

## 6

### A Comparative Analysis of Virtual Versus Physical Process-Migration Strategies for Distributed Modeling and Simulation of Mobile Computing Networks

K.Han

Department of Computer Science, Brown University, Providence, RI 02912, kwh@cs.brown.edu

S. Ghosh

Department of Computer Science & Engineering, Arizona State University, Tempe, AZ 85287, Tel: 602-965-1760, Fax 602-965-2751, sumit.ghosh@asu.edu

#### Abstract

This paper focuses on the high level principles that underlie the distributed modeling and accurate simulation of mobile computing networks on a parallel processing testbed. The testbed consists of a network of workstations configured as a loosely-coupled parallel processor and it closely resembles reality. A key issue is the representation of the stationary and mobile entities of the mobile computing network through concurrent and interacting processes in the testbed. The nature of the representation will influence the accuracy and performance of the simulation. This paper first reviews a process representation technique that has been proposed in the literature for modeling railway networks and then analyzes its limitations. This strategy is referred to as Virtual Process Migration (VPM). The paper then proposes a new strategy, termed Physical Process Migration (PPM), that aims to address the limitations of VPM. It details the software techniques underlying both approaches, describes their implementations on a realistic testbed, and then contrasts their performance under different representative scenarios. While VPM is capable of modeling modest to large-scale mobile computing networks on a testbed consisting of a few processors, the number of processors of the testbed in PPM must correspond to the number of stationary and mobile entities of the mobile computing network size that is being modeled. Analysis of the simulation results reveals that both VPM and PPM are highly effective and very useful strategies under different circumstances. For a given number of mobile and stationary entities, simulation under PPM is fast when every mobile entity requires significant computation. On the other hand, VPM exhibits superior performance relative to PPM when the number of data elements exchanged by each mobile entity at each hop is significantly high.

**Keywords:** Mobile computing, distributed algorithms, physical process migration, process migration, virtual migration, message passing, IVHS, community health care network, patient medical record integration, decision-making, distributed decision-making, scheduling

### *1. Introduction*

The large bulk and substantial power requirements of high-speed computers of the past forced many systems to adopt the traditional, centralized approach wherein sequential algorithms executed on centralized computers. The results were inefficiency and high costs, caused by slow sequential processing, the need to propagate data and information from geographically-dispersed sites to the central computer, and relay the decisions back to the sites. Examples include the Centralized Traffic Control (CTC) (Compton 1993) and the ATCS (Coll, 1990), both for railway networks. In these systems, the destination of every train is known a priori by the dispatcher – a uniprocessor computer. Additionally, a dispatcher receives, at regular intervals of time, the current position, speed, and movement of every train and the statuses i.e., whether occupied or empty, of every track in the system. The dispatcher utilizes uniprocessor optimization techniques to analyze the data sequentially, and utilizing a cost function, it computes the subsequent sub-route that every train must execute. The dispatcher then arranges for the appropriate switch settings corresponding to the signals and tracks. Other examples of the limitation of the traditional approach include the complete absence of many high-tech facilities in the past that are increasingly becoming commonplace in today's world. These include the portable, radio linked FAX and electronic mail units as well as en-route automobile drivers accessing highway conditions and route guidance information.

Significant improvements in microprocessor technology, notably in their computing ability, speed, miniaturization, low power consumption coupled with corresponding advances in lost cost communications are increasingly opening doors to new mobile computing applications and changing the faces of many traditional systems. This drive is greatly accelerated by the design and use of innovative distributed algorithms which are essentially based on the underlying principle of multiple, concurrent, coordinating and communicating processes. The DARYN (Iyer, 1991) (Iyer, 1995) approach distributes the overall task of scheduling among all stations and trains and achieves high efficiency, robustness, and scalability. The new distributed command and control algorithm for the battlefield (Lee, 1995), where the traditional information gathering and decision generating central headquarters is replaced by semi-autonomous fighting vehicles, yields superior performance over the traditional scheme under realistic battle conditions. In reality, the new M1-A2 tanks are outfitted with powerful workstations and advanced communications gear to facilitate sophisticated target acquisition, decision-making, maneuvering, and firing. Other futuristic services that are currently on the drawing board today include economical access to patient medical records by mobile physicians and distributed approach for intelligent vehicle highway system.

In general, a mobile computing network may be characterized as follows. It consists of multiple mobile agents that require access to (i) information generated at multiple geographically dispersed sites and (ii) computing engines to execute their decisions. It may include one or more stationary agents that perform information acquisition and propagation to the mobile agents. While a static interconnection network may link the stationary agents, a dynamic interconnection network will connect the mobile agents to the stationary nodes. The mobile nodes may connect to specific stationary nodes asynchronously, i.e. at irregular intervals of time, to acquire information and following completion, they will disconnect. The use of the term, connection, in this context refers to the transport layer in the ISO-OSI terminology (bertsekas, 1992). The underlying physical layer, however, is at liberty to utilize either wired or wireless transmission. The mobile and stationary agents are located at geographically dispersed sites. While both stationary and mobile nodes may have computing and communication needs, the relative weights and frequency are problem specific. In addition, the system must be designed to accommodate evolutionary growth. That is, the system must continue to function and deliver relatively undiminished performance as the cumulative number of stationary and mobile entities increases with time.

Koch, Krombholz, and Theel (1993) introduce the concept of mobile computing, define its key characteristics, and present its scope of application. Imielinski and Badrinath (1993) identify the challenges in data management, arising from the issues of mobile hosts, wireless broadcasting, and frequent disconnections and discuss the necessary structure of the distributed algorithms. Forman and Zahorjan (1994) stress network reliability, greater autonomy for the mobile agents, asynchronous operations, and flexible consistency-semantics towards successful mobile computing networks. They also raise issues related to frequent and abrupt disconnections, high bandwidth variability, security, and portability. Satyanarayanan (1993) notes important limitations of mobile systems including poor resources, vulnerability to catastrophic failures, and the need to operate under a broad range of network conditions. He proposes the Coda file system which realizes user-transparent mobility of agents by permitting autonomous operations while disconnected and transparent reintegration changes upon reconnection. He also reports future plans to support very low bandwidth or intermittent connections in Coda. Duchamp, Feiner, and Maguire (1991) report on their early efforts in systems software development for wireless mobile computing. They focus primarily on the issue of software design to permit the movement of mobile hardware platforms without interrupting the high-level software.

In contrast to the reported research efforts (Forman, 1994)(Satyanarayana, 1993) (Koch, 1993)(Imielinski, 1993)(Duchamp, 1991), this paper is concerned with the issues of modeling and simulation of real mobile computing networks, a key phase that must precede the actual development of such networks. Specifically, this paper focuses on the general principles underlying the modeling and simulation of a specific class of solutions (Iyer, 1991)(Iyer, 1995)(Lee, 1995) to real-world mobile computing network problems. The solutions utilize asynchronous, distributed algorithms. The fundamental philosophy here is that the overall decision-making task is intelligently distributed among the mobile and stationary entities. This will maximize local

computations, minimize communications, and yield a high throughput approach that is efficient, robust, and scalable. As the system experiences evolutionary growth, the approach must continue to function and yield relatively undiminished performance. Every entity is viewed as an asynchronous and autonomous process with well-defined computational and communications needs. While some processes are "stationary," others are "mobile" within the network.

The key characteristics of mobile computing networks differ from those for load balancing. For application problems of interest to this study, the primary objective is to model and efficiently simulate the components of the network, not necessarily to equitably distribute tasks to computing processors. The exact pattern of migration of the mobile agents, in truth, is dictated by the nature of the application and the actual input data. The migration pattern is further complicated by the fact that every mobile process is autonomous, i.e. every mobile entity determines its own migration pattern based on its unique behavior, input stimulus, and dynamic interactions with the stationary entities. Every mobile and stationary entity is characterized by unique computation and communication needs. Furthermore, the nature of the migration is asynchronous, i.e. it is initiated at irregular intervals of time and may not be known a priori. Finally, in many real-world mobile computing networks, the number of mobile and stationary agents, in general, will be large which, in turn, necessitates a distributed, scalable approach.

The remainder of the paper is organized as follows. Section 2 introduces two competing process migration strategies that may be used to model and simulate mobile networks. Section 3 details their underlying software techniques while section 4 presents the details of implementation on a parallel processor testbed. Section 5 first presents the results obtained from simulating the two approaches for a representative network under realistic input conditions and then a comparative analysis. Finally, section 6 concludes the paper.

## *2. Virtual and Physical Process Migration Strategies for Mobile Computing Networks*

This paper focuses on a class of real-world, large-scale, mobile computing networks with the following characteristics: (1) The number of stationary entities is relatively modest but the number of migrating entities is large, ranging from 10s to 100s, (2) The system is likely to grow in size with time requiring that the underlying approach be scalable, (3) While the stationary entities are geographically dispersed, the mobile entities are autonomous implying that their migration patterns are unique to every mobile agent are unknown a priori, (4) While the stationary agents are permitted to communicate directly between themselves through a static interconnection network, the mobile agents are assumed not to require direct communication between themselves for the following reasons. First, given that the number of mobile agents is large, facilities to provide direct communication between any two agents are likely to incur large overhead. This may also adversely impact scalability. Second, the underlying distributed algorithms are intelligently designed so that the stationary nodes perform the function of coordinating information between the mobile agents, when necessary. The target class

of application problems include railway networks, community health care networks, battlefield networks, and intelligent vehicle highway systems.

Thus, the computer model of a mobile computing network will consist of stationary and migrating processes executing on computing engines and mechanisms to facilitate stationary-stationary agent and mobile-stationary agent communications. Every process owns its own thread of control and is thus autonomous and asynchronous relative to other processes in the system. The capabilities of the processes are defined by the nature of the system. The stationary processes acquire necessary information from other stationary processes and mobile processes which is subsequently downloaded and utilized by appropriate mobile processes. While the static network interconnecting the stationary processes is permanent, the mobile processes connect and disconnect dynamically and asynchronously, i.e. at irregular intervals of time, with appropriate stationary processes. A migration occurs when a mobile process,  $M_i$ , chooses to disassociate itself from the stationary process,  $S_j$ , and associate itself with the stationary process,  $S_k$ , for all legitimate values of  $j$  and  $k$ .

In a real system, every stationary and mobile agent is provided with its own computing engine and facilities to initiate communication with other agents. It is therefore logical to assume that in a simulation of a mobile computing network, every stationary and mobile process will have access to its own computing engine. However, many parallel processing testbeds, including the one utilized in this paper, are likely to have far fewer available processors than the total number of stationary and mobile agents. This results in two principal strategies for representing mobile entities through processes in the testbed. They are termed virtual and physical process migration strategies and are detailed subsequently.

### *2.1 Virtual Process Migration Strategy*

The obvious logical choice is to represent the relatively modest number of stationary agents as actual processes, assign them to the processors of the parallel processing testbed on a one-on-one basis, and represent the mobile agents through virtual processes. A stationary node represents an agent located at a specific geographic position. A virtual process migrates between processors, when necessary, and its computational needs are executed by the host processor underlying the stationary node where it may happen to be located at that instant of time. By definition, a virtual process is not permanently associated with any processor. From time to time, it is associated with a processor, corresponding to a stationary node, that executes its computing needs and temporarily assigns it the status of an actual process. This strategy is termed Virtual Process Migration (VPM) and has been utilized in (Iyer, 1995). In VPM, the number of processors utilized equal the number of stationary entities. The processors are interconnected in the same topology as the stationary entities, through software protocols that are initiated at initialization time and remain unchanged throughout the simulation.

A virtual process in VPM is similar to a "thread" of an operating system. However, unlike a "thread" that

contains the code, stack, stack pointer and the program counter, a virtual process only contains the essential parameters required for its execution. The exact parameters are defined by the application program. As an example, in the modeling and simulation of the intelligent vehicle highway system, the parameters for the mobile automobiles may include the vehicle license plate, model, manufacturer, current speed, desired speed, location, heading, origin, and destination. When a mobile entity is located at a stationary node, it "appears" at the node, i.e. it is manifested as an actual process and its computing needs are executed by the host processor. Utilizing relevant information contained at the stationary node and within itself, the mobile entity determines its subsequent course of action which may include the decision to migrate to a different stationary node. Then, the simulation migrates the corresponding virtual process with all of its parameters to the appropriate stationary node where the mobile entity again "reappears." Thus, the behavior of a mobile agent is self-contained and is neither visible to the stationary node nor to other virtual processes that may be temporarily co-resident at the same stationary node. Also, at any given time, one or more virtual processes may be resident at a stationary node and compete for the computation and communication resources. Thus, a scheduler may be utilized to assign slots of computing and communication facilities to the processes. Communication of information between the stationary process and a virtual process is achieved simply through buffer copying.

## *2.2 Physical Process Migration Strategy*

Despite its utilization in (Iyer, 1995), VPM incurs important limitations. As the number of mobile entities increases, the competition for the host processors' computing and communication resources is likely to become acute thereby slowing down the simulation significantly. To address this limitation, this paper proposes a competing approach, Physical Process Migration (PPM). In PPM, every process – stationary or mobile, is allocated a unique processor. The allocation is engaged at the instant the process is initiated into the system and is disengaged when the process terminates. When a mobile process desires to communicate with a stationary process at runtime, first a communication protocol is dynamically established between the underlying processors and then information exchange is initiated. Thereafter, when the mobile process desires to interact with a different stationary process, the old protocol is first disconnected and a new connection is established. A mobile process is allowed to maintain a connection with a single stationary process at any time. Thus, the PPM strategy is a more accurate model of reality. The static interconnection network between the stationary processes remains identical to that for the VPM. Clearly, the computational need of every mobile process is executed by its underlying processor, and where the computational needs of the mobile agents are high, there is the potential for higher efficiency and throughput relative to VPM. Unlike VPM, a mobile process may easily migrate from stationary node A to stationary node Z in PPM where a direct connection from A to Z may be lacking. PPM's principal advantage is in the use of one processor per process. Unfortunately, this also results in a weakness in the context of the limitations of today's testbed technology. Since testbeds with 1000s of processors are not yet ubiquitous, simulation under PPM is limited to modest-sized mobile computing networks.

PPM also inherits the limitation of high overhead for mobile-stationary process communication which includes explicit message communication following the dynamic establishment of a communications protocol.

### *3. Software Techniques Underlying the Process Migration Strategies*

The static network interconnecting the stationary processes in both VPM and PPM is established during initialization of the simulation. As indicated earlier, every stationary process is assigned a distinct processor or workstation, termed node. During execution, first, a process opens a unique external input file, utilizing the identifier of the underlying workstation. This file contains the node's operating characteristics that includes its connectivity to other stationary processes. Second, the process starts to build the point-to-point connections, one at a time, utilizing the Berkeley socket protocols. When establishing a point-to-point connection between two processors, the initiator process executes a "connect" while the corresponding receiving process executes an "accept." Every connection is half-duplex, implying directed edges in the network, and there may be multiple, overlapping cycles in the network. Furthermore, "connect" requires the receiving process identifier as an argument and it is non-blocking while "accept" is blocking and is designed to receive any "connect," i.e. from any processor. This threatens the network initialization with the possibility of deadlock and, to counter it, the following algorithm is utilized. The underlying nodes possess unique identifiers. When a stationary process at a node (identifier X) requires connection with a stationary process at a different node (identifier Y,  $Y > X$ ), X always executes a "connect" while Y will execute an "accept." Upon completion of the network configuration in its memory, every node initiates execution of the stationary and mobile processes which differs for the VPM and PPM strategies.

#### *3.1 Software Techniques Underlying VPM*

In VPM, every mobile entity is represented through a set of parameters which are organized into a structure. The size of the structure is a function of the application and may be dynamic. For a comparative study of the performance of VPM and PPM, the following fields are assumed for every mobile entity structure. The first field is the identifier of the entity, the second reflects its computational need, the third encapsulates the remaining number of messages that this entity must exchange with the host stationary process, and the fourth field stores the remaining number of hops in this entity's migration pattern.

The computational load of a mobile unit is represented by an integer, ranging from 100 to 10,000,000, and it constitutes the index of a simple "for loop." That is, the number of iterations in the "for loop" equals the load value and the execution time of the iterative loop emulates the actual computational time. In VPM, at any given time instant, one or more mobile entities may be co-resident at a stationary node, competing for the single thread of control. To ensure that every mobile entity receives its fair share of the thread of control,

the simulation proposes to use "time slicing," wherein every virtual process voluntarily gives up control after executing the loop for every 100 iterations.

Upon arrival at a node, a mobile entity is remanifested as an actual process and is enqueued in the scheduler's list. The scheduler allocates a time slot and executes the body of the mobile entity in the time slot. Its function is expressed subsequently, in pseudo-code.

During migration, the parameters of the mobile entity are encapsulated in a message. The address and size of the message are passed to the operating system through the "write" system call, which then writes it to the appropriate outgoing socket and executes the transfer.

At the receiving end, the node polls for the arrival of the mobile entity using the "select" system call with 0 timeout. Select maintains the ability to monitor multiple socket connections from within a single function call. When the message arrives, the node remanifests it as an actual process and enqueues it in the scheduler's list.

The scheduler implements round-robin scheduling of the stationary process and one or more mobile processes that may be co-resident in the host processor. When it is scheduled for execution, a mobile entity is first dequeued, then executed, and then either requeued into the scheduler's list or marked for migration to a different node.

### *3.2 Software Techniques Underlying PPM*

#### **Mobile entities**

At initialization, every mobile entity is associated with a processing node which is responsible for executing the iterative loop, message transfer, and migration routines. A message transfers with respect to a stationary node requires the presence of a network protocol.

In the event of migration, first the old connection between the mobile process and a stationary process, if any, must be terminated. Normally, at either end of the connection, a "close" system call is executed. However, if the execution of the system calls by the two processors are not synchronized, the connection may be closed only partially and a SIGPIPE signal is generated when a process attempts to write to it. To avoid this undesirable side effect, cooperation is required at both ends. In this paper, when a connection is slated to be terminated, first a termination command is sent from the mobile node to the stationary node. Second, the mobile node awaits an acknowledge from the stationary node, following which both processes execute the "close" system call. Next, a new connection is established with the target stationary node.

#### **Stationary entities**



The behavior of a stationary node includes three functions – (i) accept connection from the mobile entities, (ii) exchange data with mobile entities, and (iii) accept termination request from a mobile entity.

Given that a connection from a mobile node to a stationary node is initiated asynchronously and dynamically by the mobile entity, every stationary node must necessarily provide an entry point where the mobile node can initiate a connection. To realize the entry point, every stationary node binds a special socket and periodically listens to it through a select system call to determine whether a mobile node desires connection with it. If affirmative, the stationary node executes an accept system call, reads and stores the identifier, and initiates the establishment of a connection with the mobile node. Upon connection, two kinds of messages are communicated – data elements and disconnection request. Data elements are exchanged between the mobile and stationary nodes and, in this paper, the data is dummy and simply discarded. When the mobile node intends to disconnect, it propagates a disconnection message to the stationary node. In turn, the stationary node will propagate an acknowledgement of the disconnection and execute the “close” system call to disconnect.

#### *4. Implementation Issues*

The VPM and PPM strategies are implemented on a testbed of 65+ SUN Sparc 10/40 workstations that are configured as a loosely-coupled parallel processor. While each workstation is outfitted with 32 Mbytes of memory and executes Solaris 2.3 operating system, they are interconnected by a 10 Mbit/sec ethernet. In addition, the code design permits execution under both SUN OS 4.1.3 and the freely available Linux operating systems (GNU). The code is written in C++, is approximately 2000 lines in length, and is compiled by public-domain GNU g++ compiler. While the code executes in the background while user execute programs on the consoles, the data presented here is obtained from simulations that are run late at night when network load is minimal.

#### *5. Simulation Results and Performance Analysis*

For a comparative analysis of their performance, both VPM and PPM strategies are modeled and simulated on an parallel processing testbed. The testbed closely resembles reality and a number of experiments are designed and executed. Corresponding to an actual mobile computing network where the key parameters include the size of the static network, i.e. the number of stationary nodes, the interconnection topology of the static network, the number of mobile entities, the computational load of the mobile agents, the number of messages exchanged between the mobile and stationary entities at each hop, and the migration pattern of the mobile entities, the simulation represents these parameters through independent variables. The key measure of performance is the maximum over the wall clock times required by all processors in the testbed. In the experiments, the number of entities chosen reflect the fact that the testbed is limited to 65 workstations. The number of stationary nodes

range from 5 to 10, the static interconnection topology is assumed to be fully connected, and the number of mobile agents ranges from 5 to 50. The computational load of a mobile unit is represented by an integer, ranging from 100 to 10,000,000, and it constitutes the index of a simple "for loop." That is, the number of iterations in the "for loop" equals the load value and the execution time of the iterative loop emulates the actual computational time. The number of data elements exchanged between a mobile and stationary agent is assumed to range from 1 to 100, where each data element is 128 bytes long dummy. Every mobile agent's migration pattern is stochastic, unique, and asynchronous, i.e. the mobile entity may migrate at irregular intervals of time. The only constraint imposed on the mobile agents is that an agent will not immediately reconnect to the stationary node to which it was connected most recently. In the simulation, unless otherwise specified, every mobile agent connects and disconnects with the stationary nodes a total of 1000 times.

The results presented here reflect a total of over 200 simulation runs, each requiring an average 1000 seconds of wall clock time, 65 concurrently executing workstations, and several Mbytes of data collected from the simulation. When a mobile entity connects with a stationary entity, it performs computations, defined by the load value, and then exchanges data with the stationary process. The simulation terminates when every mobile entity has completed the specified number of connections and disconnections. The measured simulation time includes the time required for establishing the software protocol connection, the computation time, time for exchange of data, and disconnection time.

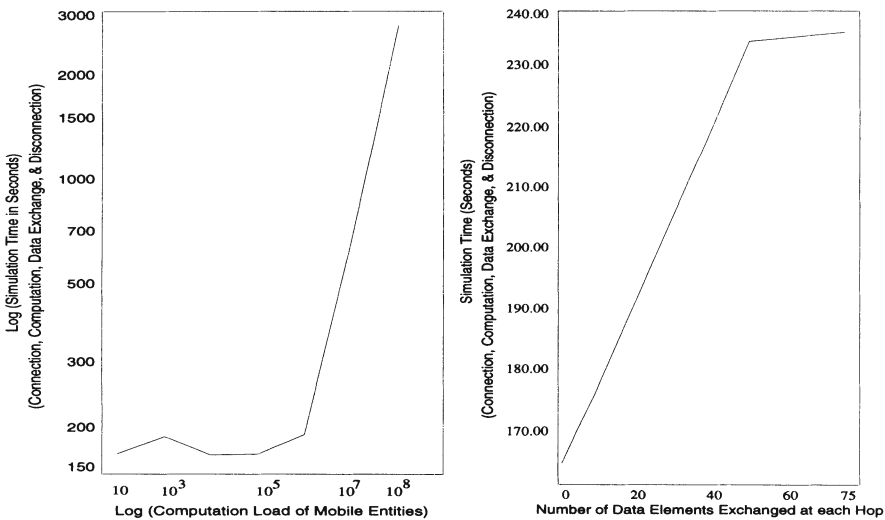


Figure 1: PPM Simulation Time as a Function of (a) Computational Load of Mobile Entities, (b) Number of Data Elements Exchanged at each hop.

Figure 1a presents a log-log plot of the PPM simulation time as a function of the computational load of the mobile agents. The number of stationary and mobile nodes are each set to 5 and the number of data elements exchanged at each hop is 10. In the graph, the simulation time remains relatively uniform up to a load of 1,000,000 and then increases sharply. Given that the number of messages, namely 10, is low, and that each mobile agent connects and disconnects a total of 1000 times, the connection and disconnection times dominate the overall simulation time for loads below 1,000,000. For loads beyond 1,000,000, the computation time emerges as the dominating component of the overall simulation time. Figure 1b presents the variation of the simulation time as a function of the number of data elements exchanged at each hop. The number of stationary and mobile entities are both set at 5 and the computational load is 100,000. The graph reveals that data exchange time dominates the overall simulation time and that the latter increases linearly with increasing number of data elements.

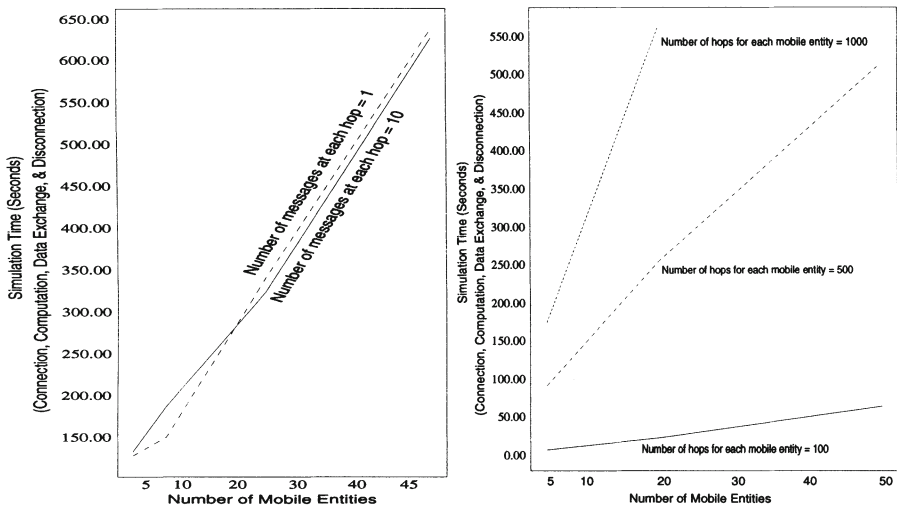


Figure 2: PPM Simulation Time as a Function of the Number of Mobile Entities in the System, (a) 10 Stationary Nodes, Computational Load 100,000, and Varying Number of Data Elements Exchanged, (b) 5 Stationary Nodes, Computational Load 100,000, and Varying Number of Hops per each Mobile Entity.

Figure 2a presents the simulation time as a function of the number of mobile entities in the system. The number of stationary nodes and computation load are set to 10 and 100,000 respectively. Two graphs, corresponding to 1 and 10 data elements exchanged between a mobile entity and a stationary entity at each hop, are shown in Figure 2a. As the number of mobile entities are increased, the simulation time increases linearly. However, the nature of the graph does not appear to be appreciably affected by the number of data elements exchanged, implying that for the modest load values of 100,000 and anywhere from 1 to 10 data elements exchanged per

hop, the connection and disconnection times dominate the overall simulation time. The graphs in Figure 2b further corroborate this finding. The number of messages exchanged at each hop is 1. For increasing number of hops per every mobile entity ranging from 100 to 500 to 1000, the number of connections and disconnections increase and this immediately impacts the simulation times as evident through the increasing slopes of the corresponding graphs.

The graphs in Figures 1 and 2 reveal that each of the components of the overall simulation time – computation time, connection and disconnection time, and time for data exchange, may dominate the overall simulation time when the corresponding parameter is allowed to increase while the other parameters are held constant. Also, their individual influences on the overall simulation time is approximately linear.

## 6. Conclusion

This paper has proposed a new strategy, termed Physical Process Migration (PPM), that aimed to address the limitations of VPM. It detailed the software techniques underlying both approaches, described their implementations on a realistic testbed, and then contrasted their performance under different representative scenarios. While VPM is capable of modeling modest to large-scale mobile computing networks on a testbed consisting of a few processors, the number of processors of the testbed in PPM must correspond to the number of stationary and mobile entities of the mobile computing network size that is being modeled. Analysis of the simulation results has revealed that both VPM and PPM are highly effective and very useful strategies under different circumstances. For a given number of mobile and stationary entities, simulation under PPM is fast when every mobile entity requires significant computation. On the other hand, VPM exhibits superior performance relative to PPM when the number of data elements exchanged by each mobile entity at each hop is significantly high.

## 7. References

- Artsy, Y. and Finkel, R. (1989) Designing a Process Migration Facility - The Charlotte Experience. *COMPUTER*, 22(9), 47-56.
- Bertsekas, D. and Gallager, R. (1992). *Data Networks*. Prentice Hall, New Jersey.
- Britannica Software. (1993) *Compton Multimedia Encyclopedia on CD-ROM*.
- Chen, M.S. and Shin, K.G. (1990) Subcube Allocation and Task Migration in Hypercube Multiprocessors. *IEEE Transactions On Computers*, 39(9), 1146-1155.
- Coll, D. C., Sheikh, A. U., Ayers, R. G., and Bailey, J. H. (1990) The communications system architecture of the North American Advanced Train Control System. *IEEE Transactions on Vehicular Technology*, 39(3), August, 244-255.
- Duchamp, D., Feiner, S.K., and Maguire, G.Q.. (1991) Software Technology for Wireless Mobile Computing. *IEEE Network*, 5(6), November, 12-18.
- Forman, G.H. and Zahorjan, J. (1994) The Challenges of Mobile Computing. *IEEE Computer*, 27(4), April, 38-47.
- Gait, J. (1990) Scheduling and Process Migration in Partitioned Multiprocessors. *Journal Of Parallel and Distributed Computing*, 8(3), 274-279.

- Glazer, D.W. and Tropper, C. (1993) On Process Migration and Load Balancing in Time Warp. *IEEE Transactions On Parallel and Distributed Systems*, 4(3), 318-327.
- GNU, Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
- Hac, A. (1989) A Distributed Algorithm for Performance Improvement Through File Replication, File Migration, and Process Migration. *IEEE Transactions on Software Engineering*, 15(11), 1459-1470.
- Imielinski, T. and Badrinath, B.R. (1993) Data Management for Mobile Computing," *SIGMOD Record*, 22(1), March, 34-39.
- Iyer, R. and Ghosh, S. (1991) DARYN, A Distributed Decision-Making Algorithm for Railway Networks: Modeling and Simulation. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, October 13-16, 1991, University of Virginia in Charlottesville, VA 22903, 269-274.
- Iyer, R.V. and Ghosh, S. (1995) DARYN, A Distributed Decision-Making Algorithm for Railway Networks: Modeling and Simulation. *IEEE Transactions on Vehicular Technology*, 44(1), February.
- Koch, H., Krombholz, H. and Theel, O. (1993) A Brief Introduction to the World of Mobile Computing, Technical Report THD-BS-1993-03, Computer Science Department, University of Darmstadt, Germany.
- Lee, T. and Ghosh, S. (1995) A Novel Approach to Asynchronous, Decentralized Decision-Making in Military Command and Control. *Proceedings of the Second International Symposium on Autonomous Decentralized Systems*, Phoenix, AZ, April 25-27.
- Lin, W-M. and Yang, B. (1995) Load balancing technique for parallel search with statistical model. *Proceedings of the Fourteenth Annual IEEE International Phoenix Conference on Computers and Communications*, March 28-31, Phoenix, Arizona.
- Milutinovic, V.M., Houstis, C.E., and Crnkovic, J.J. (1988) A Simulation Study of 2 Distributed Task Allocation Procedures. *IEEE Transactions On Software Engineering*, 14(1), 54-61.
- Satyanarayanan, M. (1993) Mobile Computing. *COMPUTER*, 26(9), 81-82.
- Suen, T. T. Y. and Wong, J. S. K. (1992) Efficient Task Migration Algorithm for Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems*. 3(4), 488-499.

## *8. Biography*

Kwun Han is a senior in the Computer Science Department at Brown University and his research interests are in networking, artificial intelligence, and distributed algorithms.

Sumit Ghosh is an associate professor in the Computer Science department at Arizona State University. He received the M.S., and Ph.D. degrees from the Computer Systems Laboratory of the electrical engineering department at Stanford University in 1981 and 1984. He served as a Principal Investigator - Member of Technical Staff at Bell Laboratories Research, Holmdel, New Jersey, up to Dec 1988, then as an assistant professor at Brown University, Providence, Rhode Island, up to August 1995. His research within the Networking and Distributed Algorithms laboratory, focuses at the application layer of networking, modeling and simulation of ATM networks, approximate reasoning for self-healing broadband-ISDN network control, stability of asynchronous distributed decision-making algorithms, mathematical representation and synthesis of distributed algorithms, parallel execution of VHDL models, new timing semantics for VHDL language, distributed architecture for integration of patient medical records, and distributed visualization.