Batch Based Cancellation: A Rollback Optimal Cancellation Scheme in Time Warp Simulations

Yi Zeng Wentong Cai Stephen J Turner
School of Computer Engineering
Nanyang Technological University
Singapore 639798
{pg00847560, aswtcai, assjturner}@ntu.edu.sg

Abstract

An efficient cancellation scheme is essential to the performance of Time Warp simulations. The pitfalls of rollback echoes, chasing hazards and cascading rollbacks can be identified as being attributable to the inefficiency of the conventional per-event based cancellation scheme. Instead of capturing the happen-before relation between events, which is used by the range based cancellation scheme, the batch based cancellation scheme proposed in this paper utilizes a modified paradigm of vector time, namely, state vector, to capture the dependence of events. We prove that with conformance to specific rules regulating the advancement of LPs*, the events to be cancelled by a straggler message can be determined using a range of the state vector. Thus, knowledge of the range enables any* LP *to recover from the receipt of a straggler message at the cost of at most one rollback (i.e., rollback optimal). The results of preliminary experiments conducted using a manufacturing model show that the proposed scheme is successful in reducing the number of anti-messages and increasing the ratio of the number of committed events to the number of processed events.*

1 Introduction

In Time Warp simulations, the dependence of events can be expressed using state dependence or scheduling dependence or a transitive closure of both [4]. For any two processed but not yet rolled back events at the same *LP*, the later event is defined to be state dependent on the earlier event due to the order of their accesses to the *LP*'s state variables; for an event scheduling another event, the latter is defined to be scheduling dependent on the former. To remove the wrong computations based on an event when it is rolled back, all events that depend on it have to be cancelled [1].

The conventional way of cancelling events is solely carried out on a per-event basis. When an *LP* decides to cancel an external event which was previously scheduled by an outgoing positive message, it proceeds by constructing its counterpart, namely, an anti-message, and sending it to the same destinations. Upon receipt of the positive message and its anti-message, the receiving *LP* cancels the scheduled event and performs necessary rollbacks if that event has been processed.

The above cancellation scheme empirically proves to be simple and feasible in Time Warp simulations. But it still leaves performance concerns behind and has drawn researchers' attention. Given that *LPs* proceed aggressively in a Time Warp simulation, during the period between processing an incoming positive message and receiving its anti-message, a certain amount of events may have been scheduled by an *LP*. Without loss of generality, these events are dependent on the cancelled event, hence need to be rolled back by means of sending further anti-messages. For the cancellations triggered by a straggler message, it is apparent that the cost is mainly dominated by the way *LPs* process and cancel events. In the presence of dynamic CPU workload and varying channel speed in a parallel or a distributed simulation, the cost is hardly predictable.

Several pitfalls in Time Warp simulations which can be attributable to the cost of cancellations have been figured out by researchers. Rollback echoes and chasing hazards have been identified [4] in particular circumstances which could severely impact or even destroy the simulation. Multiple rollbacks, an inherent performance pitfall of the conventional cancellation scheme, has been investigated in [2], which argues that anti-messages may result in unwanted multiple rollbacks at a single *LP* and trigger a cascading rollback situation,

[1] In this paper, "cancel" means not only a rollback of a processed event but also a cancellation of an unprocessed event.

Proceedings of the 18th Workshop on Parallel and Distributed Simulation (PADS'04)

where the rollback cycles back to the original LP where the straggler message was received.

There are many approaches that have been proposed to reduce the cost of cancellations. As opposed to aggressive cancellation, lazy cancellation [5] reduces the number of anti-messages by delaying the sending of anti-messages till reprocessing of the event produces a different positive message. Unfortunately, the performance comparison shows that lazy cancellation can arbitrarily outperform aggressive cancellation and vice versa [11]. Lazy re-evaluation [14] lets an LP directly jump back to the state prior to when the rollback occurred if the straggler message does not affect its state. To avoid chasing hazards, wolf calls [9] expedite the propagation of the knowledge of cancellations by sending protocol messages with higher-priority over positive messages in their transmission. However, all these approaches do not address the inefficiency of the conventional per-event based cancellation scheme itself.

Chetlur and Wilsey addressed this issue and proposed the range based cancellation scheme [2] by taking advantage of vector time [12, 10], a well-understood mechanism to capture the happen-before relation (denoted as $\rightarrow$) [7] between events in distributed systems. A new type of message, namely, *CANCEL_MESSAGE*, was introduced as a replacement of anti-messages in Time Warp simulations. By exploiting the happen-before relation among messages, the range information carried by a single *CANCEL_MESSAGE* is capable of cancelling multiple events at an LP, thus the number of messages sent among LPs can be reduced and many unwanted rollbacks can be avoided.

However, further study reveals that Chetlur and Wilsey's approach cannot guarantee the removal of all unwanted rollbacks. In other words, some LPs still suffer from multiple rollbacks to cancel events incurred by a straggler message (so the scheme is not rollback optimal). In this paper, the batch based cancellation scheme is proposed to address this problem.

The remainder of this paper is organized as follows. Section 2 presents several concerns in Chetlur and Wilsey's approach which motivate our further investigation. Section 3 presents the basic mechanism, state vector, which is employed to capture the dependence of events in Time Warp simulations. Based on this machanism, Section 4 gives supporting theories which prove the possibility of a rollback optimal cancellation scheme. This results in our implementation, the batch based cancellation scheme, which is elaborated in Section 5. Section 6 presents the experimental results obtained from the comparison of the conventional cancellation scheme and our batch based cancellation scheme running on a manufacturing model [8]. Section 7 concludes the paper.

2 Motivation

In discrete event simulations (DES), the delivery order of scheduled events at an LP is mandatorily determined by their simulation time, a virtual representation of the real time in the physical system being modelled. This rule is also referred to as the local causality constraint (LCC). LCC is prone to being violated in Time Warp simulations because the optimistic advancement of LPs gives rise to the possibility of receiving a straggler message, a positive message scheduling a "past" event, at an LP. Once this happens, LPs have to cancel wrong computations accordingly. Figure 1 shows a common cancellation scenario found in Time Warp simulations, where the events are denoted as black dots and scheduling of the events through positive messages and cancellation of the events through anti-messages are represented as solid arrows and dashed arrows respectively. In addition, the straggler message is labelled with S.

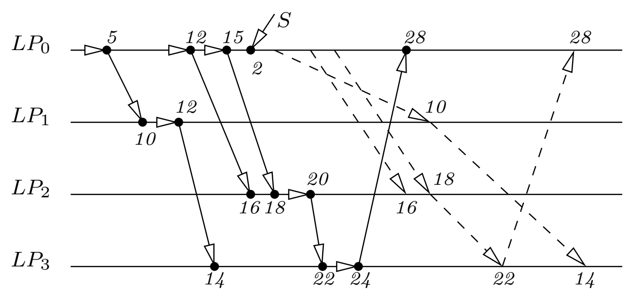

Figure 1. Per-event based cancellations triggered by a straggler message

The performance concern of the conventional scheme can be explained using Figure 1. (1) When LP_0 receives the anti-message for event $e_{0,28}$[2], LP_0 may have already advanced to a certain time in the future and scheduled considerable events. Further efforts are therefore needed to cancel these wrong computations. (2) Because the anti-messages for events $e_{3,14}$ and $e_{3,22}$ are sent from different LPs, LP_3 may suffer from more rollbacks if the anti-message for $e_{3,14}$ arrives later.

The mechanism of Chetlur and Wilsey's approach is shown in Figure 2, where the dashed arrows represent the *CANCEL_MESSAGEs*. The combination of simulation time (the first component) and vector time (the remaining N components) forms the Total Clock, where N is the number of LPs in the simulation ($N = 4$ here). Total Clock is always sent along

[2] To simplify the representation, $e_{i,t}$ denotes the event scheduled at LP_i with time-stamp t.

with messages (not shown in the figure). The updating rules of simulation time and vector time are independent from each other and correspond to those defined in [2, 6, 7]. Each *CANCEL_MESSAGE* carries a range, denoted by $[a, b]_i$, which indicates that the set of events that have been cancelled at LP_i are those whose i^{th} component of vector time lies in between a and b. Based on a received range, the receiving LP cancels events by checking their vector time. A new range for the cancelled events is then deduced and sent out via new *CANCEL_MESSAGEs* (if the range is not empty).

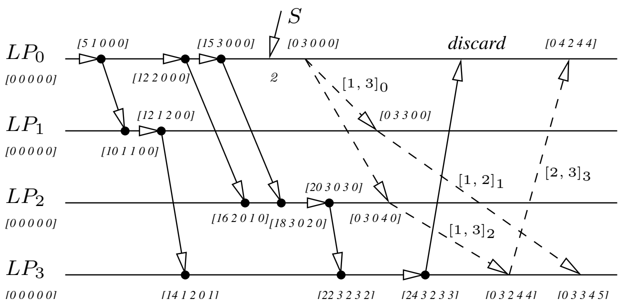

Figure 2. The range based cancellations triggered by a straggler message

Comparing Figure 2 with Figure 1, it can be seen that Chetlur and Wilsey's approach has several advantages. (1) A *CANCEL_MESSAGE* is capable of cancelling multiple events. Upon receipt of range $[1, 3]_0$, LP_2 is able to cancel events $e_{2,16}$, $e_{2,18}$ and $e_{2,20}$. This results in a new range $[1, 3]_2$ being sent to LP_3. (2) Because the received ranges are recorded at LPs, an LP has intelligence to discard some incoming messages which are to be eventually cancelled so as to prevent wrong computations from being further propagated. For example, LP_3 schedules an event at LP_0 at simulation time 24. Because event $e_{0,15} \to e_{3,24}$ and $e_{0,15}$ has already been cancelled, the event scheduled by $e_{3,24}$ is simply discarded based on the range $[1, 3]_0$ recorded at LP_0.

Regarding the number of rollbacks from which an LP may suffer to thoroughly remove the wrong computations incurred by a certain straggler message, we define a cancellation scheme to be rollback optimal if any affected LP is guaranteed to be recovered at the cost of at most one rollback. Chetlur and Wilsey's approach has reduced rollbacks to some extent, however, it is not rollback optimal (in Figure 2, LP_3 rolls back twice). A rollback optimal cancellation scheme requires some kind of mechanism to identify all the events which are destined to be cancelled upon receipt of a straggler message.

3 State Vector

The happen-before relation proves to be the most basic but universal partial order relation in distributed systems and vector time proves to be the simplest means to characterize it [1, 12]. However, vector time (as shown in [2]) is not an ideal candidate to capture dependence of events. This is illustrated in Figure 3. LP_i schedules e_{j,t_2} and later on cancels it due to the rollback caused by a straggler message. For events e' and e'' scheduled by LP_j, where e' is scheduled after e_{j,t_2} but before the cancellation and e'' is scheduled after the cancellation, both $e_{i,t_1} \to e'$ and $e_{i,t_1} \to e''$ hold. But, in fact, e'' does not depend on e_{i,t_1}.

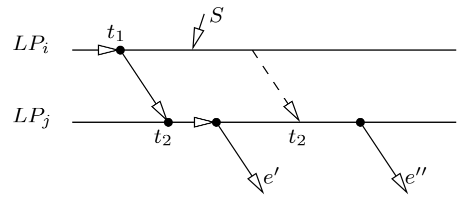

Figure 3. Happen-before relation between events

Suppose that every LP changes its state whenever it processes an event. A monotonically increasing scalar, namely, state counter, is introduced into each LP to uniquely identify its state. A vector of state counters, namely, state vector, is constructed at each LP as well. For LP_i's state vector, denoted as $SV(LP_i)$, $SV(LP_i)[j]$, $j \neq i$ reflects LP_i's current knowledge of LP_j's latest state counter, while $SV(LP_i)[i]$ is just the placeholder of its own state counter.

The propagating and updating rules of LP_i's state vector are shown in Figure 4. It can be seen that the last rule makes state vector different from vector time.

With the rules in Figure 4, $SV(LP_i)[i]$ uniquely identifies an event processed at LP_i. Given an event e processed by an LP with state vector $SV(e)$, the latest event in LP_i on which e depends can, therefore, be determined by $SV(e)[i]$ (formally stated in Theorem 4.4). Looking back to Figure 3 again, the link is broken because $SV(e'')[i]$ is restored from an earlier state. $SV(e'')[i] < SV(e_{i,t_1})[i]$ indicates that e'' does not depend on e_{1,t_1}. Other properties of state vector and dependence are exploited in Section 4.

4 Characterization of Cancelled Events

A widely accepted model [12] is adopted and modified in the proof of the following theorems. A Time Warp simulation is viewed as consisting of N sequential LPs, communicating solely by exchanging two

1. Let N be the number of *LPs* in the simulation. Initially, $SV(LP_i)[k] = 0$, $0 \leq i, k < N$.
2. Let $SV_m(e)$ be the state vector of e which is piggybacked on its scheduling message m and let $SV(e)$ be the state vector of e at the time e is processed. When LP_i processes e, these updates are performed: $SV(LP_i)[i]$ first increases by one; then $SV(LP_i) = sup(SV(LP_i), SV_m(e))$, where *sup* performs component-wise maximum operation; then $SV(e) = SV(LP_i)$.
3. At the time LP_i schedules event e, either for itself or for another *LP*, $SV(LP_i)$ is piggybacked on the scheduling message, i.e., $SV_m(e) = SV(LP_i)$.
4. Except for $SV(LP_i)[i]$, other components of SV are state saved. At the time a rollback occurs at LP_i, $SV(LP_i)[i]$ increases by one and other components are restored from the state to which LP_i rolls back.

Figure 4. The propagating and updating rules of state vector at LP_i

kinds of messages, namely, positive messages and *CANCEL_MESSAGEs*. Each *LP* changes its state whenever it processes the event scheduled by a positive message or performs a rollback. The transmission channels are assumed reliable and FIFO.

Let E_i denote the set of events processed at LP_i, and let $E = \bigcup_{i=0}^{N-1} E_i$ denote the set of all events processed in the simulation. As we assume that each LP_i is strictly sequential, we can index the events of LP_i in their processing order: $E_i = \{e_{i1}, e_{i2}, \ldots, e_{in}, \ldots\}$. This processing order is referred to as the *standard enumeration* of E_i. Given E_i, the local successor set of event e_{in}, denoted as $\mathcal{S}_l(e_{in})$, represents all the events locally processed after e_{in}, $\mathcal{S}_l(e_{in}) = \{e_{ik} | k > n\}$.

Theorem 4.1 through Theorem 4.5 state the properties of state vector and the dependence of events. They are obvious by applying the rules shown in Figure 4 and are listed below without proving.

Theorem 4.1. *Let $e \Rightarrow e'$ denote that event e' depends on e. Dependence is transitive, i.e., for three events e, e' and e'', if $e \Rightarrow e'$ and $e' \Rightarrow e''$, it also holds that $e \Rightarrow e''$.*

Theorem 4.2. *$SV(LP_i)[i]$ increases monotonically throughout a simulation.*

Theorem 4.3. *For any event $e_{ik} \in E_i$, $SV(e_{ik})[i] = k$.*

Theorem 4.4. *For event $e' \in E_j$, suppose that $SV(e')[i] = k$, then $e_{ik} \Rightarrow e'$.*

Theorem 4.5. *For events $e_{ik} \in E_i$ and $e' \in E_j$, if $e_{ik} \Rightarrow e'$, then $SV(e')[i] >= k$.*

Observing that receiving straggler messages is the root cause of rollbacks in *LPs*, let m_s denote a straggler message received at LP_o. LP_o is thus named as rollback originator. During the cancellations triggered by m_s (for the ease of discussion, assume there is no other intervening cancellations triggered by other straggler messages), let $C_i(m_s)$ denote the set of events cancelled at LP_i and $C(m_s)$ denote all the events cancelled in the simulation. Thus, $C(m_s) = \bigcup_{i=0}^{N-1} C_i(m_s)$. Note that for $i \neq o$, $C_i(m_s)$ is essentially the set of events that have dependence on the events in $C_o(m_s)$, i.e., $\{e' | e' \in E_i \wedge \exists e \in C_o(m_s), e \Rightarrow e'\}$.

Although $C(m_s)$ can be expressed in terms of $C_o(m_s)$, $C_o(m_s)$ or $C(m_s)$ is still unpredictable without any regulation of the advancement of *LPs*. To be able to identify $C_o(m_s)$, which is expressed as a range (shown in Theorem 4.7), and furthermore, identify $C(m_s)$ using this range (shown in Theorem 4.8), the rules of event processing at LP_i are given in Figure 5. In the first rule, if event e is processed, it would be rolled back unnecessarily because e' will be eventually cancelled. In the second and the third rules, event e can be discarded immediately as it is known to be dependent on a cancelled event.

At the time LP_i processes event e, where $SV_m(e) = \{x_0, x_1, \ldots, x_{N-1}\}$,

1. If a processed (but not yet committed) event e' at LP_i depends on a cancelled event e_{jk}, $j \neq i$, $k \leq x_j$ on which e does not depend, processing of e is blocked until e' has been cancelled;
2. If e_{ix_i} has been cancelled locally, e is discarded without processing.
3. If e depends on a cancelled event e_{jk}, $j \neq i$, $k \leq SV(LP_i)[j]$, e is discarded without processing.

Figure 5. The rules of event processing at LP_i

Theorem 4.6. *Suppose that the rollback originator LP_o receives straggler message m_s and events e_{oa} and e_{ob} are the earliest and latest events among those being cancelled by LP_o. It holds that,*

$$C_o(m_s) = \{e_{oa}\} + \mathcal{S}_l(e_{oa}) - \mathcal{S}_l(e_{ob}).$$

Proof. The event set $\{e_{oa}\} + \mathcal{S}_l(e_{oa}) - \mathcal{S}_l(e_{ob})$ represents all the events cancelled by LP_o upon receipt of straggler message m_s. The second rule in Figure 5 prohibits their effects from further spreading at LP_o, hence this set essentially equals $C_o(m_s)$. □

Theorem 4.7. *The o^{th} component of the state vector of events in $C_o(m_s)$ forms a continuous range $[a,b]_o$, that is,*

1. $\forall k \in [a,b]_o$, $e_{ok} \in C_o(m_s)$;
2. $\forall e_{ok} \in E_o$ *and* $e_{ok} \notin C_o(m_s)$, $k \notin [a,b]_o$.

Proof. From Theorem 4.3, $SV(e_{oa})[o] = a$ and $SV(e_{ob})[o] = b$. According to Theorem 4.2 and Theorem 4.6, the correctness is obvious. □

Theorem 4.7 offers an efficient way to convey the set of events being cancelled at a rollback originator to other *LPs* (recall the similar way in which *CANCEL_MESSAGEs* work in the range based cancellation scheme). As shown in the next section, our proposed scheme also employs *CANCEL_MESSAGEs* to carry these ranges.

Theorem 4.8. *Given range $[a,b]_o$ representing $C_o(m_s)$, $\forall e \in E_i$, $i \neq o$,*
$e \in C_i(m_s)$ *iff* $SV(e)[o] \in [a,b]_o$.

Proof. (1) Necessity. On one hand, for any event $e \in C_i(m_s)$, it holds that $\exists k \in [a,b]_o$, $e_{ok} \Rightarrow e$. Applying Theorem 4.5, $SV(e)[o] \geq k \geq a$. On the other hand, it must hold that $SV(e)[o] \leq b$. Otherwise, assume $SV(e)[o] = k' > b$, then $e_{ok'} \Rightarrow e$ according to Theorem 4.4. $e_{ok'}$ does not depend on any event in $C_o(m_s)$, or it should have been discarded according to the second rule in Figure 5. Therefore, e must have dependence on other events as shown in Figure 6, where dependence of events is denoted as curved arrows. There must be two events e'' and e' which were processed in this order at LP_i, $i \neq o$: e depends on e'' and e'' depends on an event $e_{ok''}$ in $C_o(m_s)$ as $e \in C_i(m_s)$; e is identical to or depends on e' and e' depends on $e_{ok'}$ as the knowledge of k' was propagated to e; e'' was processed before e' (otherwise, e'' was discarded by LP_i according to the third rule in Figure 5). This violates the first rule in Figure 5 at the time LP_i processed e' because processing of e' should have been blocked until e'' was cancelled. (2) Sufficiency. Let $SV(e)[o] = k$, $k \in [a,b]_o$. Applying Theorem 4.4, $e_{ok} \Rightarrow e$. Because $e_{ok} \in C_o(m_s)$, $e \in C_i(m_s)$. □

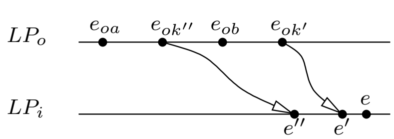

Figure 6. The dependence of event e

The rules in Figure 5 and Theorem 4.6 through Theorem 4.8 form the basis for a rollback optimal cancellation scheme. The essential difference from the range based scheme is that instead of continuously constructing ranges along the propagation path of *CANCEL_MESSAGE*s, which could gradually limit their abilities to determine events in $C_i(m_s)$, $i \neq o$ (see ranges $[1,3]_0$ and $[1,3]_2$ in Figure 2), a rollback optimal cancellation scheme comes to the conclusion that LP_i is able to directly deduce what $C_i(m_s)$ is like based on the received $[a,b]_o$ from the rollback originator LP_o. For any processed events in $C_i(m_s)$, LP_i performs a single rollback to cancel all of them; For any received but not yet processed events or any events arriving in future, LP_i simply discards them if they are identified to be in $C_i(m_s)$. This ensures that any LP will discard all those events to be cancelled by a straggler message in a batch manner and at the cost of at most one rollback (rollback optimal).

5 Batch based Cancellation Scheme

5.1 Rollback History

From Section 4, it can be seen that the rules in Figure 5 are essential for the correctness of the scheme. Empirically, these rules can be fulfilled by employing the concept of rollback history. We have shown in Theorem 4.7 and Theorem 4.8 that range $[a,b]_o$ identifies all the events in $C(m_s)$. The rollback history of an LP, denoted as $RH(LP)$, is actually defined as a list of those ranges that were locally generated or learnt from other *LPs* through message passing.

Piggybacking its current rollback history on any event scheduled by an LP makes it possible for other *LPs* to process the event in the way that satisfies the first rule in Figure 5. Particularly, at the time an LP processes event e, it first looks into those events that have been processed but not yet committed. If an event that is to be cancelled by a certain range in the rollback history carried by e is found, processing of e is blocked until that event is cancelled.

The greatest challenge to the above approach is the increasing size of rollback histories carried by events, which could incur significant communication cost and storage cost, and render the approach itself empirically useless. Thanks to the FIFO property of communication channels, communication cost can be reduced by sending rollback histories incrementally and reconstructing them at the receiving *LPs*. To do this, a similar technique to that of compressing vector time described in [13] is employed together with more sophisticated data structures.

LP_i locally maintains a rollback history table, denoted as $RHT(LP_i)$. For $0 \leq j < N$, $j \neq i$, $RHT(LP_i)[j]$ records LP_i's latest knowledge of

Proceedings of the 18th Workshop on Parallel and Distributed Simulation (PADS'04)
1087-4097/04 $20.00 © 2004 **IEEE**

$RH(LP_j)$. $RHT(LP_i)[i]$ is the placeholder of LP_i, but in addition, each range inside is tagged with a scalar, and denoted as $[a,b]_o^s$, where s was the current value of LP_i's state counter when the range was recorded. LP_i keeps a "last scheduling" vector, denoted as $LS(LP_i)$, as well. $LS(LP_i)[j]$, $0 \leq j < N$, $j \neq i$, tracks the latest value of LP_i's state counter when an event was scheduled at LP_j.

Let $\Delta RH(e)$ denote the increment of the sender LP's rollback history piggybacked on e, and let $RH(e)$ denote its reconstructed rollback history at the receiving LP. The updating rules of $RHT(LP_i)$ are shown in Figure 7.

1. Initially, $RHT(LP_i)[j]$, $0 \leq j < N$, is set empty, and $LS(LP_i)[j] = 0$, $0 \leq j < N$.
2. At the time LP_i receives event e from LP_j, $\Delta RH(e)$ is appended to the end of $RHT(LP_i)[j]$ and then emptied. The reconstructed $RH(e)$ is the current value of $RHT(LP_i)[j]$, represented by a pointer to the current end of the latter.
3. At the time LP_i processes an event e from LP_j, $\forall [a,b]_o \in RH(e)$, $[a,b]_o^{SV(LP_i)[i]}$ is merged (without any duplication) into $RHT(LP_i)[i]$, and then $[a,b]_o$ is removed from $RHT(LP_i)[j]$.
4. At the time LP_i schedules an event e to LP_j, $\Delta RH(e)$ is first set to the list of ranges in $\{[a,b]_o | [a,b]_o^s \in RHT(LP_i)[i] \wedge s > LS(LP_i)[j]\}$. $LS(LP_i)[j]$ is then updated with the current $SV(LP_i)[i]$.

Figure 7. Updating rollback history table in FIFO environments

Note that in the third rule, removal of ranges from $RHT(LP_i)[j]$ means that it is not an exact reconstructed $RH(e)$ in the strict sense. However, its correctness is obvious.

The rollback history table is also subject to fossil collection. After a GVT calculation cycle, let x_j, $0 \leq j < N$, be the maximum state counter fossil collected at LP_j. Because it holds that there are no events (including transient events) dependent on any cancelled event e_{jk}, $k < x_j$, $\forall [a,b]_j \in RHT(LP_i)$, $[a,b]_j$ is discarded if $b \leq x_j$.

5.2 Routines

An LP is assumed to have a similar architecture to that depicted in [3], which routinely maintains three queues, namely, the input queue (IQ), the output queue (OQ) and the state queue (SQ). Because *CANCEL_MESSAGEs* are processed at the time they are received and will not be rolled back for reprocessing, an IQ only queues positive messages. Let E_{IQ_i} denote the currently buffered processed events, i.e., neither having been rolled back nor fossil collected, at IQ_i. It is obvious that $E_{IQ_i} \subseteq E_i$ and empirically only events in $E_{IQ_i} \cap C_i(m_s)$ are considered to be rolled back by the straggler message m_s.

Let $ST(e)$ and $ST(LP_i)$ denote the simulation time of event e and LP_i respectively, and let $m.e$ denote the event scheduled by message m. The major routines of the batch based cancellation scheme are presented in Figure 8 through Figure 11.

```
PROCESS(m) {
  /* Rule 1 in Figure 5 */
  IF (∀[a,b]o ∈RH(m.e),∄e' ∈ E_IQi,SV(e')[o]∈[a,b]) {
    ST(LPi)=ST(m.e);
    SV(LPi)[i]++;
    SV(LPi)=sup(SV(LPi),SVm(m.e));
    /* Rule 3 in Figure 7 */
    merge RH(m.e) into RHT(LPi)[i];
    execute m.e;
  }
}
```

Figure 8. Processing a scheduled event at LP_i

```
SCHEDULE(e) {
  construct message m scheduling e;
  set ST(m.e);
  SVm(m.e)=SV(LPi);
  /* Rule 4 in Figure 7 */
  set ΔRH(m.e);
  queue m into OQi and send m to its destinations;
}
```

Figure 9. Scheduling an event at LP_i

Figure 8 shows the steps for LP_i to process a positive message from IQ_i. LP_i first checks if it is safe to proceed. If yes, LP_i continues with updating its simulation time, state vector and rollback history table, and then executing the scheduled event, which involves saving LP_i's current state in SQ_i, updating the current state and probable scheduling of new events (Figure 9). Note that since possible receipt of straggler messages has been detected and handled at an earlier stage (see Figure 10), the messages to be delivered here can be processed without further checking for causality violations.

Figure 10 demonstrates the handling of a message when it is received from the communication channel. Basically there are four different ways according to the criteria the message meets. (1) Message m is a *CANCEL_MESSAGE* carrying a range. LP_i first checks if any processed event falls within the range. If there is,

```
RECEIVE(m) { /* m from LP_j */
  IF (m is a CANCEL_MESSAGE) {
    IF (∃ min k, e_ik ∈ E_IQi ∧ e_ik in m's range) {
      ROLLBACK(e_ik, m);
    }
    merge m's range into RHT(LP_i)[i];
    ∀m′ ∈ IQ_i, discard m′ if m′ in m's range;
  } ELSE {
    /* Rule 2 in Figure 7 */
    reconstruct RH(m.e);
    IF (∃[a,b]^s_o ∈RHT(LP_i)[i], SV(m.e)[o]∈[a,b]) {
      /* Rule 2 and Rule 3 in Figure 5 */
      discard m;
    } ELSE IF (m is a straggler message) {
      find earliest e_ik to be rolled back by m;
      construct a new range [k,SV(LP_i)[i]]_i;
      construct CANCEL_MESSAGE m″ carrying the range;
      ROLLBACK(e_ik, m″);
      merge the range into RHT(LP_i)[i];
      LS(LP_i)[i]=SV(LP_i)[i];
      ∀m′ ∈ IQ_i, discard m′ if m′ in the range;
      insert m into IQ_i;
    } ELSE {
      insert m into IQ_i;
    }
  }
}
```

Figure 10. Receiving a message from communication channels at LP_i

LP_i performs a rollback action (see Figure 11). Then LP_i merges this range into its rollback history to identify any events for cancellation arriving in the future. Finally, LP_i looks into its IQ_i to discard those events that have been received but are now identified to be cancelled. (2) Message m is a positive message but identified as to be cancelled by a received range. It means that LP_i is already free from the rolled back state on which m depends. Thus, m is simply discarded. (3) Message m is a straggler message, so LP_i becomes a rollback originator. Except for the need to create a new *CANCEL_MESSAGE*, similar actions to

```
ROLLBACK(e, m) {
  SV(LP_i)[i]++;
  restore the proper state from SQ_i;
  unprocess messages being rolled back in IQ_i;
  IF (cancelled messages in OQ_i) {
    send m to their destinations;
  }
}
```

Figure 11. Rollback to the state prior to executing e at LP_i

those in (1) are carried out. Finally, message m is inserted into IQ_i and expected to be the next message to be processed. (4) Message m is a positive message identified as neither to be discarded nor a straggler. It is simply inserted into IQ_i for future processing.

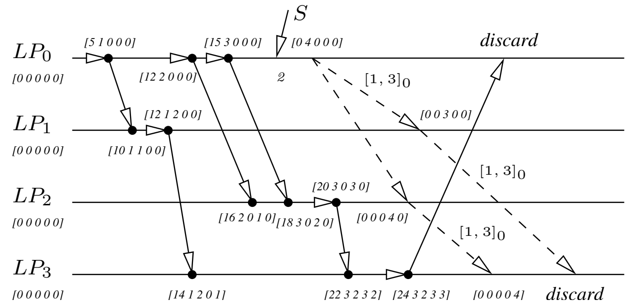

Figure 12. The batch based cancellations triggered by a straggler message

Figure 12 shows a possible scenario applying the batch based cancellation scheme to the scenario in Figure 1. Upon receipt of the straggler message at LP_0, the set of events to be cancelled at LP_0 is determined as $C_0(m_s) = \{e_{01}, e_{02}, e_{03}\}$ (or $\{e_{0,5}, e_{0,12}, e_{0,15}\}$). Range $[1,3]_0$ is sent along with a multicast *CANCEL_MESSAGE* to LP_1 and LP_2. Upon receipt of the range, $C_1(m_s) = \{e_{11}, e_{12}\}$ (or $\{e_{1,10}, e_{1,12}\}$) and $C_2(m_s) = \{e_{21}, e_{22}, e_{23}\}$ (or $\{e_{2,16}, e_{2,18}, e_{2,20}\}$) are determined respectively and the range is further forwarded to LP_3 independently by LP_1 and LP_2. The earlier range received by LP_3 triggers the cancellation of $C_3(m_s) = \{e_{31}, e_{32}, e_{33}\}$ (or $\{e_{3,14}, e_{3,22}, e_{3,24}\}$). The later received one is found to be a duplicate and is discarded.

6 Experiments

Our experiments were carried out on a Dell 2650 Server, a cluster of 11 nodes (dual 2.6GHz Xeon CPUs, 1GB RAM) interconnected through myrinet and running MPI-GM.

A generic manufacturing model [8] was used to evaluate the proposed scheme. According to the model, a manufacturing process is viewed as a sequence of production, assembling and testing stages. The production stage consists of a number of parallel production lines, each producing a different quality-guaranteed component through a sequence of processing stations (PS) and control stations (CS) (See Figure 13). Note that PS and CS are always paired. Once a flaw is found by a CS, having been introduced by its previous PS, the component is sent for reprocessing immediately. Each component is also tested at the end of its production line. In case of malfunction, the defective component is wholly reworked. Figure 14 shows the procedure of the assembling and testing stages, in which constituent components collected from the production lines are assembled to a product at any assembling station (AS) and tested at one of the testing stations (TS). For the sake of clarity, three different kinds of connectors are used for the interconnection. A forker (F) routes an input to several output links with configurable probability. A merger (M) generates an output when it receives inputs from all its input links. A collector (C), however, generates an output on any input. Note that connectors do not increase simulation time.

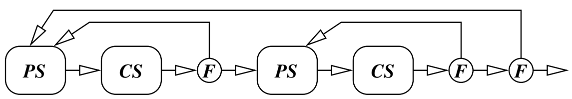

Figure 13. Production line

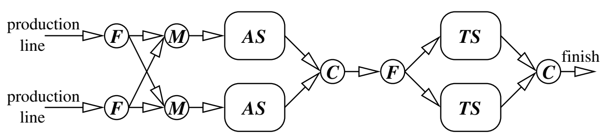

Figure 14. The assembling and testing stages

Without loss of generality, our running model is configured with a moderate number of *LPs*. The production stage is configured with two production lines and each line has one pair of PS and CS installed. There are two ASs and two TSs at the assembling and testing stages. All the units (eight stations plus 11 connectors), i.e., 19 *LPs*, are one-to-one mapped on 19 CPUs.

Two sets of experiments were carried out with different settings of event granularity and each set reports the comparison between the conventional per-event based cancellation scheme and the batch based cancellation scheme. In each run, 10000 components are fed to each production line. Table 1 and Table 2 show the measurements obtained using fine and coarse event granularity respectively. Note in each case the number of events and messages are the sums of those collected from all the *LPs*.

Studying the measurements reported from the experiments, the properties of the batch based cancellation scheme can be better understood. (1) Due to the ability to cancel multiple messages, the reduction of the number of anti-messages was significant. (2) In the batch based scheme, a smaller total number of processed events and straggler messages were always achieved. This can be explained by the rules in Figure 5 and Theorem 4.8. Once a potential wrong computation is identified by a range at an *LP*, it is always prevented from happening or being further propagated. (3) In order to evaluate the effectiveness of the batch based cancellation scheme, the concept of efficiency, which is defined as the ratio of the number of committed events and the total number of processed events, is introduced. Although both schemes committed similar amount of events, which meant similar behavior in the simulations, counting the total number of processed events, the batch based scheme had better results. (4) Based on the execution time obtained from both schemes, the speedups of the batch based scheme compared to the per-event based scheme were calculated. It can be seen that the batch based scheme has only slight performance gain over the per-event based scheme. This is understandable when we look into the running model more closely. The manufacturing model lacks the necessary long range forward links to expedite propagation of ranges. Due to reworking of components in the production stage, PSs have higher probabilities of receiving straggler messages. Once such a straggler message is received by a PS and the range identifying cancelled events is generated, the range has to follow a unique propagation path defined by the model and subsequent units have no means of early detection of potential wrong computations. Therefore in this case, units in the assembling and testing stages still suffer from excessive rollbacks. However, the batch based cancellation scheme may be modified to enable the detection of potential wrong computations at an earlier time, this is addressed in Section 7.

7 Conclusions

In this paper, a rollback optimal cancellation scheme, the batch based scheme, is presented. Rollback optimal means that any *LP* is able to recover from the receipt of a straggler message at the cost of at most one rollback. To do this, a state vector is used to capture the dependence of events and a rollback history is utilized to regulate the advancement of *LPs* and discover at an earlier stage any possible events to be eventually cancelled. We prove that in the batch based cancellation scheme, the set of events to be cancelled by a straggler message is fully deterministic in terms of the range generated by the rollback originator.

The proposed scheme is feasible in most Time Warp simulations. Given specific knowledge about the communication pattern of the running model, better performance can be expected. With the knowledge of the critical path in a simulation, an *LP* can send a range to those *LPs* on the path even if there is no event exchange among them. For the manufacturing model discussed previously, PSs can directly send their generated

	total events	committed events	efficiency	straggler messages	anti-messages/ cancel_messages	execution time	speedup
per-event based	406729	206521	50.78%	16987	96484	130883ms	1.00
batch based	300371	206315	68.69%	9962	38462	109961ms	1.19

Table 1. comparison of schemes with setting of fine event granularity

	total events	committed events	efficiency	straggler messages	anti-messages/ cancel_messages	execution time	speedup
per-event based	524513	237952	45.37%	22218	136016	446927ms	1.00
batch based	429634	237581	55.30%	20448	68432	408732ms	1.09

Table 2. comparison of schemes with setting of coarse event granularity

ranges to ASs and TSs to intercept any wrong events issued earlier. The extreme case is to broadcast CANCEL_MESSAGEs. Since each LP is guaranteed to be informed about the range and capable of cancelling all the events that need to be cancelled, the receiving LPs need not forward the range.

The most noteworthy overheads introduced by the batch based cancellation scheme include the communication cost of the additional state vector and increment of rollback histories carried by positive messages, the storage cost of maintaining rollback history tables and the computing cost of looking up and fossil collecting rollback history tables. With the assumption of FIFO channels, the compression approach for the rollback history is also applicable to the state vector. Observing that an LP in a simulation normally only communicates with a small number of LPs, the rollback history table at the LP can be maintained within a reasonable amount of space. Currently, the rollback history is simply implemented as a linear list of ranges, and each time an LP receives or processes an event, the list has to be fully scanned. This is believed to be one of the performance hindrances in our experiments. The next challenge to the proposed scheme is to organize rollback histories more efficiently so as to provide an optimized lookup time.

References

[1] B. Charron-Bost. Concerning the size of logical clocks in distributed systems. *Information Processing Letters*, 39:11–16, July 1991.

[2] M. Chetlur and P. A. Wilsey. Causality representation and cancellation mechanism in time warp simulations. In *Workshop on Parallel and Distributed Simulation*, pages 165–172, May 2001.

[3] A. Ferscha. *Handbook of Parallel and Distributed Computing*, chapter Parallel and Distributed Simulation of Discrete Event Systems, pages 1003–1041. McGraw-Hill, 1996.

[4] R. M. Fujimoto. *Parallel and Distributed Simulation Systems*. Wiley Book Series on Parallel and Distributed Computing. Wiley, New York, NY 10158, USA, 1999.

[5] A. Gafni. Rollback mechanisms for optimistic distributed simulation systems. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 61–67, 1988.

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

[7] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[8] Chu-Cheow Lim, Yoke-Hean Low, Wentong Cai, Wen-Jing Hsu, Shell-Ying Huang, and Stephen J. Turner. An empirical comparison of runtime systems for conservative parallel simulation. In *IPPS/SPDP Workshops*, pages 123–134, 1998.

[9] V. Madisetti, J. Walrand, and D. Messerschmitt. WOLF: A rollback algorithm for optimistic distributed simulation systems. In *Proceedings of the Winter Simulation Conference*, pages 296–305, San Diego, California, 1988.

[10] M. Raynal and M. Singhal. Logical time: A way to capture causality in distributed systems. Technical Report RR-2472, INRIA - Rennes, March 1995.

[11] R. L. Reiher, R. M. Fujimoto, S. Bellenot, and D. R. Jefferson. Cancellation strategies in optimistic execution systems. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 112–121, 1990.

[12] R. Schwarz and F. Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing*, 7(3):149–174, 1994.

[13] M. Singhal and A. Kshemkalyani. An efficient implementation of vector clocks. *Information Processing Letters*, 43:47–52, August 1992.

[14] D. West. Optimizing time warp: Lazy rollback and lazy re-evaluation. Master's thesis, University of Calgary, Calgary, Alberta, 1988.