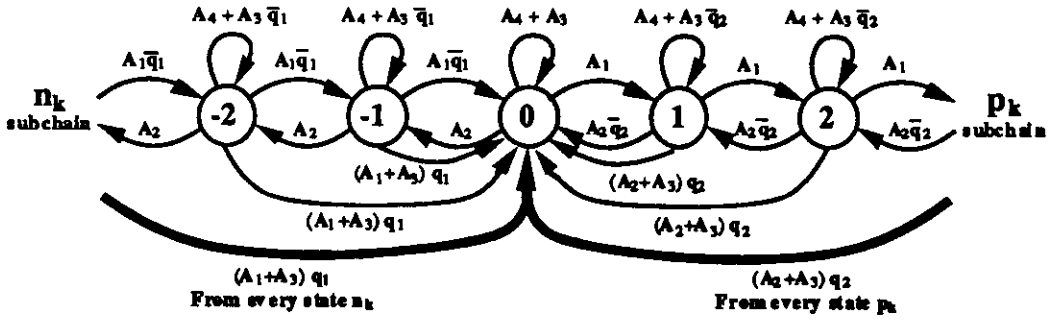

Figure 2: State transition probability diagram for $\beta_1^{\cdot} = \beta_2 = 1$.

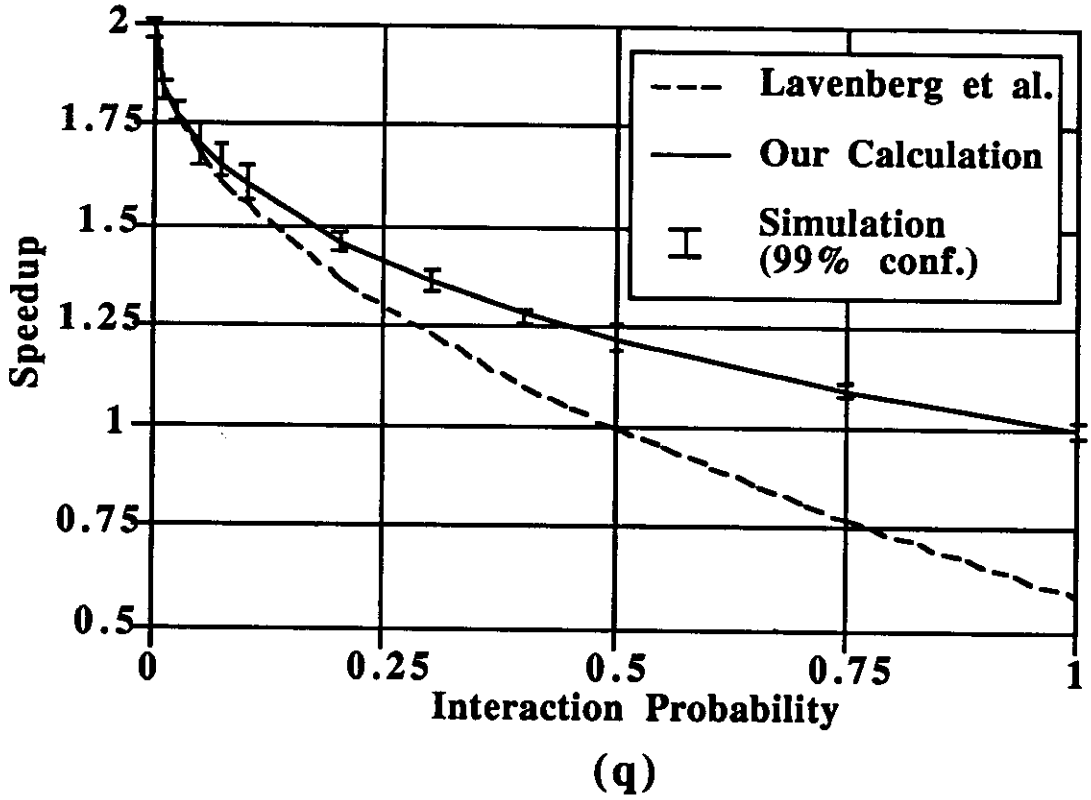

Figure 3: Comparison of speedup results for a simplified case.

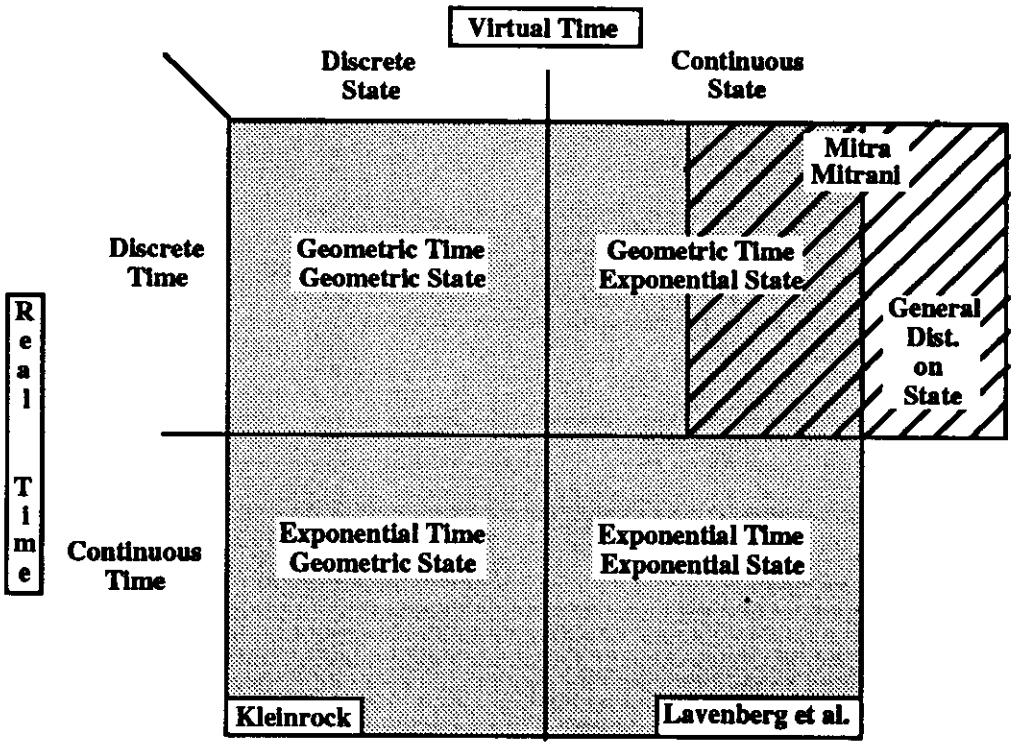

Figure 4: Previous work.

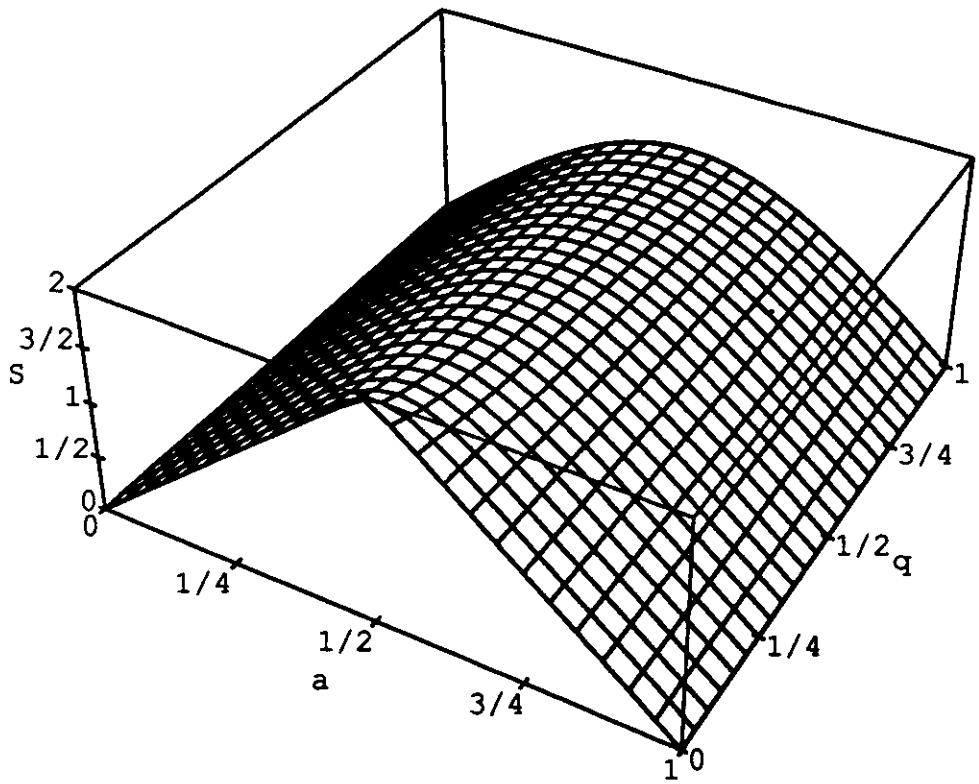

Figure 5: Speedup for the Symmetric case $q_1 = q_2 = q$

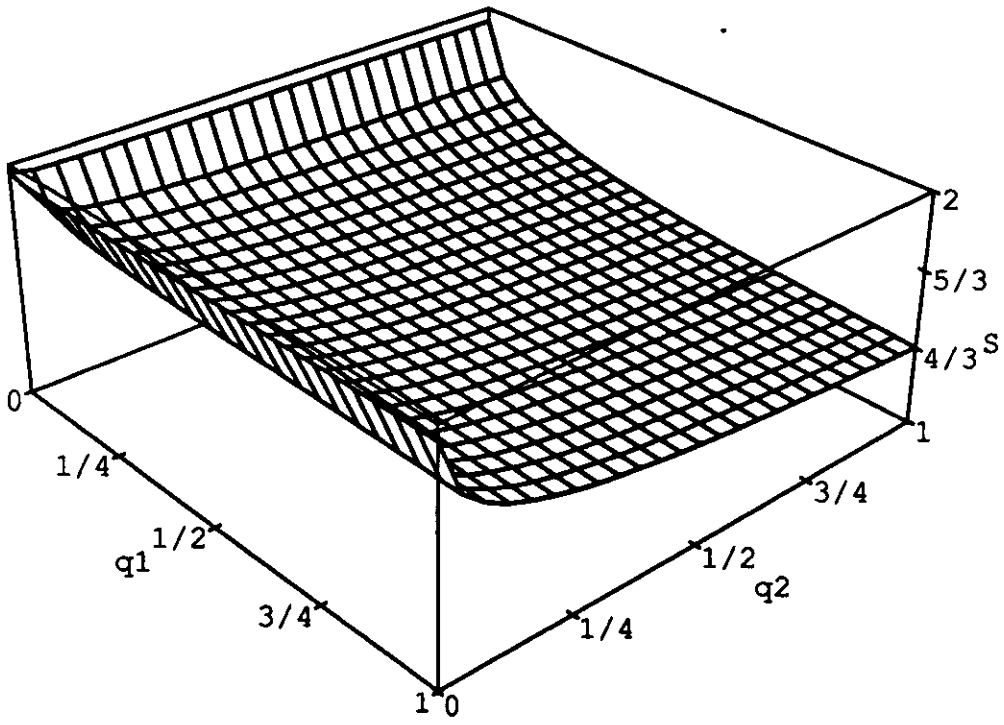

Figure 6: Speedup for the Balanced case $\lambda_1 = \lambda_2 = \lambda$.

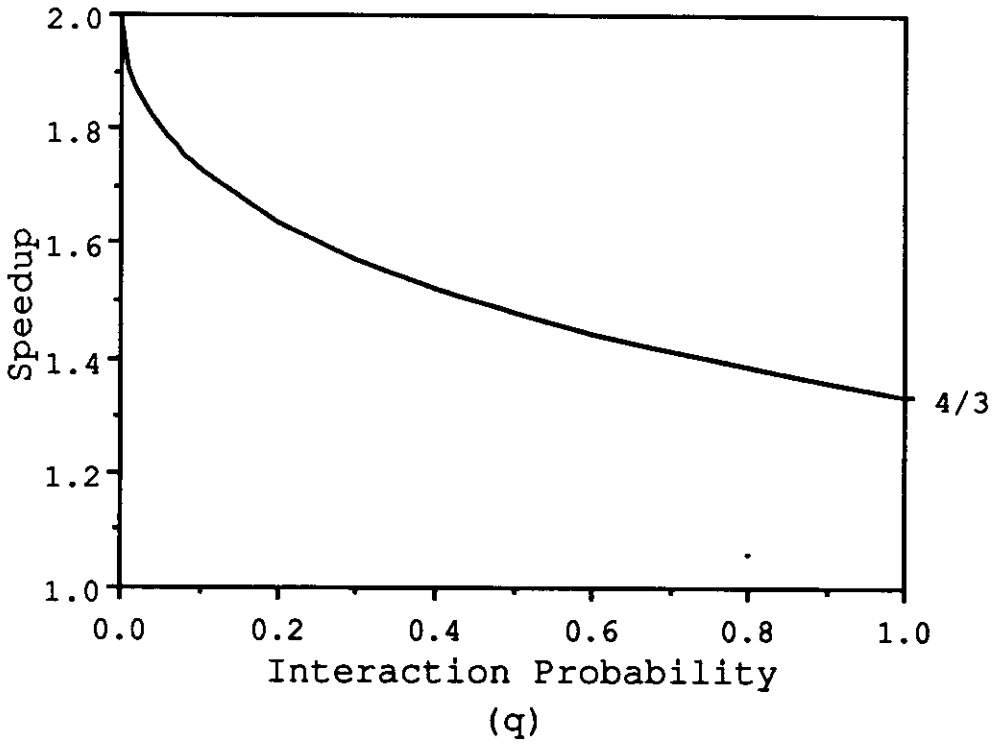

Figure 7: Speedup for the Symmetric, Balanced case $q_1 = q_2 = q$ and $\lambda_1 = \lambda_2 = \lambda$.

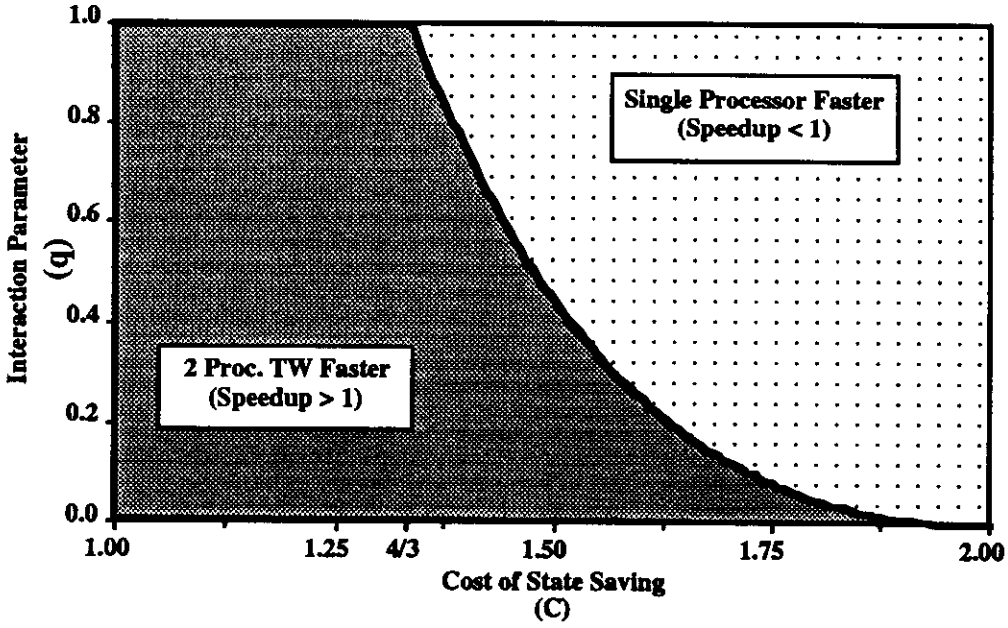

Figure 8: Cost for state saving and its effect on performance.