Proc. International Mediterranean Modelling Multiconference of European Multi-Simulation Symposium (EMSS06), Barcelona, Spain, October 2006, pp. 545-551

PARALLEL QUEUING NETWORK SIMULATION WITH LOOKBACK-BASED PROTOCOLS

Gilbert G. Chen and Boleslaw K. Szymanski
Department of Computer Science, Rensselaer Polytechnic Institute
110 Eighth Street, Troy, NY 12180, USA
E-mail: {cheng3,szymansk}@rpi.edu

KEYWORDS
Lookback, queuing networks, parallel simulation.

ABSTRACT

We first describe how to design a lookback-enabled FCFS server with an infinite buffer, and how the more realistic condition of a finite buffer will complicate the modeling. We then present another design based on the idea of lazy evaluation which is not only simpler and clearer, but also more efficient. We also developed a hybrid lookback-based protocol to reduce the modeling complexity associated with direct exploitation of lookback. The hybrid lookback-based protocol is an alternative to traditional lookahead-based approach to linking simulations, yet it possesses several advantages over the latter which make it perform better.

INTRODUCTION

Queuing network simulations have been extensively studied by many PDES researchers. There are many varieties of queuing networks, differing in interconnections between components and many specific parameters of components, such as the mean of the service time, the random distribution of the service time, the scheduling policy, etc. We chose the particular one that was described in (Bargodia and Liao 1992), which is known as Closed Queuing Network, or CQN.

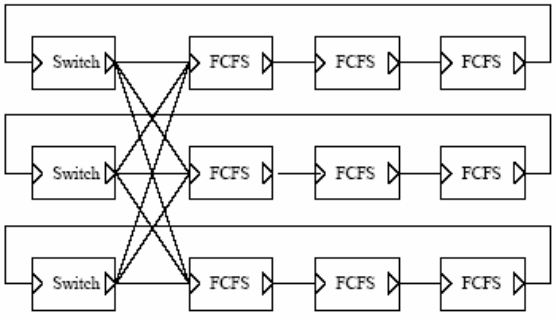


Figure 1: A CQN with 3 Switches and 9 FCFS Servers

A CQN consists of a configurable number of switches and FCFS (First-Come-First-Served) servers (Figure 1). Each packet traveling through one of the FCFS server tandems will eventually arrive at a switch. The switch will then dispatch the packet to one of the tandems randomly. CQNs are readily amenable to conservative protocols because lookahead usually comes from the FCFS servers where packets have to be delayed for random service time. Denote the arrival time, the departure time and the service time of the *i*-th packet P_i by Ai, D_i *and* S_i respectively. If the service is available at Ai, then $D_i = A_i + S_i$ otherwise $D_i = D_{i-1} + S_i$.

The departure time can be determined as soon as the packet arrives. If eager scheduling is employed, the departure event will be immediately generated and sent out. The lookahead, which is equal to $D_i - A_i$, is at least S_i. Nicol noticed that lookahead can be augmented if the service time of the next packet can be pre-sampled (Nicol 1988). In such a case, the lookahead becomes $D_i - A_i + S_{i+1}$. The service time must be independent of packets if the service time is to be sampled ahead of time. This, however, may not always be true in practice. A more general case is the one in which a lower bound on the next service time can be obtained, and the lookahead is now $D_i - A_i + min\{S_{i+1}\}$.

(Bagrodia and Liao 1992) proposed a technique to reduce the rollback distance for optimistic CQN simulation. They distinguished between the receive time of a packet and the time at which a packet can be served. When a straggler packet arrives, the logical process needs to be rolled back only to the earliest time this message can be served. This approach bears some resemblance to lookback-based protocols, for both algorithms stem from the observation that rollback to the timestamp of the straggler is unnecessary. However, lookback-based protocols can totally eliminate stragglers by deliberately delaying the execution of events that would increase the virtual lookback time above the minimum timestamp of any stragglers.

DEFINING LOOKBACK IN CQN SIMULATION

Lookback is defined as the ability of a component to change its past without affecting others (Chen and Szymanski 2002b, Chen and Szymanski 2003). It enables a new class of synchronization protocols that lie in between conservative and optimistic protocols.

Before applying lookback-based protocols to the CQN simulation, we must make sure that every component in the CQN contains substantial amount of lookback. This is, however, not a necessary condition for the applicability of lookback-based protocols. Even in a

simulation where lookback is zero in every component, they still work, because the earliest event in the entire simulation is always guaranteed to be eligible for execution. Without adequate lookback, the performance of these protocols would be greatly impacted.

Let us first look at the switch component only. The function of a switch is to generate a uniformly distributed random integer to determine the destination of a packet. Normally a switch keeps a private random number generator which is invoked when a packet arrives. A problem arises with this solution when packets can arrive in out-of-timestamp order. The statistical properties of the destination distribution remain unaffected if the random numbers are still generated in the same way. However, the parallel simulation is no longer *repeatable*, meaning two simulation runs with exactly the same set of parameters may produce totally different results as out-of-timestamp order events are produced when different processors advance the simulated time unevenly; therefore they depend only on the runtime behaviors of the simulation, which is completely uncontrollable.

A simple solution to preserve the repeatability is to let the switch obtain a random number from a random number generator carried by each packet. This would not be a burden in terms of memory usage because a random number generator in fact does not occupy much space. For instance, a Linear Congruential Generator requires as few as 8 bytes. Packets passing through the switch now become completely independent of each other.

Nevertheless, this solution is unnecessary in this particular CQN simulation. We noticed an interesting fact that a switch may never receive out-of-timestamp order packets, because a switch receives packets from only one FCFS server. The simulation would not have received out-of-timestamp order packets, because a switch receives packets from only one FCFS server. The simulation would not have been correct had a switch received a straggler. Therefore, the switch component needs no change in lookback-based CQN simulation.

As to the FCFS server, we have proven that in it lookahead under eager delivering and lookback under lazy delivering are always the same (Chen and Szymanski 2002a, Chen and Szymanski 2003). With lazy delivering, the virtual lookback time of the FCFS server at any time is always equal to the arrival time of the last packet leaving the server. Any received packet with a timestamp smaller than that of the current packet in service must preempt the later. All those stragglers with a timestamp greater than or equal to the virtual lookback time can be correctly inserted into the waiting queue at positions decided by their timestamps. For packet P_i at its departure time D_i, the virtual lookback time is Ai, resulting in a lookback of D_i-A_i equal to the lookahead under eager scheduling given previously. If pre-sampling is employed, at time $D_i+min\{S_{i+1}\}$ which is the earliest time next packet can complete the service, the virtual lookback time is still A_i. Therefore at this instant the lookback is D_i-A_i+$min\{S_i+1\}$, still the same as the lookahead.

Apparently, packets with timestamp smaller than the virtual lookback time will trigger a causality error. But such errors are preventable by adhering to the lookback constraint: when a packet is about to leave the server, its arrival time must be checked against the LBTS. If the arrival time is greater than the LBTS, it means that another packet with a smaller timestamp may arrive later. The packet cannot be sent out, and the departure event is returned back to the local event list. Two cases can happen afterwards. Either the LBTS is advanced by invoking a new round of LBTS computation, making the same event eligible for execution according to the lookback constraint, or a new event with a smaller timestamp may be received which satisfies the lookback constraint.

LOOKBACK-ENABLED FCFS SERVER

The FCFS server has two event handlers. One is called when a new packet is received, the other is called when an in-service packet is about to leave the server. For simplicity, let us first assume that the FCFS sever has an infinite buffer so that no packet will be dropped. We will come back to the case of finite queues in the next section.

Handling Arrivals

The sequential algorithm for handling the arriving packets is simple. If the server is free, the departure event is scheduled, and the packet is marked as in-service while the server is marked as busy. Otherwise, the packet is placed at the tail of the internal queue. When the packet is a straggler, there are only two ways to handle the packet. Either the packet must be marked as the current in-service packet, or it must be inserted into the internal queue at a position determined by its timestamp. If the server is free, the first case must be chosen. If it is not, the current in-service packet may have a larger timestamp, so the arriving packet can preempt the current in-service packet, cancel the old departure event and schedule a new one. Only when neither of these two conditions is true will the packet go to the internal queue.

How to compute the departure time is worth elaboration. In the case of a straggler packet, it could happen that the last in-service packet with a departure time later than the timestamp of the straggler has already left the server, and the service starting time should be the last departure time instead of the

timestamp of the straggler. Consequently, the correct formula to calculate the departure time should be $D_i=max\{A_i,D_{i-1}\}+S_i$. It can be verified that this is in accordance with the other formula for calculating D_i given earlier. The implication is that we must keep track of the departure time of last packet. This can be done by writing the value of the departure time to a local variable when a packet is about to leave.

Handling Departures

In the sequential case, processing the departure event is also simple, the packet is sent out via the outport and if the internal queue is not empty, the packet at the head of the queue moves to service.

With stragglers, one significant difference is that before delivering the in-service packet to the outport, its arrival time must be checked against the LBTS. If the arrival time is smaller than the LBTS, the event handler returns immediately with a false value, indicating that the departure event cannot be successfully processed.

A departure event could be a straggler. This occurs when a packet straggler schedules a departure event earlier than the current simulated time. Therefore, when computing the departure time for the new in-service packet, the starting service time must be the maximum of its arrival time and the departure time of the packet just departing (or the timestamp of the current departure event).

Dealing with Finite Queues

So far the correctness of the lookback-enabled FCFS server is based on an assumption that the internal queue is infinite so no packet would be dropped. When stragglers are allowed, the decision whether or not a packet should be dropped due to a full queue is extremely difficult to make.

To see how complicated the situation is, let us look at a seemingly plausible hypothesis that if a packet arrives at a time when the internal queue is already full, then this packet is destined to be discarded, no matter how many packets with a smaller timestamp will be received later.

The intuition behind this hypothesis is that a straggler packet can only make the internal queue more crowded. It would have been very useful if it were true because the decision of dropping a packet could then be made immediately once the packet arrives when the internal is full. However, it is not sustainable, due to the fact that a straggler packet may leave earlier than the current simulated time. Figure 2 depicts such an example.

In the example we assume that the size of the internal queue is 1, meaning that any packets other than the in-service packet have to be discarding. At *T1* Packet *A* comes after packet *B*, so it has to be dropped if there are no stragglers. When a straggler *C* arrives at *T4*, not only does it force packet *B* to be dropped, but also it leaves early at *T2* to allow packet A to occupy the empty server!

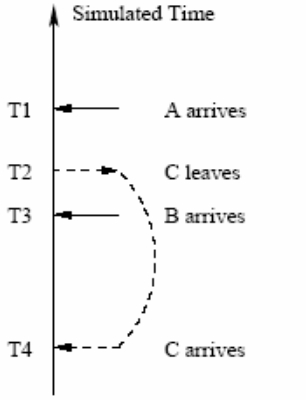


Figure 2: A Counterexample for the Hypothesis

Initial Positions

The implication of the above example is that the decision of dropping a packet cannot be made at the time the packet just arrives. A straightforward solution is to keep track of *initial positions*, which can be used to help make the dropping decision. The initial position of a packet is defined as the number of packets in the queue at the arrival time of the packet if all packets were received in the timestamp order. The sufficient and necessary condition for a packet to be discarded is to have an initial position greater than or equal to the size of queue.

At what time should we make the dropping decision? We only have two choices: either at the time the packet enters the service or at the time the packet leaves the server. The latter seems feasible, because a packet only leaves when it cannot be affected by any stragglers, so its initial position in the queue will not change any more. However, it is not a good choice for two reasons. First, it is a waste of CPU time to schedule and process the departure event of the to-be-dropped packet. We should try to avoid scheduling these events. The second reason will be presented when we discuss the possibility of mixing up lookback-enabled components and sequential components. This leaves the former option, making the dropping decision when the packet enters the service. At that time the initial position of the packet is still subject to changes caused by stragglers and by the disproof of *Hypothesis* whether or not a packet should be discarded is never certain. Fortunately, we can assure that the case depicted in hypothesis would never happen when the packet is entering the service.

Theorem: At the time a packet is to be marked as in-service, if its initial position is greater than or equal to

the size of queue, then it is destined to be discarded anyhow.

Proof. As shown in Figure 3, suppose that the current simulated time is *T2* and the packet *A* with an arrival time of *T3* is about to start the service. It is allowed to do so because the packet *B* which arrived at *T5* has just completed its service (its service starting time is irrelevant to discussion here). There should be no other arrival events between *T3* and *T5* at this time, because otherwise it would be such an event that enters the service at *T2*. Any straggler that is received later must have a timestamp no smaller than *T5*, because otherwise packet *B* would not be allowed to leave and consequently packet *A* would not have any chance to start the service (in the departure event handler the lookback constraint is always checked first). If any straggler occurs, say, packet *C*, with a timestamp between *T3* and *T5*. Its departure time must be greater than *T2*, because it arrives later than packet *B*. Therefore, no departure events can be added to the window [*T3*,*T5*] after the simulated time reached *T2*, which means the initial position of packet *A* can only be increased once it enters the service. □

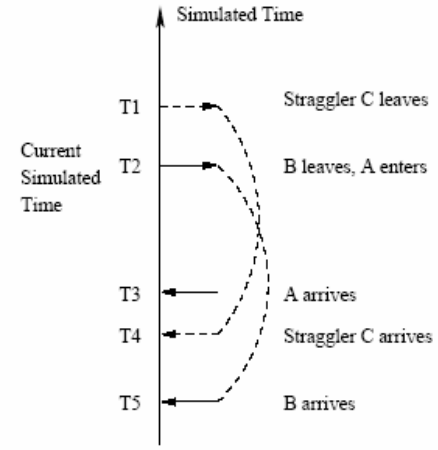


Figure 3: A Straggler Can Only Increase the Initial Position of Another at Entering the Service

Notice that according to *Theorem* we are sure that a packet can be dropped if it has an initial position that is too large, but we are uncertain whether or not the a packet with a small initial position can survive the entire service time. It is possible that a number of stragglers (sometimes one is enough) preempt a packet that is already in service and increase its initial position forcing it to be dropped.

Now the rest of the task is to calculate the initial position for every packet. For this purpose we must maintain a departure time list that records the times at which packets leave the *server (*if the in-service packet is viewed as not occupying a place in the queue, the departure time list would contain the time each packet leaves the *queue* and becomes the in-service packet).

Since there are only two types of events for an FCFS component, and each event could be either a straggler or not, we need to consider these four cases as well as an additional one in which a packet is dropped. For a non-straggler packet, the initial position is simply the number of packets currently in the queue. It will not affect the initial position of any other packets. For a straggler packet, calculating the initial position those packets that have already left but whose departure time is greater than the timestamp of the straggler must be considered. It can be shown that these packets must be in the queue at the time the straggler arrives (otherwise they would not leave earlier than the straggler). The number of these packets can be obtained by scanning through the departure time list. The initial position of a straggler packet is then the number of these packets plus the number of packets currently in the queue with a smaller timestamp. A straggler packet would increase by one the initial position of every packet with a larger timestamp. A non-straggler departure event does not change the initial position of any other packets. A straggler departure event, triggered by a straggler packet that leaves earlier than the current simulated time, decreases by one the initial position of every packet in the queue with a larger timestamp. Finally, a dropped packet decreases by one the initial position of every packet in the queue with a larger timestamp.

Lazy Evaluation

The algorithm to maintain the initial positions for all received packets is not only complicated but also inefficient. The worse case happens when a packet to be sent into service is found out having to be dropped. The entire queue must be scanned in order for initial positions of all the packets currently in the queue to be decreased by one. Apparently, FCFS components based on this algorithm can only be fitted to the cases where packets are rarely dropped.

A better design originates from the fact that initial positions can be derived recursively. Denoting by IP_i the initial position of the *i*-th packet, we have IP_i=0, if the server is free at A_i and $IP_{i-1}+1-LP[A_{i-1},A_i]$ otherwise, where $LP[A_{i-1},A_i]$is the number of packets leaving between A_{i-1} and A_i.

The useful observation is that the above formula for IP_i will have all its arguments fully determined by the time when the *i*-th packet is entering the service. At this time the *i-1*-th packet has already left, so D_{i-1} is known. Whether the service is free at A_i can be determined by comparing D_{i-1} with A_i. If $A_i \geq D_{i-1}$, the *i*-th packet arrived after the last one left, so the server is free and the packet can be immediately put into service. Otherwise, $D_{i-1}>A_i$, and IP_i must be calculated from IP_{i-1} and the number of packets leaving between A_{i-1} and A_i. We only need to make sure the no straggler can leave within the window $[A_{i-1},A_i]$. A straggler cannot arrive before A_{i-1} because the *i-1*-th packet has already left.

Any straggler arriving after A_{i-1} must leave at a time later than D_{i-1}, which is greater than A_i. Hence, after i-th packet enters the service, LP[A_{i-1}, A_i], the number of packets leaving between A_{i-1} and A_i is fixed and cannot be affected by any stragglers.

This idea reflects the general guideline of *lazy evaluation* in designing lookback-enabled component: *everything should be computed as late as possible*. The previous implementation, referred to as *eager evaluation*, attempts to calculate the initial position as soon as the packet arrives. Obviously, the calculation of the initial position at the arrival time is wasteful, for it will much likely be changed by the arrival of a straggler. At the time the packet enters the service it is much less likely that a straggler would occur, so it is meaningful to carry out the computation now. Notice the result of computation should be regarded as merely estimation, since it may still be affected by a straggler, though with a much smaller probability. This estimation is only true when no straggler would occur. The dropping decision based on the estimation, however, is guaranteed to be always correct: we would never drop a packet that should not be dropped.

Performance Comparison

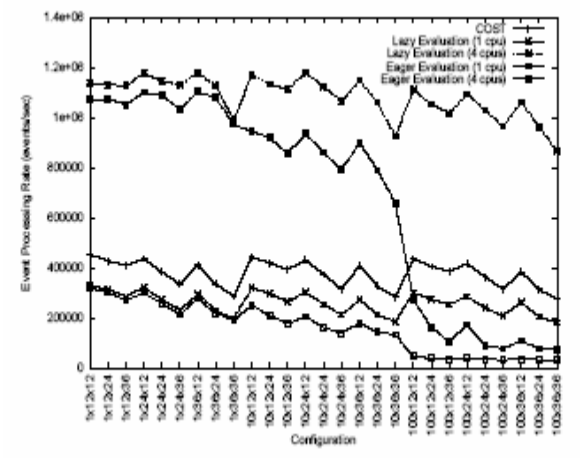


Figure 4: Eager and Lazy Evaluation of Initial Positions

To see the effects of calculating the initial positions at different times, we made two changes on the simulation. The capacity of the FCFS queue was set equal to the number of packets initially in the queue, and new packets are created by a Poisson process on each queue continuously throughout the simulation. Figure 4 shows their performances with the LB-GVT protocol. As mentioned before, the inefficiency of the eager evaluation comes from in-service packets that have to be discarded, for these packets affect all other packets currently in the server. The effect becomes apparent when each server can hold a large number of packets.

A HYBRID LOOKBACK-BASED PROTOCOL

The only disadvantage of lookback-based protocols is their associated modeling complexity. Example is the lengthy discussion in the previous chapter about the implementation of the lookback-enabled FCFS server with finite buffer, which is a trivial problem if events are received in timestamp order. The most effective technique to alleviate the modeling complexity is design a hybrid lookback-based protocol that is capable of mixing lookback-enabled components with sequential components.

To understand how the hybrid lookback-based protocol works, it must first be clarified where stragglers come from. If all time-aware components on the same processor share a single simulation clock, it is clear that only those components that reside on the boundary can receive stragglers from other processors. Thus a natural idea is to "absorb" all stragglers in the boundary components so that other intra-processor components will never observe any stragglers.

Lookahead must exist in the boundary components to ensure that the intra-processor components do not to receive stragglers. Consider an event e in an intra-processor component with a timestamp T'. Suppose that the current simulated time of the processor is T and $T'>T$. Since all components in the same processor share the same simulation clock, all boundary components must be at the same simulated time T. If they send out in the future any messages with timestamp less than T', the event e could be affected so its execution cannot continue. Therefore, the sufficient but not necessary condition for event e with timestamp T' to be safely executable is that all boundary components must guarantee that all messages they will send later must have a timestamp greater than or equal to T'. By definition of lookahead, this means that every boundary component must have a lookahead of at least T'-T.

Another requirement is that a boundary component can never output a message smaller than the current simulated time. Suppose that a boundary component at simulated time T outputs a message with timestamp T' such that $T'<T$. It is possible that a previously processed event e in an intra-processor component contains a timestamp T'' satisfying $T'<T''<T$. Apparently the event e may be affected by the message at T', which becomes a straggler in the intra-processor component.

This requirement rules out the choice of making the dropping decision when a packet is departing from the FCFS server. If the departing packet is to be dropped after the calculation of its initial position, the next packet may have an earlier departure time, given that the service time is dependent on the packet itself. The next packet would then leave at a timestamp earlier

than the current simulated time, resulting in a potential straggler for other components.

GUARD EVENTS

Guard events are scheduled by boundary components to set a lower bound on the timestamp of events they will send. The simulation engine handles the guard events in the same way as regular events, so these two types of events can share the same event list. When the simulation engine sees a guard event, it dispatches the guard event to the boundary component that scheduled it. The component may or may no schedule another guard event with same or greater timestamp, depending on the estimated minimum timestamp of future events.

Inclusion of guard events demands no changes on the lookback-based protocols. A regular event will be chosen only if it is the earliest event in the processor, therefore at this time there cannot be any guard events with smaller timestamp. For convenience, we refer to the LB-GVT protocol that allows for guard events as the hybrid lookback-based protocol, or the Hybrid LB-GVT protocol. The LB-EIT protocol is not considered because its relatively poor performance.

There is a slight difference between guard events and regular events: the GVT computation should not take guard events into account. Otherwise, the hybrid lookback-based protocol would be deadlock-prone with a zero lookahead boundary component, because the guard event with the smallest timestamp would be chosen to execute which will in turn schedule another guard event with the same timestamp, resulting in an infinite chain of guard events. If the GVT is computed from the timestamps of regular events, then the earliest regular event is always eligible for execution, because all guard events must have a timestamp equal to or greater than the GVT calculated in this way.

Guard events may become unnecessary when a boundary component has a regular future event which is known to be the earliest among all future events, because in this case any guard event must have a timestamp greater than that of the earliest regular event, so it can be scheduled after the regular event has been processed.

COMPARING HYBRID LOOKBACK-BASED AND CONSERVATIVE PROTOCOLS

In the FCFS server detailed in the last chapter, a guard event is needed when there are no packets in the server or all packets in the server have an arrival time greater than the GVT. It is needed in the latter case because another arriving packet preempting the current in-service packet may leave earlier than the current in-service packet. The guard event is not needed when the current in-service packet arrived before the GVT, since the departure event of such a packet cannot be affected by a straggler and therefore is the earliest among all future events. The timestamp of the guard event, if necessary, is calculated according to the following formula:

GT_i=max{last-departure-time, GVT}+max{S}.

min{S} represents the minimum service time of any packet.

For comparison, we developed a similar lookahead-based protocol, referred to as LA-EIT, which exploits only the lookahead on the boundary components. Here the set of boundary components are different from those defined for the hybrid lookback-based protocol, for they are now components that may send messages to other processors, rather than components that may receive messages from other processors.

The LA-EIT protocol works as follows. Denote EOT_{ij} as the Earliest Output Time of messages any boundary components residing on the *i*-th processor may send to the *j*-th processor. The Earliest Input Time of the *i*-th processor is $EIT_i=min_j\{CLOCK_j+LA_{j,i}\}$, where $CLOCK_j$ is the current simulated time of the *j*-th processor, and $LA_{j,i}$ is the lookahead in the boundary components between the *j*-th and *i*-th processor. After the EIT is known, each processor can then execute safe events with a timestamp smaller than the EIT. Notice this protocol is prone to deadlock if there exists a cycle of zero lookahead, as in the traditional Chandy/Misra/Bryant protocol.

Despite of the increased modeling complexity, the hybrid LB-GVT protocol has several advantages over the LA-EIT protocol . Guard events incur less overhead because they are scheduled and received within the same processor, whereas null messages must be sent from one processor to another. They also disappear automatically when lookahead becomes larger than the difference between the GVT and the current simulated time, whereas in the LB-GVT protocol null messages are indispensable for advancing the simulated time. The GVT computation is less sensitive to topology. The EIT computation requires a large number of null messages with high component connectivity. LB-GVT is deadlock free even if the lookahead is zero.

EXPERIMENTAL RESULTS

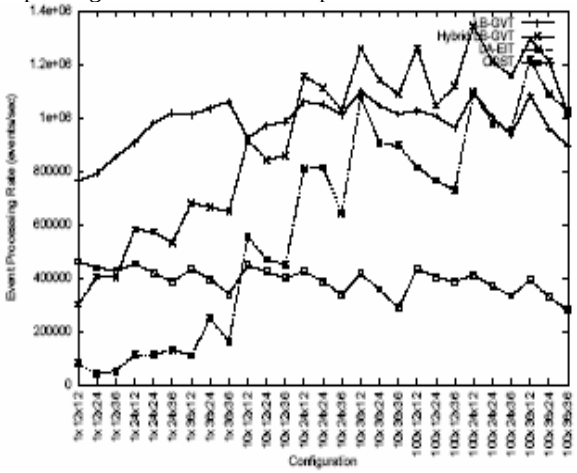
The top of Figure 5 shows the performance of LB-GVT, hybrid LB-GVT, and LA-EIT running on 4 processors and of COST which is our sequential simulation tool described in Section 3. The LB-GVT protocol is the most stable, for it is able to exploit lookback intra-processor components when there are few packets in each server. The hybrid LB-GVT and LA-EIT protocols work better when there are abundant packets, mainly due to the fact that intra-processor components can be more efficiently implemented as sequential component. Overall, the hybrid LB-GVT protocol consistently outperforms the LA-EIT protocol.

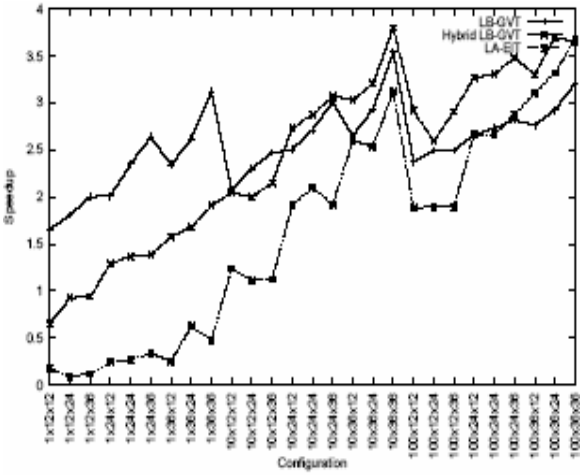
The middle of Figure 5 shows the speedup of these three protocols on 4 processors with respect to the COST and the bottom, with respect to their sequential execution. The speedup of the LA-EIT protocol relative to its sequential execution is the worst. This is mainly due to the topology of the CQN network. When a packet arrives at a switch, the switch may to dispatch one null message for each processor to inform the changes on EOTs, even if only one processor is the destination of the packet.

CONCLUSION

The CQN simulation was our first attempt to apply lookback-based protocols. Initial performance results were promising. On 4 processors we were able to obtain nearly linear speedup. This was the very first evidence that lookback-based protocols did work.

We also proposed a hybrid lookback-based protocol that is able to mix up lookback-enabled components and sequential components. By only exploiting lookback in the boundary components, it relieves the burden of modeling dramatically. The experiments on CQN simulation showed that it outperforms a conservative protocol based on lookahead. Granted, the results are fully extensible to other kinds of simulation. For instance, FIFO (First-In-First-Out) links that are widely used in communication networks are an FCFS server with a delay. A lookback-enabled FIFO link can be implemented by adding a few changes to the FCFS components detailed in this paper. It will then enable us to conduct communication network simulations by only exploiting lookback in link components.





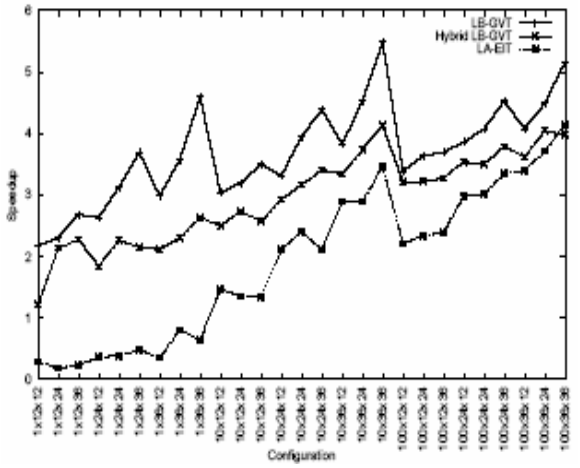


Figure 5: Performance, Speedup relative to COST and Speedup relative to Sequential Execution for Three Parallel Protocols (4 CPUs) on CQN

ACKNOWLEDGEMENT

This work was partially supported by the NSF grant OISE-0334667. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Bagrodia R.L. and W.-T. Liao. 1992. “Transparent optimizations of overheads in optimistic simulations”. In *Proceedings of the 1992 Winter Simulation Conference*}, 637-645.

Chen G. and B.K. Szymanski. 2002a. “Lookahead, rollback and lookback: Searching for parallelism in discrete event simulation”. In *Proceedings of the Summer Computer Science Conference*.

Chen G. and B.K. Szymanski. 2002b. “Lookback: A new way of exploiting parallelism in discrete event

simulation". In *Proceedings of the 16th Workshop on Parallel and Distributed Simulation*, 153-162.

Chen G. and B.K. Szymanski. 2003. "Four types of lookback". In *Proceedings of the 17th Workshop on Parallel and Distributed Simulation*, 3-10.

Jha V. and R.L. Bagrodia. 1993. "Transparent implementation of conservative algorithms in parallel simulation languages". In *Proceedings of the Winter Simulation Conference*, 677-686.

Mascarenhas E. et~al. "Minimum cost adaptive synchronization: Experiments with the Parasol system". *ACM Transactions on Modeling and Computer Simulation* 8, No. 4, 401-430.

Nicol D.M. 1988. "Parallel discrete-event simulation of FCFS stochastic queuing networks. *SIGPLAN Notices* 23, No. 9, 124-137.