Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

DYNAMIC ENTITY DISTRIBUTION IN PARALLEL DISCRETE EVENT SIMULATION

Michael Slavik
Imad Mahgoub
Ahmed Badi

Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, Florida 33431

ABSTRACT

Event based simulations are an important scientific application in many fields. With the rise of cluster computing, distributed event simulation optimization becomes an essential research topic. This paper identifies cross-node event queues as a major source of slow down in practical parallel event simulations and proposes dynamically moving entities between nodes to minimize such remote event queues. The problem statement is formalized and an algorithm based on an approximation algorithm for the Capacitated Minimum K-Cut Problem is proposed. The algorithm is simulated and results are presented that show its effectiveness. For simulations with reasonably regular structural relationships between entities, reductions of remote entity queues from 80 to 90 % are demonstrated.

1 INTRODUCTION

Cluster computing is quickly becoming the primary supercomputing method, due to its great and improving cost effectiveness. Nodes in the cluster are typically common computers connected via a network and may even be running different platforms. Cluster computing thus present challenging problems for software designers: applications must be effectively parallelized to run on multiple machines with separate memory and instruction streams, application execution must be synchronized by the machines communicating with each other over a relatively slow network (typically an IP network), and application software may need to run on multiple platforms to get full utilization from the network. Because of these challenges, tools are needed to ease cluster-based application development.

One of the common applications to run in a supercomputing environment is discrete event simulation (DES). DES is used in a wide variety of applications such as manufacturing models, computer network models, and artificial intelligence research. Because of this, there has been interest in creating cluster-based discrete event simulators called parallel discrete event simulators (PDES). Amdahl's Law (Amdahl 1967) bounds the effectiveness of parallelization in applications with sequential components, and DES is naturally highly sequential; the semantics of DES in fact demand events are executed in sequential order. Thus DES is an extremely difficult problem to distribute across a cluster, and is very sensitive to dependencies among simulation entities. Event simulations must be written specifically with cluster applications in mind, or else it is not uncommon to see the total simulation time *increase* as a result of executing the simulation on a cluster. In this study, we examine the problem and propose a method of avoiding this situation.

1.1 Terminology

A quick note on terminology. Throughout this paper we will use the terms **node**, **entity**, and **event**, for which it will be useful for all readers to have a common understanding. A node refers to a single machine on the cluster. An entity is an autonomous simulation object, consisting of its own memory and event handling methods. An event is a method invoked on an entity at a particular simulation time. An entity invoking an event on another entity is referred to as the entity queuing an event to the other entity and such an occurrence is called an **event queue**.

2 PROBLEM STATEMENT

A distributed event simulation instance consists of a set of entities E residing on a set of nodes N. These entities have memory associated with them and contain references to each other. Using these references, entities queue events to other entities in the simulation. Running on a single machine, these associations among entities cause no special problem. Running on a cluster, entities queuing events to entities resident on different nodes limits the extent to which the simulation can be executed in parallel. The best case event simulation for a distributed environment is when the entities naturally separate into $|N|$ groups that do not queue

978-1-4244-2708-6/08/$25.00 ©2008 IEEE

events between each other, and each node contains all of the entities in one of the groups. At the other end of the spectrum, the entities may contain only references to entities residing on different nodes and all event queues cross node boundaries. Such a simulation is guaranteed to run slower on the cluster as a whole than on any single machine in the cluster. An entity queuing an event to an entity on the same node is called a *local queue*, while event queues across node boundaries are called *remote queues*. Maximizing the number of local queues maximizes the benefits of running the simulation on a cluster.

Clearly the distribution of entities in the cluster can have a large impact on the performance of the simulation. If entities that queue events to each other often are located on different nodes, it will increase the number of remote queues and slow down the simulation considerably. Previous PDESs such as DSIM (Chen and Szymanski 2005) provide simulation APIs to coordinate the placement of entities at initialization time so the programmer can instruct the simulator on the optimal distribution. This works well for simple simulations, but has a number of drawbacks. Existing simulations designed to run on a single machine must be modified to place entities optimally during initialization, and it may not be clear what the optimal distribution is. In some simulations it may be impossible to know up front what the best way to place the entities is, or the optimal distribution of entities may change over time. In such cases, there is a need for a simulator that can dynamically redistribute entities during runtime.

2.1 Related Work

To the knowledge of the authors, no PDESs currently exist that are capable of moving entities between nodes. Many PDESs have been implemented, such as DSIM (Chen and Szymanski 2005), Parallel NS-2, SPADES (Riley and Riley 2003), and PARASOL (Pasquini and Rego 1999), but all of them rely on static entity distribution. One reason for this is probably that knowing *how* to move an entity, even if it is clear one needs to be moved, is not a simple task. Typically an entity at the simulator implementation level is a structure or object in C or C++ that may contain pointers to other objects or variables that are included in the memory associated with the entity. To move that entity, all that data has to be rounded up and transported to another node; not a straightforward task with bare C/C++. Because of this limitation, there has similarly been no research we are aware of into algorithms for moving entities between nodes.

2.2 Direct Model

JiST (Java in Simulation Time) (Barr 2004), a sophisticated Java-based DES recently extended by our research team to execute in a cluster environment, drastically simplifies the task of dynamically moving entities. Java natively supports reflection, allowing developers to programmatically inspect object structure during runtime. Because of this, data members of entity objects can be easily examined, serialized, and transmitted to another node.

The ability to transport entities between nodes then raises the larger question: when should an entity be moved from one node to another? How do you determine the optimal distribution of entities? The direct formalization of this problem as follows. Let $p : E \times E \rightarrow \Re$ be the probability an entity queues an event to another entity, so

$$\forall i \in E : \sum_{j \in E} p(i,j) = 1. \tag{1}$$

Let $r : E \rightarrow \Re$ be the rate at which an entity generates events. Let $S : E \rightarrow \Re$ be the size of an entity, $U : N \rightarrow \Re$ be the capacity of a node, and N_i be the set of entities resident on node i. Let $\delta : E \rightarrow N$ be the mapping of entities to nodes. Let c_m, c_l, and c_r be the cost of moving an entity, a local queue, and a remote queue, respectively. Define the cost functions $C_m : E \times N \rightarrow \Re$, $C_e : E \times N \rightarrow \Re$, and $C : E \times N \rightarrow \Re$ for some time length t:

$$C_m(e,n) = \begin{cases} 0 & \text{if } \delta(e) = n \\ c_m & \text{o.w.} \end{cases} \tag{2}$$

$$C_e(e,n) = \begin{cases} c_l & \text{if } \delta(e) = n \\ c_r & \text{o.w.} \end{cases} \tag{3}$$

$$C(e,n) = \sum_{i \in E} C_e(e, \delta(i)) r(e) p(e,i) t + C_m(e,n). \tag{4}$$

Note that $C(e,n)$ is the cost of moving entity e to node n, including the expected future costs of events queued by e. Now we have the optimization problem: find $\delta(e)$ to minimize

$$\sum_{e \in E} C(e, \delta(e)) \tag{5}$$

subject to

$$\forall n \in N : \sum_{e \in N_n} S(e) \leq U(n). \tag{6}$$

This problem as written is a non-linear form of the Generalized Assignment Problem (GAP). GAP is NP-Hard (Cohen, Katzir, and Raz 2006), and approximation algorithms for the non-linear GAP are only known for cases where the problem can be written as piecewise linear (Martello and

Toth 1990). Here, the non-linearity arises because the cost function C is dependent on the entity mapping δ, and thus cannot be easily made piecewise linear. There are no known algorithms for GAP problems with such dependencies, so our direct model will need to be reformulated.

2.3 Graph Model

Define an *interaction graph* G with the set of entities E as vertices. The edges between the vertices have a weight $w : E \times E \rightarrow \Re$ defined as

$$w(u,v) = r(u)p(u,v) + r(v)p(u,v). \tag{7}$$

The weight of an edge is the average number of events queued between the two entities. The weight of an edge between entities u and v is 0 (the edge does not exist) if $p(u,v) = p(v,u) = 0$, meaning the two entities will never queue events to each other.

Now the problem is to partition the vertices into $|N|$ sets such that the weight of the edges with vertices in two different sets is minimized, subject to the constraint of Eqn 6. This problem is referred to as the Capacitated Minimum k-Cut Problem (KCUT) and is also NP-Hard (Guttmann-Beck and Hassin 2000). However, KCUT admits a polynomial time approximation algorithm if k is fixed (as in this case where $k = |N|$) and the global optimum is approximable to $1 + 1/k$ (Gaur, Krishnamurti, and Kohli 2005).

This model tacitly sets $c_l = 0$ by ignoring all edges within sets of vertices. The cost of moving entities (c_m) is also ignored, thereby encouraging the algorithm to move entities no matter the short term fixed cost if the move will result in continuing long term gains of reduced remote queues. Thus this model is good for long simulations (t is large), but could make poor decisions in short simulations or toward the end of simulations. After the algorithm to solve the problem is introduced, we will discuss modifications to the algorithm that can be done to reintroduce the cost of moving an entity into the decision process.

3 ALGORITHM

The algorithm presented here is based on the one introduced by Gaur et al. (Gaur, Krishnamurti, and Kohli 2005). It begins with the vertices of the graph partitioned into feasible subsets. In this case, this is performed automatically by the simulator when it initially distributes entities to the nodes. Let

$$w_{uN_i} = \sum_{v \in N_i} w(u,v) \tag{8}$$

Algorithm 1: An approximation algorithm for capacitated minimum k-cut

```
Data: An interaction graph
Result: An approximately optimal distribution of
        entities
Partition the graph into feasible subsets
while true do
    Find a pair of vertices u ∈ N_i and v ∈ N_l such
    that Eqn 9 is satisfied
    if no such pair of vertices exist then
        break
    end
    else
        Move u to node l and v to node i
    end
end
```

be the total weight of edges between u and the vertices of N_i. Define the swapping condition for vertices $u \in N_i$ and $v \in N_l$ as

$$|N_i|w_{uN_i} + |N_l|w_{vN_l} < |N_i|w_{uN_l} + |N_l|w_{vN_i}. \tag{9}$$

The condition will be true essentially when the graph after the switch is more optimal than before the switch. The terms in the equation are scaled by $|N_i|$ and $|N_l|$ for technical reasons related to the overall convergence of the algorithm (Gaur, Krishnamurti, and Kohli 2005). The high level algorithm then is given in Algorithm 1.

Gaur et al. (Gaur, Krishnamurti, and Kohli 2005) show that when the swapping condition (Eqn 9) is met for all pairs of vertices, the resulting solution is within $1 + 1/k$ of the global optimum. Additionally, since the space of pairs of vertices is searchable in polynomial time (there are $\binom{|N|}{2} = \frac{|N|(|N|-1)}{2}$ pairs of vertices), results from (Orlin, Punnen, and Schulz 2004) can be applied that show the algorithm will complete in polynomial time. Note also that since entities are always swapped as opposed to moved individually, the capacity constraints of Eqn 6 are always met.

3.1 Distributed Computation

Since in this application the algorithm will always be executing on a cluster, it is advantageous to distribute the computations as much as possible across the nodes. Note that generally $r(u)$ and $p(u,v)$ are only known on the node containing u. Therefore the quantity $w(u,v)$ is not known on any node when u and v are resident on separate nodes. Define the partial edge weight w' as

$$w'(u,v) = r(u)p(u,v) \tag{10}$$

so that

$$w(u,v) = w'(u,v) + w'(v,u). \tag{11}$$

Define also the partial weight sums

$$w'_{uN_i} = \sum_{v \in N_i} w'(u,v) \tag{12}$$

and

$$w'_{N_i u} = \sum_{v \in N_i} w'(v,u). \tag{13}$$

w'_{uN_i} is the total partial weights of edges beginning at u and ending in N_i. $w'_{N_i u}$ is the total partial weights of edges beginning in N_i and ending at u. Node i knows w'_{uN_i} for all $u \in N_i$ and $w'_{N_i u}$ for all $u \in N$. We can now rewrite Eqn 8 as

$$w_{uN_i} = w'_{uN_i} + w'_{N_i u}. \tag{14}$$

This approach is analogous to converting the interaction graph into a directed graph with two edges between each pair of nodes. The edges now represent the cost of remote queues on an entity (the destination vertex) by another entity (the source vertex). Each node knows the weight of edges beginning at entities contained within it.

Going back to the swapping condition (Eqn 9), we see that half the information is contained on node i (w_{uN_i}, w'_{uN_l}, and $w'_{N_i v}$) and the other half on node l (w_{vN_l}, w'_{vN_i}, and $w'_{N_l u}$). We assume that $|N_i|$ is known at all nodes for all i, and define a partial swapping condition for pair of entities $u \in N_i$ and $v \in N_l$ containing only information resident on node i.

$$|N_i| w_{uN_i} < |N_i| w'_{uN_l} + |N_l| w'_{N_i v} \tag{15}$$

In order for the swapping condition to be true for a pair of entities, the partial swapping condition must be true for the pair of entities on at least one of the nodes holding one of the entities. Thus each node will search for pairs of entities for which the local partial swapping condition is true. When a pair is identified, it is sent to the root node. If the root node has received that pair from the other node, it knows it can swap the two entities. If it has not received that pair, it can either wait to hear from the other node or request from the other node the values in the partial swap condition. Thus the algorithm proceeds according to Algorithm 1, but the data computation and pairwise searching is distributed evenly across the nodes, with a root node needed only to perform simple pair aggregation.

3.2 Taking c_m Into Account

As mentioned earlier, the graph model of the entity distribution problem ignores the cost of moving an entity between nodes. As this cost could be substantial, at least relative to the expected gain of swapping two entities, it needs to be accounted for. Thus we bring back t, the window of time over which we are considering the value of swapping two entities, and c_m, the cost of moving an entity. The swapping condition can be modified to account for the moving cost as follows. Note that we are assuming without loss of generality that $c_r = 1$ and c_m is given as a multiple of c_r (i.e. $c_m = 2$ means that the cost of moving an entity is twice that of a remote queue).

$$|N_i| w_{uN_i} t + |N_l| w_{vN_l} t + 2c_m < |N_i| w_{uN_l} t + |N_l| w_{vN_i} t \tag{16}$$

or

$$(|N_i| w_{uN_l} + |N_l| w_{vN_i} - |N_i| w_{uN_i} - |N_l| w_{vN_l}) t > 2c_m \tag{17}$$

This is a stronger condition saying in effect that two entities will only be swapped if the expected gain from the swap over t time units exceeds the cost of moving the two entities. Note that the modified algorithm no longer solves KCUT, but in fact solves a problem closer to the direct model. Early in the simulation, t will be large so as to overpower the factor of c_m. However as the simulation nears the end, t will decrease and c_m will become a factor and unwise entity swapping will be prevented.

3.3 Parameter Estimation

Among the data used in the calculations described above, r, p, and c_m are not known a priori. c_m can be discovered through testing by measuring the time required to move an entity compared to the time lost due to a remote queue and will be constant from simulation to simulation. r and p will vary according to the simulation and therefore must be measured at runtime.

As the simulation is running, the simulation controller on each node is aware of all the events queued to and from entities resident on that node. Therefore, as the simulation runs node i can keep a running estimate of $p(u,v)$ for $u \in N_i$ and $v \in N_i$ and $r(u)$ for $u \in N_i$. These estimates will always be imperfect, however, and in the simulation section we will look at the effect of inaccurate values of r and p.

Table 1: Performance Simulation Results

Topology	Percent Decrease in Remote Queues
Random	32%
Torus	81%
Hypercube	88%

4 SIMULATION RESULTS

To verify the effectiveness of the algorithm, a simple PDES is created. Each entity contains a set of dependent entities it sends events to and the probability with which it sends events to each of these entity. When an entity receives an event, it randomly selects an entity out of its dependency set with the given probability and queues an event to it.

The dependency set of each entity is determined by the topology, a simulation parameter. Three topologies are used: random, torus, and hypercube. Random topology assigns each entity a given number of randomly selected dependencies. Torus topology imagines the entities arranged in a two dimensional grid with wrap-around (or the surface of a toroid / donut) and assigns each entity four dependencies: the entities directly above, below, left, and right in the grid. Hypercube topology arranges the entities as vertices in a binary hypercube of given dimension n with the dependency set of each entity being the n neighbors in the hypercube structure. These different topologies offer varying amounts of structure to the optimization algorithm.

In the simulations for which results are presented here, the random topology contains 1000 entities each with 5 dependencies, the torus topology uses a 25 by 40 grid of 1000 entities, and the hypercube topology has dimensionality 10 constituting 1024 entities. Entities are distributed randomly across 10 nodes, 1000 random seed events are created, and then the simulation begins, running for 10000 time steps.

4.1 Overall Performance

For each set of simulation parameters, the simulation is run with both the entity distribution algorithm turned on and off, so that a comparison can be made in the number of remote queues. Results are presented as a percent decrease in remote queues from the baseline simulation with no entity movements. Table 1 summarizes the results.

The optimization clearly shows the ability to reduce the number of remote queues in the event simulation. For simulations with regular structure, such as a torus or hypercube, this reduction can be dramatic. To put the results into perspective, if a parallel event simulator processing a simulation with hypercube topology spends 50% of its time recomputing events due to remote queues, an 88% reduction in remote queues leads to a 400% decrease in overall simulation time.

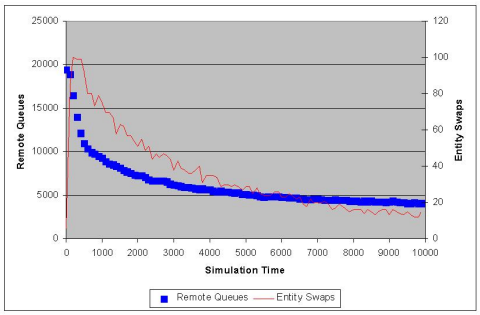

Figure 1: Hypercube Simulation Algorithm Performance

4.2 Convergence

Figure 1 shows a plot of the number of event queues and entity movements versus simulation time for the hypercube simulation. Results from other simulations are similar. The algorithm converges fairly rapidly, making 80% of its gains in the first 40% of the simulation. The number of entity swaps also decreases quickly with time.

4.3 Effects of Parameter Estimation

First, a series of simulations are run to determine the effects of estimating $p(u,v)$ and $r(u)$. In this implementation of the algorithm, nodes wait until an entity has sent 50 events before they begin using the calculated values of $p(u,v)$ and $r(u)$. This approach is compared with using the exact value of $p(u,v)$, which in this case is known by the entities, and with setting $r(u) = 1$ for all u, thus taking $r(u)$ out of the equations altogether.

Results show that these variations have no impact on the overall results. Figure 2 depicts this for $p(u,v)$, with the plots of estimated and exact nearly overlapping. In this case of $p(u,v)$, this shows that exact values (which practical simulators will never have) are unnecessary to the convergence of the algorithm. Designers may even be able to reduce the number of events per node required before parameter estimation begins in order to speed up convergence. In the case of $r(u)$, it shows that in practical terms it may be safely excluded, although no extra computational overhead is required to use it.

4.4 Effects of c_m

c_m, the ratio of the cost of an entity movement to the cost of a remote queue, may be difficult to measure in practice or may vary from simulation to simulation. Thus it is important to know the effect the value of c_m has on the

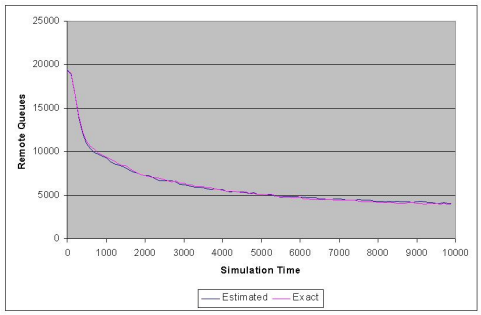

Figure 2: Effect of $p(u,v)$ Estimation on Performance

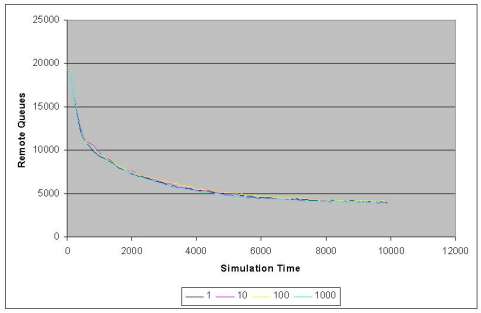

Figure 3: Effect of the Value of c_m on Performance

overall performance of the algorithm. A set of simulations are run that vary c_m from very small (1) to very large (1000).

Again results show that the value of c_m has no bearing on the performance of the algorithm. Figure 3 shows the plots for all 4 test cases nearly overlapping. One can see why by examining the modified swap condition (Eqn. 17). The importance of c_m is determined in large part by the value of t, which in this case equal to the current time subtracted from the end simulation time (10000). Thus for most of the simulation, the effect t swamps the effect of c_m even if it is very large. The exact value of c_m then may have an impact at the end of simulation, but by that time nearly all the gains from entity movements have already been realized.

In real implementations, designers may wish to further reduce the number of entity movements for practical reasons, such as reducing network bandwidth consumption. One option is to remove the factor of t from the swap condition and replace $2c_m$ with the absolute minimum value at which entity swaps should occur. In this case, the value of the constant will have significant impact of the performance of the algorithm and therefore will need to be carefully tuned.

5 CONCLUSION

Discrete event simulation is an important application in fields ranging from economics to electronics. It has proven to be a difficult problem to run in parallel, due to its sequential nature. This difficulty is embodied by the remote event queue, which is the source of inter-node dependencies. By reducing the number of these remote queues, the simulation will run more freely in parallel.

This paper has proposed moving entities between nodes during runtime to reduce the number of remote queues. A formalization of this dynamic entity distribution problem was presented, then an algorithm to solve the problem was proposed. The algorithm was simulated and showed that dramatic reductions in remote event queues are achieved by employing the algorithm. Additionally, several practical complications such as estimating the probability with which entities queue events to other entities bear little importance to the overall performance of the algorithm.

5.1 Future Research

The main focus of future research in this area will be incorporating this new technique into a real parallel discrete event simulator. This will allow evaluation of the algorithm in a real environment and possibly new practical problems will be identified and solved. Ultimately this will help speed up the execution of event simulations in general, which will aid research in many areas.

REFERENCES

Amdahl, G. 1967. Validity of the single-processor approach to achieving large-scale computing requirements. *Computer Design* 6 (12): 39–40.

Barr, R. 2004, March. Jist - java in simulation time user guide.

Chen, G., and B. Szymanski. 2005. Dsim: scaling time warp to 1,033 processors. In *Proceedings of the Winter Simulation Conference*, ed. M. Kuhl, N. Steiger, F. Armstrong, and J. Joines, 346–355. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Cohen, R., L. Katzir, and D. Raz. 2006. An efficient approximation for the generalized assignment problem. *IPL: Information Processing Letters* 100.

Gaur, D. R., R. Krishnamurti, and R. Kohli. 2005. The capacitated max-k-cut problem. In *International Conference on Computational Science and Its Applications (ICCSA), LNCS*.

Guttmann-Beck, N., and R. Hassin. 2000. Approximation algorithms for minimum K -cut. *ALGRTHMICA: Algorithmica* 27.

Martello, S., and P. Toth. 1990. *Knapsack problems: Algorithms and computer implementations*. New York: Wiley.

Orlin, J. B., A. P. Punnen, and A. S. Schulz. 2004. Approximate local search in combinatorial optimization. *SIAM Journal on Computing* 33 (5): 1201–1214.

Pasquini, R., and V. Rego. 1999. Optimistic parallel simulation over a network of workstations. In *Proceedings of the Winter Simulation Conference*, ed. P. Farrington, H. Nembhard, D. Sturrock, and G. Evans, 1610–1617. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Riley, P., and G. Riley. 2003. Spades - a distributed agent simulation environment with software-in-the-loop execution. In *Proceedings of the Winter Simulation Conference*, ed. S. Chick, P. Sinchei, D. Ferrin, and D. Morrice, 817–825. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

AUTHOR BIOGRAPHIES

MICHAEL SLAVIK is a Computer Engineering Ph.D. Candidate in the Department of Computer Science and Engineering at Florida Atlantic University. He was previously a software developer for Motorola, Inc in the iDEN and WiMAX handset development group. His research interests include distributed computing, wireless network evaluation, wireless sensor networks, and machine learning. His email address is <mslavik@fau.edu>.

IMAD MAHGOUB is a full professor of Computer Science and Engineering Department and director of Mobile Computing Laboratory at Florida Atlantic University, Boca Raton, FL. His research interests include mobile computing and wireless networking, advanced computer architecture, parallel and distributed processing, and performance evaluation of computer systems. He has published over 80 research papers in these areas. He served on the program committees of several international conferences and symposia. He is a senior member of the IEEE. He is also a member of the IEEE Computer Society, and the ACM. His email address is <imad@fau.edu>.

AHMED BADI is a Ph.D. Candidate in the Department of Computer Science and Engineering at Florida Atlantic University. His research interests are wireless sensors, network communication protocols, performance evaluation, and wireless networks reliability. He has several publications in wireless sensor networks, and performance evaluation. Before joining the Ph.D. program he worked as a software developer for several years. He can be reached at <ahassan1@fau.edu>.