Appeared in MASCOTS '94

Distributed Simulation of Large-Scale PCS Networks

Christopher D. Carothers and Richard M. Fujimoto
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332

Yi-Bing Lin and Paul England
Bellcore
Morristown, New Jersey 07962

Abstract

There has been rapid growth in demand for mobile communications over the past few years that has led to intensive research and development of complex PCS (personal communication service) networks. Capacity planning and performance modeling is necessary to maintain a high quality of service to the mobile subscriber while minimizing cost. Simulation is widely used in such studies, however, because these models are extremely time consuming to execute, only small-scale PCS networks have previously been simulated. In this paper, we examine the use of the Time Warp distributed simulation mechanism in simulating large scale (1024 or more cells) PCS networks. An object-oriented, distributed discrete event simulator using Time Warp *has been developed, and initial performance measurements completed. Speedups in the range of 2.8 to 7.8 using 8 Unix workstations have been obtained, enabling simulation runs that require 20 hours on a single workstation to be completed in only 3.5 hours.*

1 PCS Network Simulation

A personal communication service (PCS) network [3] is a wireless communication network, which provides communication services for *portables* or *PCS subscribers*. The service area of a PCS network is partitioned into several sub-areas or *cells*, and radio channels are assigned to the cells. When a subscriber makes a phone call, the system tries to allocate a radio channel for the connection. If all channels are busy, then the phone call is blocked. It is important to engineer the system so that the blocking probability is low (e.g., less than 1%).

Discrete event simulation is a popular technique for designing and modeling large high-capacity personal communication services (PCS) networks. In a PCS simulation, the network is usually represented by hexagonal or square cells. Ideally, PCS simulation experiments should model large networks containing thousands of cells. However, due to limited computing resources, most studies only examine small-scale networks containing fewer than 50 cells [7, 13], and output statistics of the boundary cells are usually discarded to avoid the boundary effect. Lin and Mak [9] showed that this approach may lead to biased output statistics, and suggest using a large network with a wrap-around topology to achieve reliable simulation results.

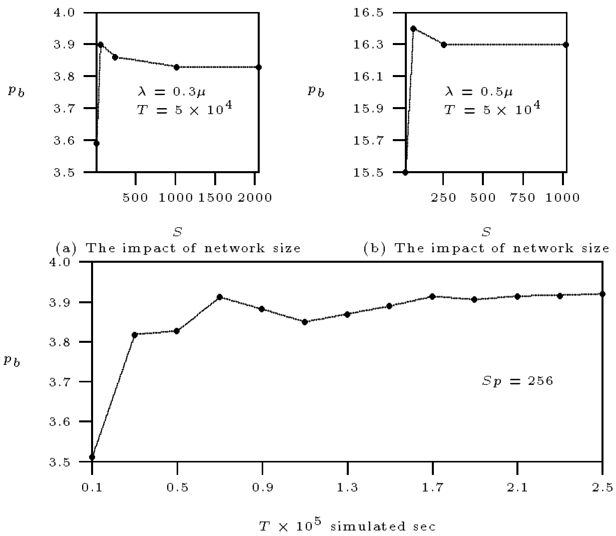
To illustrate the impact of network size, consider the following PCS simulation. Assume that each cell is assigned a fixed number of radio channels, referred to as the fixed channel assignment scheme. A portable (or mobile phone) resides at a cell for a period of time, then moves to another cell. When a phone call arrives, a radio channel is used to connect the corresponding portable. If all channels are busy, then the call is blocked (dropped). The output measure is the blocking probability p_b (the probability that a phone call is dropped).

The blocking probability of a fixed channel assignment PCS network can be modeled analytically using the Erlang-B system [7, 13]. However, Lin and Chen [8] showed that the Erlang-B system is only appropriate for predicting the blocking probability when the number of portables is much larger than the number of available channels. Further, unlike a simulation, an analytic model for a fixed channel assignment PCS network cannot be easily modified to study other aspects of a PCS network such as the handoff effect [11] and dynamic channel assignment [7, 13].

Here we assume that the expected number of portables in a cell is N. A portable resides in a cell for a period of time t (in this paper, t has an exponential distribution with mean $\frac{1}{\eta}$, as proposed in [12]), then move to one of the four neighboring cells with equal probability. The call arrivals to a portable form a Poisson process with the arrival rate λ. The call holding times have an exponential distribution with mean $\frac{1}{\mu}$. Figure 1 (a) and (b) illustrate the impact of the network size S (where the simulation covers $T = 5 \times 10^4$ seconds of simulated time, the mean call holding time is $\frac{1}{\mu} = 180$ seconds, $N = 50$ and $\eta = 0.04\mu$).

The figures suggest that $S > 256$ cells are required to avoid the wrapping effect. Figure 1 (c) suggests that $T > 5 \times 10^4$ (simulated) seconds are required to obtain the steady state blocking probability. For $S > 256$ and $T > 5 \times 10^4$ seconds, the number of events executed is on the order of 10^7. We have observed between 10^3 and 10^4 events per second in commercial discrete

Figure 1: Estimating the problem size



(a) The impact of network size (b) The impact of network size

(c) The impact of the termination time

event simulation packages, such as ModSim [2] on a high performance engineering workstation, so a simulation of this magnitude will require many hours of CPU time. Execution times will increase as network complexity and the functionality of the simulator increase. Further, such large simulation programs may require more memory than is available on a single machine.

The remainder of this paper focuses on using a collection of networked workstations to speedup the simulation. Section 2 describes the the synchronization problem that arises in executing discrete event simulations on computers containing multiple processors, and the Time Warp mechanism used to address this problem. An implementation of Time Warp that we have developed is also briefly described. Section 3 reviews the PCS simulation model that was implemented using the distributed Time Warp system. Section 4 describes results of a study to measure the performance of this implementation. Section 5 presents conclusions of this study, and future work.

2 Distributed Time Warp

For a detailed description of the Time Warp mechanism we refer the reader to [5]. A version of Time Warp has been developed that executes on a collection of DEC 5000 workstations, Sun Sparc workstations, or a mixture of these machines. All performance results presented here were performed on DEC machines interconnected by an Ethernet. The Time Warp system is written in C++. A principal objective of this implementation is to enable efficient simulation of thousands of "light weight" LPs (i.e., processes that contain a small amount of state and perform little computation in each event) in an object-oriented environment on networked, heterogeneous workstations. Here we will describe some of the user and kernel level features of the Time Warp system.

LPs are implemented as C++ objects, and are referred to as "entities." Each LP (entity) consists of a state vector that stores the LP's private data, and a set of methods that define the allowable operations that can be performed on that data. Each method corresponds to a type of event. For instance, in a PCS simulation, an entity for a cell will include methods invoked by mobile units to enter or leave that cell.

It is anticipated that most simulations will contain far more entities than processors, so each processor will typically contain hundreds or thousands of entities (LPs). A priority queue data structure called the calendar queue [1] is used in each processor to efficiently select the next entity to execute. The processor's scheduler always selects the entity containing the smallest timestamped event as the next one for execution. Each entity includes a linear list to hold the unprocessed events (method invocations) scheduled for that entity. It is anticipated that the number of unprocessed events for each entity will remain small (often only a few events) for most applications.

Interprocessor communications are handled by *pTask*, a library and programming environment for developing distributed C++ applications [4]. The system supports heterogeneous platforms and automatically converts data sent between different types of machines. *pTask* also includes support for special nodes such as supercomputers and graphics workstations. Communications between identical computational nodes circumvent the conversion routines, thereby improving performance. The system also provides a set of graphical utilities for debugging and network monitoring.

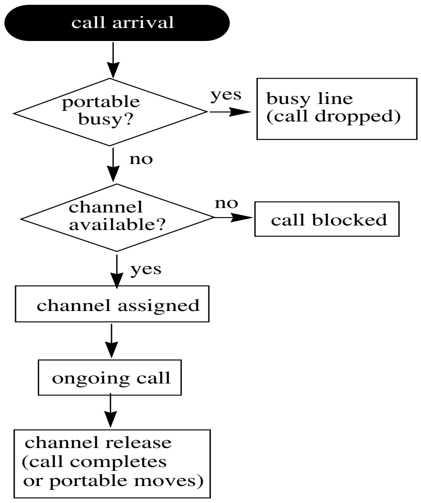
3 Description of the PCS Model

The PCS application model is organized around two object types: *Cell* and *Portable*. The Cell represents a cellular receiver/transmitter that has some fixed number of channels allocated to it and is the central entity (LP) object type for the simulation. The Portable represents a mobile phone unit that resides within the Cell for a period of time and then moves to one of the four neighboring Cells. The behavior of a Portable is modeled by different types of events, such as Portable move, call arrival, and call completion.

The PCS model views the Portable as only a receiver of calls and is not concerned with the originator of a call. As shown in figure 2, when a new call arrives at a Cell [1], the Cell first determines the status of the destination Portable. If the destination Portable is busy with another call, this call counts as a *busy line*. If the Portable is not busy, the Cell determines channel availability. If all channels are busy, this call is counted as a *block*. If a channel is available, it is allocated for the destination Portable's use and the call is allowed to connect.

[1]We assume call interarrival times are exponentially distributed.

Figure 2: Flowchart for PCS model.



While a call is in progress [2], the Portable tracks its location. When the Portable determines it is moving [3] out of the current Cell's signal range, it drops the currently used channel, and requests a channel from the nearest neighbor Cell into which it is moving. If all channels from the neighboring destination Cell are busy, this call counts as a *handoff block*, also known as a *forced termination.* If a channel is available, the call reconnects and continues without interruption.

To implement the physical functionality of a PCS network, we wrote a *self-initiating* simulation [10] [4], where the Cells, via the Portables, generate their own incoming calls. This aspect of our implementation will become clearer in the discussion below and has a significant effect on the behavior of the distributed simulation.

First, the Cell object instances are statically mapped onto the physical processors in a grid like pattern that minimizes the communications between processors. During creation, each Cell sends itself an initial event that invokes the *StartUp* method. Upon processing the *StartUp* method, each Cell will initialize and schedule some fixed number of Portables. Each Portable contains three independent exponentially distributed timestamp fields: call completion, next call and move timestamps. The call completion timestamp represents the time at which the Portable will complete the current call. If a call is not in progress the call completion timestamp is ∞. We assume that no calls are initially in progress. The next call timestamp represents the time when the next call will arrive. The move timestamp represents the time at which the portable will leave the current cell and reside at one of the neighboring cells. The next call and move timestamps are initialized by independent exponentials with different means. The minimum of these three timestamp fields determines the type and destination of the Portable. If the next call timestamp is the minimum, then the Portable is scheduled to be sent to the current Cell, also known as *Self*, at the next call timestamp. Upon receiving this Portable, the Cell will invoke the *NextCall* method. If the move timestamp is the minimum, then the Portable is scheduled to be sent to one the current Cell's randomly selected neighbors at the move timestamp. Upon receiving this Portable, the neighboring Cell will invoke the *MoveIn* method.

Upon invoking the *NextCall* method, the Cell checks channel availability. If all channels are busy, then this call is counted as a *blocked* call and the next call timestamp of the Portable is incremented by an exponentially distributed timestamp. Now, if a channel is available, it is allocated. Next, the call completion timestamp is incremented by an exponentially distributed timestamp and the minimum of the three timestamp fields is computed. If the call completion timestamp is the minimum, then the Portable is scheduled to be sent to the current Cell at the call completion timestamp. Upon receiving this Portable, the Cell will invoke the *CompletionCall* method. If the move timestamp is the minimum, then the Portable is scheduled to be sent to the current Cell at the move timestamp. Upon receiving this Portable, the Cell will invoke the *MoveOut* method. If the next call timestamp is the minimum, then that call is counted as a *busy line* call because this Portable is already engaged in a call. The next call timestamp is then incremented by an exponentially distributed timestamp and the minimum is computed again. The minimum of the three timestamp fields is computed repeatedly until either the call completion or move timestamps are the minimum.

Upon invoking the *CompletionCall* method, the Cell resets the call completion timestamp of the Portable to ∞ and frees the channel currently held by the Portable. Again, the minimum of the three timestamp fields is computed. The actions are identical to those described for the *StartUp* method. Note that when a call completes, a Portable will generate its next incoming call. This is the heart of the simulation's *self-initiating* process.

Upon invoking the *MoveIn* method, the Cell increments the move timestamp by an exponentially distributed amount. Next, the Cell determines if this Portable has a call in progress. If a call is in progress, then the Cell checks channel availability. If all channels are busy, then this call is counted as a *handoff block*, the call completion timestamp is reset to ∞ and the minimum of the three timestamp fields is computed. The actions are identical to those described for the *StartUp* method. If a channel is available, then that channel is allocated. Again, the minimum of the three timestamp fields is computed. The actions are identical to those described for the *NextCall* method. If no call is in progress, then the actions are identical to those described for the *StartUp* method.

[2] We assume the call duration is exponentially distributed with mean 3 minutes.

[3] We assume Portable movement is exponentially distributed.

[4] The PCS simulation is actually a hybrid of self-initiating and message-initiating models.

Upon invoking the *MoveOut* method, the Cell frees the channel currently held by the Portable and schedules the Portable to be sent to a randomly selected neighboring Cell at the move timestamp. Because handoffs occur virtually instantaneously, simulated time is NOT incremented. This method is only invoked, when a call is in progress and the Portable is moving between Cells.

4 Performance Results

We conducted experiments to measure speedup where the number of portables per cell, mobility and call interarrival times were varied. The completion time for a phone call was exponentially distributed with mean 3.0. The ratio of portables per cell / channels per cell was fixed at 2.5. All experiments were conducted on 8 processors, each processor containing 128 cells, for a total of 1024 cells. All experiments ran for 2.5×10^5 simulated seconds.

Speedup was calculated as the best distributed event rate [5] for a particular portables per cell, mobility and call interarrival time divided by the best sequential event rate for a particular portables per cell. The network environment experienced NFS traffic caused by other machines on the same subnet which caused variations in the execution time of the distributed runs by as much as 15%. Each sequential run required as much as 20 hours to complete. The standard deviation of the event rate was small for the sequential runs where the number of portables per cell was held constant, but mobility and call interarrival times were varied. Therefore, the best sequential run is the best among the runs where the number of portables per cell was held constant.

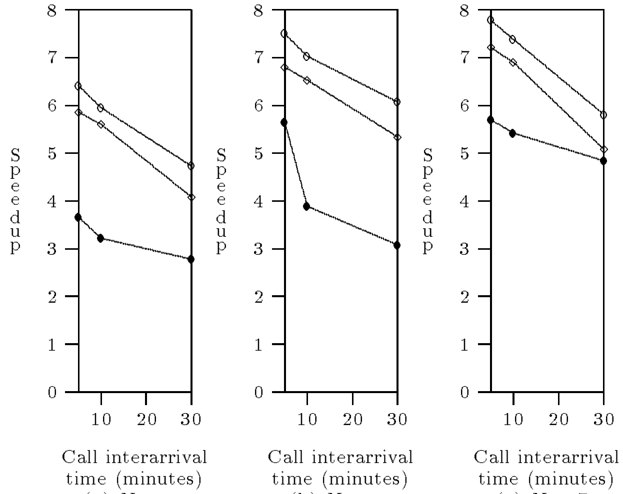
To give a fair measure of speedup, we used a sequential simulator that has the same calendar queue data structure for managing the pending set of events, but does not have the state saving, rollback and fossil collection overheads associated with the distributed simulator.

As shown by Figure 3, speedup increases when:

The number of portables per cell increases

By increasing the number of portables per cell, the message density [6] is increased. [6] shows that as the message density for exponentially distributed timestamps, increases so does speedup. Our results here reiterate those findings. The reason for the increase in performance is due to a reduction in rollbacks. By increasing the message density, simulated time advances at a slower rate because each processor has more work per unit of simulated time to perform. Meanwhile messages are still being sent the same distance into the future, resulting in fewer rollbacks, which leads better efficiency [7] and speedup.

Figure 3: Speedup versus number of portables per cell (N), mobility and call interarrival times. Call holding times are exponentially distributed with a 3 minute mean. Experiments performed using 8 processors. $\bullet$: Mobility = 1/(15 minutes), $\diamond$: Mobility = 1/(45 minutes), $\circ$: Mobility = 1/(75 minutes).



(a) $N = 25$ (b) $N = 50$ (c) $N = 75$

The call interarrival time decreases

Decreasing the call interarrival time has two effects on the performance of the PCS simulation. First, the amount of local work available to a processor is increased. Because calls are self-initiating, they result in messages being sent locally [8]. By decreasing the call interarrival time, calls occur more frequently in simulated time, which results in more local work available to a processor. Second, the frequency of synchronization is decreased. Portable movement is characterized by sending a remote message between two Cells. The sending of a remote message acts as a synchronization between two Cells. The more synchronization in an application, the less available parallelism, which limits the amount of speedup that can be obtained. Accordingly, by decreasing the call interarrival time, the number of calls between successive Portable moves is increased, which decreases the ratio of remote to local messages, resulting in a decrease in synchronization frequency. The net effect of increasing local work and decreasing synchronization frequency is a decrease in the probability of rollbacks, which leads to an increase in speedup.

The mobility decreases

[5] Event rate is defined as the total number of committed events divided by the wall clock time to complete the simulation.

[6] We define *message density* as total number of unprocessed events in the system divided by the number of entities or LPs. Our definition is consistent with the one presented in [6].

[7] Efficiency is defined as the events committed / events processed.

[8] We define a *local message* as a message where the sender and receiver are the same entity instance. A *remote message* has differing sender and receiver, although the sender and receiver could be located on the same physical processor.

Table 1: Comparison of average rollback length between PCS and PHOLD workload model.

Application	RB Length	Application	RB Length
PHOLD, N=25	1.8	PCS, N=25	7.5 - 9.0
PHOLD, N=50	2.2	PCS, N=50	7.5 - 10.5
PHOLD, N=75	2.4	PCS, N=75	6.5 - 13.0

Decreasing the mobility also results in a increase in local work and a decrease in synchronization frequency, which decreases rollback probability, leading to better speedup.

In performing our speedup experiments, we observed that the PCS simulation experienced long rollback lengths for all workloads, as shown in table 1. In previous simulations using the Time Warp mechanism on a shared memory multiprocessor, we observed much shorter rollback lengths (< 2). We believed the cause arises from the self-initiating nature of the PCS model. To determine the cause, we implemented a distributed version of the PHOLD model [6], which has been shown to have short rollback lengths on shared memory multiprocessors. Our implementation of PHOLD had 1024 LPs. Communications between entities was uniform and fully connected. We ran PHOLD with the same workloads as done in the PCS experiments, 25, 50 and 75 initial messages per LP. As shown in table 1, the rollbacks lengths for PHOLD were quite short, despite the large overhead for interprocessor communications. These findings suggest that the self-initiating nature of the PCS simulation causes longer rollback lengths. We base this conjecture on the infrequent sending of remote messages, which results in fewer synchronizations between LPs. Accordingly, when two LPs do communicate, they tend to be far apart in simulated time resulting in a long but relatively few rollbacks. PHOLD, on the other hand, synchronizes very often because of the large number of remote messages, resulting short but more rollbacks.

5 Conclusions & Future Work

We propose a discrete event model for simulating PCS networks on an object oriented, distributed implementation of Time Warp. The advantage of our approach is that we are able to simulate realistically sized PCS networks in a short amount of time, which gives us better insight into the behavior of these networks. In the worst case, our distributed simulation technique completes a simulation run in 16 minutes, whereas the sequential simulation completes in about 1 hour. However, in the best case, the distributed simulation completes in 2 hours whereas the sequential simulation completes in just under 16 hours.

Moreover, we observed long rollback lengths, which we attribute to the infrequent synchronizations caused by the self-initiating nature of the PCS model. This raises some interesting issues concerning how best to develop these simulations. Do self-initiating models, such as PCS, perform better than message-initiating models, like PHOLD? If so, under what conditions? Currently, we do not have the answer to these questions and feel that further investigation is warranted before any final conclusions can be reached.

Finally, there exist many fertile areas of study within the PCS application, such as handoff/handover and channel assignment, wireless packet switching, and location tracking.

References

[1] R. Brown. Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem. *Communications of the ACM*, 31(10):1220–1227, Oct. 1988.

[2] O. F. Bryan, Jr. Modsim II - an object-oriented simulation language for sequential and parallel processors. In *1989 Winter Simulation Conference Proceedings*, pages 122–127, 1989.

[3] Cox, D. C. Personal Communications – A Viewpoint. *IEEE Communications Magazine*, 128(11), 1990.

[4] England, Paul. The pTask System for High Performance Distributed Computing. Submitted for publication, 1993.

[5] R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October 1990.

[6] R. M. Fujimoto. Performance of Time Warp under synthetic workloads. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 23–28, 1990.

[7] Kuek, S.S., and Wong, W.C. Ordered Dynamic Channel Assignment Scheme with Reassignment in Highway Microcells. *IEEE Trans. Veh. Technol.*, 41(3):271–277, 1992.

[8] Lin, Y.-B., and Chen, W. Impact of Busy Lines and Mobility on Cell Blocking in a PCS Network. Submitted for publication, 1993.

[9] Lin, Y.-B., and Mak, V.K. On Simulating a Large-Scale Personal Communications Services Network. To appear in ACM TOMACS, 1993.

[10] David M. Nicol. Performance bounds on parallel self-initiating discrete-event simulations. *ACM TOMACS*, 1(1):24–50, January 1991.

[11] Tekinary, S., and Jabbari, B. A Measurement Based Prioritization Scheme for Handovers in Cellular and Microcellular Networks. *IEEE J. Sel. Areas Comm.*, October 1992.

[12] Wong, W. C. Packet Reservation Multiple Access in a Metropolitan Microcellular Radio Environment. *IEEE J. Sel. Areas Comm.*, 11(6):918–925, 1993.

[13] Zhang, M., and Yum, T.-S. Comparisons of Channel-Assignment Strategies in Cellular Mobile Telephone Systems. *IEEE Trans. Veh. Technol.*, 38(4):211–215, 1989.