Effectiveness of Global Event Queues in Rollback Reduction and Load Balancing

Sushil K. Prasad and Basem Naqib
Mathematics and Computer Science
Georgia State University, Atlanta
matskp@gsusgi1.gsu.edu

Abstract

Compared to highly optimized optimistic simulators which use local event queues for individual processors on a shared-memory computer, we demonstrate that employing a single global event queue drastically reduces the number of rollbacks, brings down the storage requirements, and achieves superior load balance. On a bus-based Silicon Graphics multiprocessor, these virtues consistently translated into faster execution times and higher speedups on those synthetic networks of medium- to coarse-grained logical processes which were ridden with rollbacks and load imbalance on local-event-based simulators. A dynamic randomization-based load distribution scheme for local-event-queue simulators is also shown to be an effective improvement.

1 Introduction

The optimistic discrete event simulation algorithms have proven to be efficient, versatile and robust except for those pathological networks of logical processes (lps) which cause excessive rollbacks, require inordinate amount of storage for saving past states and uncommitted messages, and those which dynamically create load imbalance thus rendering static partitioning ineffective. On uniform memory-access machines, load imbalance can be effectively reduced by relaxing the processor-to-lp affinity (as demonstrated in this work), but the problems of rollback and storage requirements persist. The primary reason is the asynchrony among various components of the simulated network.

It stands to reason that by employing a single global event queue instead of $p$ local event queues on a $p$-processor computer, the earliest $p$ messages in the entire network can be repeatedly simulated thus naturally synchronizing the entire network. The scheduling of earliest $p$ messages has been argued for as early as 1985 by Jefferson in his landmark paper [3]. Theoretically efficient implementations have been given by Prasad [4, 5] which show that, in a network of $n$ lps, $p$ earliest messages can be optimistically simulated in $O(\log n)$ time, for $p$ as large as $n$, on a parallel random access machine model.

In this paper, we report an experimental comparison between optimistic simulators employing local and global event queues. Two highly optimized local-queue-based simulators are used for comparison with the global-queue-based simulator. The first simulator, which forms the basis for the other two as well, directly inserts messages into the destination lps via locks, and into the associated local event queues implemented as parallelized calendar queues [7]. The simulator employs lazy cancellation and time windows.

The second simulator employs a dynamic load distribution via randomization. To our knowledge, no other optimistic or conservative simulator based on multiple event queues has attempted such a load distribution. The global-queue-based simulator was also easily obtained from the basic local-queue-based simulator by (i) directing all deletes and inserts to a single event queue, and (ii) switching off its time windowing mechanism.

More details and references, which could not be included for lack of space, can be found in [6]. The rest of paper is organized as follows. Section 2 describes the design of the three simulators. Section 3 contains the details of the experiments performed and the statistical data collected. Section 4 presents the results from the three simulators. Section 5 contains some concluding remarks.

2 Design of Optimistic Simulators

We first describe the basic local-event-queue-based simulator which is used to obtain the other two simulators. To simulate a network of $n$ lps on $p$ processors, $p$ local event queues are maintained. Each local queue contains a copy of the earliest message at each of the lps assigned to a processor. Each processor deletes the earliest message from its local queue, simulates it at the associated lp, then directly inserts the newly-produced messages at the destination lps. Mutually exclusive access to each lp is obtained via locks.

2.1 Data Structure of an lp

Each lp has a single channel containing all its messages, implemented as a time-ordered linked-list [9]. The rollback execution takes place lazily [2]. Fossil collection is performed whenever an lp is accessed for simulation. In each simulation cycle, the time of the earliest message simulated represents the local virtual time (lvt) for that processor. If the local queue is found empty, the lvt is advanced beyond the total simulation clock desired. After execution of a message each, the processors use barrier synchronization to calculate the gvt as the minimum of all the $p$ lvt's. The messages produced with timestamps beyond the termination condition is quickly detected when all the
of simulation tasks while others starve for work. This or two processors can get burdened with the majority

rent year variables to actually decrease concurrently. Queue.

can happen, for instance, if the overworked processor

respective of the initial lp-to-processor allocation, one

processors when simulating pathological networks. Ir-

creates, but more severely with load imbalance among

based Simulator

 requirement as the number of processors employed in-

elements in the calendar queue has generally proven to

be the best setting. As opposed to a sequential or a

conservation simulation in which the insert items are

never earlier than the last minimum, in an optimistic

environment, the minimum often needs to be slid backwards. This required specific changes in the code for

insertion to enable the last bucket pointer and the cur-

current year variables to actually decrease concurrently.

All the features described so far were incrementally
tested in varying combinations, and the current com-
bination has yielded the best performance.

Dynamic Load Distribution

The first simulator, although generally efficient,
suffers not only from rollbacks and additional storage
requirements as the number of processors employed in-
creases, but more severely with load imbalance among
processors when simulating pathological networks. Ir-
respective of the initial lp-to-processor allocation, one

or two processors can get burdened with the majority

of simulation tasks while others starve for work. This

can happen, for instance, if the overworked processor

is assigned to simulate light-weight lps with very small

service times. This causes the generation of too many

messages most of which tend to stay in the local event

queue.

On a uniform-access memory machine such as Sil-

icon Graphics or Sequent multiprocessors, with each

local queue implemented as a parallel access calendar

queue, an extremely simple technique can be employed
to achieve an effective load balance. The change

needed is directing all inserts to event queues to a

randomly picked queue. Each processor continues to

delete from its own local queue, but now finds mes-

sages from different parts of the lp network.

2.3 Global-Queue-based Simulator

This was also obtained easily from the basic sim-

ulator by directing all inserts and deletes to a single

(local) event queue. Several measures are possible
to reduce contention for deletion from this global

queue. These include using one master processor to
delete for all and then distribute in each cycle. Or,

employing a truly parallel data structure, such as a

parallel heap [7], with which we are able to obtain

a reasonable speedup for large lp networks with fine-

grained work loads. However, since the focus of these

experiments are medium-to-coarse grained networks,
such measures are not warranted. In fact, even by

using only one bucket per calendar queue thus imitat-
ing a pure linked list data structure with maximum

contentions, we found that the performance of these

fairly large-grained simulations are not adversely af-
fected significantly.

3 Experimental Details

Experimental Platform: All the experiments

were conducted on an 8-processor bus-based shared-

memory computer Silicon Graphics 4D/280GTX. Only

six processors have been employed. The other
two are normally busy with system and user tasks.
The parallel programming constructs available are

similar to that on Sequent's Symmetry Series. The

language used is C.

3.1 Network Parameters

The simulators were tested on randomly generated

networks of lps. The parameters which were varied

include: number of lps, indegree and outdegree of lps,

service times of lps, granularity of each message, the
total simulation time, number of initial messages and

their timestamp distribution. Number of lps was var-

ied from 2 to 2 and found to have similar behaviors

in terms of rollbacks, storage requirements, and load

imbalance. Therefore, we have chosen networks of a

moderate size 128 for this report. Other parameters

were varied in search for pathological networks. The

chosen parameter settings either caused a large num-

ber of rollbacks and created load imbalance, or their

values were found to have no effect on these objectives.

In the latter case, a moderate value was assigned to

such parameters. The indegree was bounded by 2 and

outdegree by 1.

Service Times: Two categories of networks have

been tested, those with static asymmetric service
times and those with probabilistic service times. For

the first category, integral service times between zero

and an upper bound k is randomly assigned to each lp

(uniform distribution). The upper bound k was var-

ied from 5 to 25; k = 10 is used in the experiments

reported here. A low and integral value of service
times were deliberately chosen to stress the calendar
queues to bring about their worst access times. To make the workload asymmetric, the service times of the first ten percent of the lps were set to zero. This caused enough rollbacks for the test networks.

The second category of networks are same as the first ones initially. However, while simulating an lp in the second category, the service time used was set to 0 with a high probability (0.7 used in the experiments). This ensured far more rollbacks than in static category to further stress-test the three simulators.

Granularity: Two sets of experiments were carried out – one with medium-grained networks and another with coarse-grained networks. For medium-grain, an empty for-loop with 4,000 iterations, which is roughly equivalent to one millisecond of compute time, was executed for each simulation of a message. For coarse-grain, a ten millisecond delay was introduced using 40,000 iterations.

The total simulation clock (the total length of the simulation) was varied from 25 to 500 with similar results. The data reported use a total simulation clock equal to 100. The number of initial messages was varied from one per lp to ten per lp; likewise their initial time stamps were randomly chosen from 0 to upper bounds of 20 to 100. Experiments reported use one initial message per lp and an upper bound of 100 on their time stamps.

3.2 Simulation Parameters

The number of buckets in each calendar queue was set at the number of initial messages and the time slice for each bucket was chosen as one. This corresponds to the best settings for calendar queues for integral timestamps. In a separate set of experiments, each calendar queue used only one bucket. In the two local-queue-based simulators, a time window of size 15 was chosen with ten percent increments per simulation cycle if a processor starves. Other values were also experimented with and these parameters settings were found to yield best performances. All data points shown in the following section are averaged over seven random networks.

Statistics collected include total execution times (seconds), maximum number of messages simulated by any one processor as an indicator of load imbalance, total number of rollbacks, average and maximum lengths of rollbacks, and average and maximum lengths of channels at lps.

4 Results

The statistical data collected are summarized into three figures for brevity. Detailed tables and figures are available in [6]. Figure 1 presents the plots for the total number of rollbacks, average length of the channels, maximum number of messages executed by a processor (maximum load), and the speedup obtained relative to the best sequential time for coarse-grained simulation on static asymmetric networks. Figure 2 plots same parameters for coarse-grained probabilistic networks. Figure 3 shows speedup plots for medium-grained networks. Other parameters are similar to those for coarse-grained networks.

The results are self evident. In terms of speedup, the standing order is the global-queue-based simulator (global) followed by the local-queue-based simu-
lator with randomized load distribution (local-dist), and then local-queue-based basic simulator (local). On static asymmetric networks, global drastically reduced the rollbacks to almost zero, whereas on probabilistically asymmetric networks, global incurs the least rollbacks. The simulator local-dist generally achieves good load balance compared to local but incurs more rollbacks on probabilistic networks than local. The channel lengths were not sufficiently strained in this study but the simulator global outperforms the other two simulators even in this regard.

Even on medium-grained networks (Figure 3), global manages to extract far better speedups than others. Other parameters are not affected by the grain size.

5 Conclusion

Global-event-queue-based optimistic simulator outperforms the local-event-queue-based simulators on bus-based shared-memory machines in simulating networks which are ridden with rollbacks and load asymmetry. For such medium- to coarse-grained networks, a significant improvement in speedup can be obtained which is generally not possible for even highly optimized local-queue-based simulators. Randomized dynamic load distribution in local-event-queue-based simulator works well for those networks, and results in an overall improvement on static workloads, and better execution times on probabilistic workloads compared to the basic simulator.

From this study, we also conclude that for medium-to coarse-grained networks, the efficiency of the parallelized data structure implementing the event queues is not so critical, for small message population.

Acknowledgement: We extend our thanks Kwok Ming Yu for producing all the plots for this project.

References


