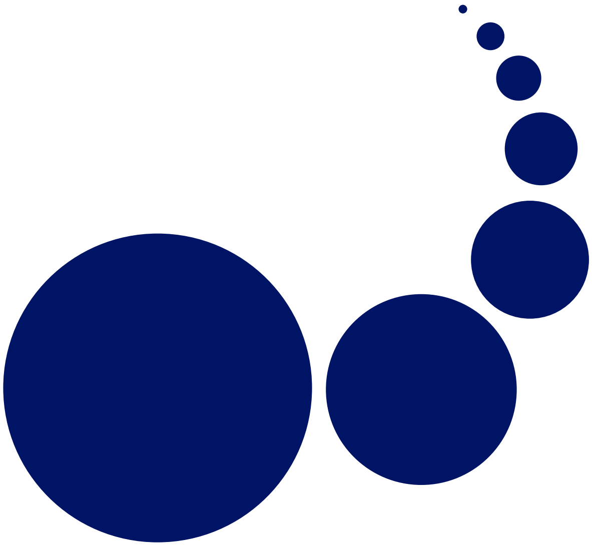
SCALABLE COMPUTING
Practice and Experience

Special Issue: Simulation in Emergent Computational Systems

Editors: Fatos Xhafa and Leonard Barolli



Volume 10, Number 1, March 2009

ISSN 1895-1767

UNIVERSITATEA DE VEST
DIN TIMISOARA

Scalable Computing: Practice and Experience ISSN: ISSN 1895-1767

Publisher:
West University of Timişoara
B-dul Vasile Pârvan 4
300223 Timisoara, România

Subscription Information: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 10, Number 1, March 2009

TABLE OF CONTENTS

Special Issue Papers:

Introduction to the Special Issue **i**
Fatos Xhafa, Leonard Barolli

Recent Advances on the Simulation Models for Ad Hoc Networks: Real Traffic and Mobility Models **1**
Arta Doci, Leonard Barolli and Fatos Xhafa

Simulation Framework for the Evaluation of Dependable Distributed Systems **13**
Ciprian Dobre, Florin Pop, Valentin Cristea

Impact of the Dynamic Membership in the Connectivity Graph of the Wireless Ad hoc Networks **25**
Arta Doci, William Springer and Fatos Xhafa

Automatic Performance Model Transformation from a Human-intuitive to a Machine-efficient Form **35**
Sabri Pllana, Fatos Xhafa and Leonard Barolli

Replay-based synchronization of timestamps in event traces of massively parallel applications **49**
Daniel Becker, John C. Linford, Rolf Rabenseifner and Felix Wolf

Performance Analysis and Optimization of Parallel Scientific Applications on CMP Clusters **61**
Xingfu Wu, Valerie Taylor, Charles Lively and Sameh Sharkawi

Minimization of Download Time Variance in a Distributed VOD System **75**
Anne-Elisabeth Baert, Vincent Boudet, Alain Jean-Marie and Xavier Roche

Virtual Large-Scale Disk Base on PC Grid **87**
Erianto Chai, Katsuyoshi Matsumoto, Minoru Uehara and Hideki Mori

Performance Evaluation of a Wireless Sensor Network for Mobile and Stationary Event Cases Considering Routing Efficiency and Goodput Metrics **99**
Tao Yang, Leonard Barolli, Makoto Ikeda, Giuseppe De Marco and Arjan Durresi

Spesial Issue Book Review:

Ad Hoc Networks: Technologies and Protocols **111**
Reviewed by Arta Doci

Research Paper:

The Edge Node File System: A Distributed File System for High Performance Computing **115**
Kovendhan Ponnavaikko and Janakiram D

Scalable Computing: Practice and Experience
Volume 10, Number 1, pp. i–ii. http://www.scpe.org

ISSN 1895-1767

INTRODUCTION TO THE SPECIAL ISSUE: SIMULATION IN EMERGENT COMPUTATIONAL SYSTEMS

With the fast development of Internet and other new technologies, new computational infrastructures and paradigms are emerging. Emergent Computational Systems comprise Grid and P2P systems and wireless ad hoc network among others. The development and analysis of such systems is a challenging task due to many intrinsically complex features of these systems. Simulation appears again to be the indispensable means for modeling, performance analysis as well as implementations and analysis of algorithms before deployment in such systems.

This special issue follows the *First International Workshop on Simulation and Modeling in Emergent Computational Systems (SMECS-2008) September 8-12, 2008, Portland, Oregon, USA* held in conjunction with the *2008 International Conference On Parallel Processing (ICPP-08).*

The papers of the workshop presented innovative methods and techniques related to Simulation, Modeling and Performance Evaluation in Emergent Computational Systems. The special issue comprises 9 papers, organized as follows.

The first paper, *Recent Advances on the Simulation Models for Ad Hoc Networks: Real Traffic and Mobility Models* by Doci et al., surveys recent advances on simulation models for wireless ad hoc networks. The paper stresses the importance of the mobility metrics in the simulation of wireless ad hoc networks.

In the second paper, *A Simulation Framework for Dependable Distributed Systems*, Dobre et al. present a solution to evaluating the correctness and performance of various dependability-related technologies for distributed systems using the formalism provided by the modeling and simulation domain. The proposed approach is based on an extension proposal of the simulation model called MONARC, which is a generic simulation framework designed for modeling large scale distributed systems.

The third paper by Doci et al., *Impact of Mobility in the Connectivity of Wireless Ad Hoc Networks*, presents an approach for assessing the impact of mobility in the dynamic connectivity graph, on both nodes and edges. The authors introduce an algorithm that computes the Maximum Node Degree, Link Durations, and Path Durations mobility metrics for the dynamic topology connectivity graphs. Lower and upper bounds for these mobility metrics are also provided.

In the fourth paper by Pllana et al., *Automatic Performance Model Transformation From A Human-Intuitive to a Machine-Efficient Form*, the authors address the issue of the development of performance models for programs that may be executed on large-scale computing systems. They propose a model to bridge the gap between the performance modeling and software engineering by incorporating UML. The user specifies graphically the performance model using UML. Then, the transformation of the performance model from the human-usable UML representation to the machine-efficient C++ representation is done automatically. The authors demonstrate the usefulness of their approach by modeling and simulating a real-world material science program.

The fifth paper by Becker et al., *Replay-based synchronization of time stamps in event traces of massively parallel applications*, studies the usefulness of event traces in understanding the performance behavior of message-passing applications. This study is built on earlier work by the same authors. Here, the parallel design and its implementation within the SCALASCA trace-analysis framework are presented.

In the sixth paper by Wu et al., *Performance Analysis and Optimization of Parallel Scientific Applications on CMP Cluster Systems* the authors present performance analysis in using CMP (Chip multiprocessors) based cluster systems for large-scale scientific applications. A detailed performance analysis to identify how applications can be modified to efficiently utilize all processors per node on CMP clusters is presented.

In the seventh paper, Baert et al., *Minimization of Variance Download Times in A Distributed VOD System*, examine the problem of minimizing the variance of the download time in a particular Video on Demand System based on Grid Delivery Network –an hybrid architecture based on P2P and Grid infrastructure. Different heuristics to solve it in practice, and validation through simulation are provided.

In the eighth paper by Chai et al., *Virtual Large-Scale Disk Based on PC Grid*, the authors develop the VLSD (Virtual Large-Scale Disk) toolkit to assist in the construction of large-scale storage using only cheap commodity hardware and software. To simulate a typical storage system in an educational environment, which may contain many small files, the authors built GCC on various storage systems and evaluated the performance. The authors show that the storage system is usable even with many small files.

In the last paper by Yang et al., *Performance Evaluation of a Wireless Sensor Network for Mobile and Stationary Event Cases Considering Routing Efficiency and Goodput Metrics*, the authors investigate how the sensor network performs in the case when the event node moves. They carried out the simulations for lattice topology and TwoRayGround radio model considering AODV and DSR protocols. For the performance evaluation, they considered two metrics: routing efficiency and goodput and they compare the simulation results for two cases: when the event node is mobile and stationary. The simulation results have shown that the routing efficiency for the case of mobile event node is better than the stationary event node using AODV protocol. Also, the goodput for the mobile event node case does not change too much compared with the stationary event case using AODV, but the goodput is not good when the number of nodes is increased.

We are grateful to all authors for submitting their papers to this special issue and to the reviewers for their feedback to the authors. We would like to thank Marcin Paprzycki (SCPE editor-in-chief) and Alexander Denisjuk (SCPE technical and managing editor) for the opportunity to edit this special issue and for their timely support.

Fatos Xhafa,
Technical University of Catalonia,
Department of Languages and Informatics Systems,
C/Jordi Girona 1-3, 08034 Barcelona, Spain
fatos@lsi.upc.edu,

Leonard Barolli,
Department of Information and Communication Engineering,
Fukuoka Institute of Technology (FIT),
3-30-1 Wajiro-Higashi,
Higashi-Ku, Fukuoka 811-0295, Japan
barolli@fit.ac.jp

Scalable Computing: Practice and Experience
Volume 10, Number 1, pp. 1–11. http://www.scpe.org

ISSN 1895-1767
© 2009 SCPE

RECENT ADVANCES ON THE SIMULATION MODELS FOR AD HOC NETWORKS: REAL TRAFFIC AND MOBILITY MODELS

ARTA DOCI[*] LEONARD BAROLLI[†] AND FATOS XHAFA[‡]

Abstract. In order to provide credible and valid simulation results it is important to built simulation models that accurately represent the environments were ad hoc networks will be deployed. Recent research results have shown that there is a disparity of 30% between protocol performance in real test beds and the one in simulation environments. In this paper we summarize the recent trends on the simulation models for ad hoc networks. First, we provide a survey of the synthetic and real traffic models used in ad hoc network simulation studies. Second, we select a representative of the most used mobility synthetic model, the real pedestrian mobility model, and the real vehicular model (for mixed traffic). We show that the real pedestrian and real vehicular model share common mobility characteristics: a) The transition matrix on both models illustrate that wireless nodes do not move from one location to another at random, but they are rather based on activities (work, shopping, college, gym); b) The dynamic membership property illustrates that nodes join and leave the simulation based on some variable distribution or patterns. Lastly, via simulations we show that when using realistic simulation models the simulation protocol performance closer reflects with the real test bed protocol performance.

Key words: wireless ad hoc networks, traffic model, mobility model, performance evaluations.

1. Introduction. Synthetic mobility models are mainly used to evaluate the protocol performance in wireless networks. The synthetic mobility models [1] are very useful, since they are easy to implement and mathematically tractable. However, the study at Uppsala University [2] shows that the wireless protocol performance in real test beds drops by 30% from the ones in the simulation platforms. The main reason for the disparity is the use of synthetic simulation models, including mobility and traffic models, which do not closely model the environments where the wireless networks will be deployed.

Recently, we have started to see a shift on using more realistic simulation models in the simulation evaluations. For example, the authors of [3] issue a ‘A call to arms: It’s time for REAL mobility models’, thus they design and implement a more realistic pedestrian mobility model. In addition, to further improve the validity and credibility of the simulation studies the authors of [4, 5] show that mobility and traffic are interconnected, as well as, implement a more realistic traffic model. The studies show that under more realistic mobility and traffic models the simulation protocol performance better reflects the protocol performance of real deployments.

In addition, more realistic vehicular mobility models are implemented [6, 7, 8, 9, 10, 11, 12]. For example, the authors of [10] implement a more realistic vehicular model by using the publicly available TIGER (Topologically Integrated Geographic Encoding and Referencing) database from the U.S. Census Bureau, giving detailed street maps for the entire United States of America, and model the automobile traffic on these maps. First, the model can be very complex, since it needs to query the database for every location. Second, the database does not provide any speed limit information for each location. Lastly, the model makes assumptions about the speed distribution and other pertinent parameters. Specifically, the models do not take into account mixed traffic conditions [13, 14].

The focus of this paper is placed on the recent advances of the real traffic and mobility models for ad hoc networks. Specifically, the paper focuses on the more realistic simulation models, which are extracted from real user trace data on student campuses, GPS mounted devices on vehicles in city roads, and devices mounted on traffic lights for the mixed traffic conditions. For example, the paper highlights the similarities of real mobility models extracted from real user data for both the pedestrian and vehicular cases. In addition, it describes the realistic traffic models.

The contributions of this paper are three fold. First, it provides a survey of synthetic and real traffic models. Second, it shows that real mobility models for pedestrian and vehicular situations share common mobility characteristics. Third, it shows via simulations that real simulation models produce performance results that are closer to the real test beds evaluations, thus proposes that to improve the validity and the credibility of the ad hoc simulation results the use of real simulation models should be the preferred models.

[*] Colorado School of Mines, Colorado, USA (adoci@mines.edu). Also, Head of Wireless Research at UNYT.
[†] Fukuoka Institute of Technology, Fukuoka 811-0214, Japan (barolli@fit.ac.jp).
[‡] Technical University of Catalonia, Barcelona, Spain (fatos@lsi.upc.edu).

2. Simulation Traffic Models. In this section we provide a survey of the synthetic simulation traffic models that are used in ad hoc network simulation studies. The most used synthetic traffic model is the Constant Bit Rate (CBR), which is described below. The main reason of the widespread use of the CBR is based on its simplicity on both design and implementation. In addition, we describe the implementation and design of a real traffic model. We show that the real traffic model suggests that traffic is not sent on constant streams, but rather on bursts.

2.1. Synthetic Traffic Models. NS 2 views traffic as a two layer approach (as shown in Figure 2.1). The first layer implements transport protocols, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), while the second layer transmits application level data. For example, FTP and Telnet are two applications that generate traffic over TCP. On the other hand, Constant Bit Rate (CBR), Exponential, Pareto, and Poisson are applications that can be used to generate traffic over UDP.

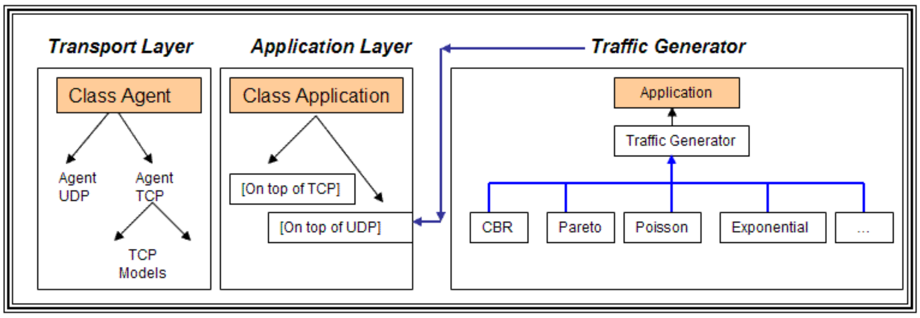


Fig. 2.1. *Traffic Generation Process in NS 2.*

Traffic modelling can be classified under two main groups. The first group is that of applications that generate continuous traffic, namely the CBR that is the most used traffic model in ad hoc network simulation studies [15, 16]. In the second group, there are applications that model packet arrivals under Pareto, Exponential and Poisson Distribution [17, 18]. The latter, can be considered as applications that generate bursty traffic. We briefly describe these models next.

2.1.1. Constant Bit Rate Model. In this model source nodes generate traffic at a constant rate that are sent every interval ΔT seconds, which is defined by the user. For example, many sensor network applications generate constant bit rate traffic. In most ad hoc network applications wireless nodes do not send traffic continuously.

2.1.2. Exponential and Poisson Models. The exponential traffic model sends traffic during an '*on*' state and does not send any traffic during the '*idle*' state. The traffic is generated during the '*on*' state with the following parameters: the packet size, the traffic rate, and the burst time[1].

The exponential distribution can be viewed as a Poisson process when setting the variable burst time to zero and the variable traffic rate to a large value. In general, the packet inter-arrival times in the Poisson process are exponentially distributed. The probability distribution function is given by Eq. (2.1) and the density function is given by Eq. (2.2).

$$F(t) = 1 - \exp^{-\lambda t}, \text{ where } \lambda \text{ is arrival rate} \tag{2.1}$$

$$f(t) = \lambda \exp^{-\lambda t}. \tag{2.2}$$

2.1.3. Pareto Model. The Pareto traffic model also has the 'on' and 'idle' states, as the exponential traffic model. The packet arrival times of the Pareto distribution are independent and identically distributed, which means that each arrival time has the same probability distribution as the other arrival times and all are mutually independent. The two main parameters of the Pareto process are the shape and the scale parameter. The probability distribution function is given by Eq. (2.3) and the density function is given by Eq. (2.4).

[1] The mean of the '*on*' time that is withdrawn from an exponential distribution

$$F(t) = 1 - \left(\frac{b}{t}\right)^{a}, \text{ where } a, b \geq 0 \text{ and } t \geq 0 \tag{2.3}$$

$$f(t) = \frac{ab^{a}}{t^{a+1}} \text{ for } t \geq b, \tag{2.4}$$

where a is the shape parameter and b is the scale parameter.

2.2. Real Traffic Model. Real traffic models have not been used much in the simulation evaluations. The main reason is the lack of the implementation, which are freely available to the research community. In this section we describe RealTrafficGen [5] model, which stands for real traffic generator. RealTrafficGen is interconnected with RealMobGen [3] mobility model and retrieves the dynamic membership of the wireless nodes, but it also adds three new features:

1. The generated traffic is dependent on the location, thus this model introduces distributions for each hotspot (based on real data).
2. Mobile nodes send more traffic then stationary wireless nodes, due to forwarded traffic from different hotspots, as well as their own traffic. This is the first model that differentiates the amount of traffic generated by the type of the node.
3. Traffic is likely to satisfy self-similarity property, rather then stationary distributions, due to diurnal circle; it is implemented as Weibull distribution.

The model creates `HotspotTrafficArray` (see an example below). The first and third rows provides the hotspot number, while the second and forth rows shows the traffic flow on the respective hotspot, which is a Weibull distribution with parameters the shape (β) and mean (μ). For example, the Weibull distributions for hotspot 1 and hotspot 8 are shown in Figure 2.2.

1	2	3	4
$(0.013, 0.85)$	$(0.028, 0.98)$	$(0.019, 0.95)$	$(0.006, 1.31)$
5	6	7	8
$(0.005, 1.48)$	$(0.010, 0.74)$	$(0.011, 0.70)$	$(0.007, 0.18)$

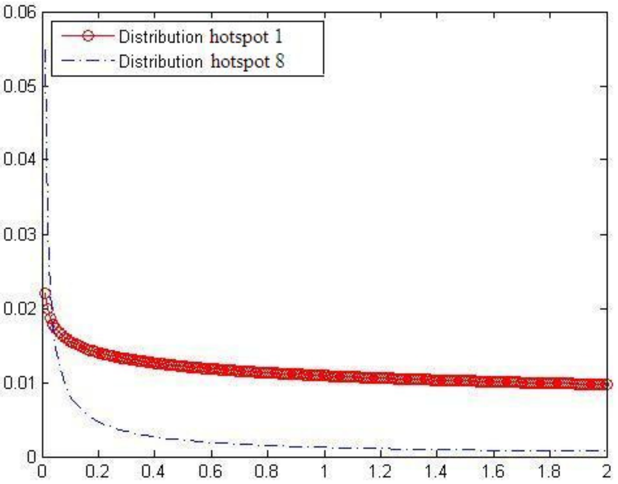


Fig. 2.2. *Weibull probability distribution function for hotspot 1 and 8.*

3. Mobility Models. The main characteristics of a mobility model are speed, pause distribution, and direction of movement. In this section we describe one representative of the following simulation mobility models: synthetic, real pedestrian, and real vehicular (mixed traffic case).

3.1. Synthetic Mobility Model. The most used synthetic mobility model is Random Walk Model (RWM) [15]. In RWM each node is assigned a randomly distributed initial location $(x0; y0)$. Then, each node randomly picks up a destination independent of their initial positions and moves toward it with speed chosen uniformly on the interval ($v0; v1$). Nodes pause upon reaching each destination. The process is repeated until the allotted simulation time is reached.

The main advantage of RWM is its simplicity. On the other hand it has two major drawbacks. First, the direction of movement is not random. Second, the nodes do not start and remain in the simulation for the entire simulation time.

3.2. Real Pedestrian Mobility Model. In this section we describe RealMobGen, which is a hybrid model that is based on Dartmouth's model of mobile network traces [22] and USC's WWP [20] survey collected from the students on campus. The model closely mimics the environments where ad hoc networks will likely be deployed, since it borrows its characteristics from models derived from real user traces. Another feature of RealMobGen, that is not existent in any other current mobility model, is the classification of nodes as stationary (46% of the nodes) and mobile (54% of the nodes). The ratio of stationary vs. mobile nodes was borrowed from the Dartmouth model.

The stationary nodes select a location based on a transition matrix that defines the probabilities for moving from one point to another. Once a location is selected, a node is turned on for a time drawn from the exponential distribution of start time for the stationary nodes. Stationary node stays at the selected location until the allotted stationary end time. The mobile nodes, also, select a start location based on the transition matrix. The mobile node enters the simulation at a time drawn from the mobile node start time. The node pauses at the selected location until the allotted pause time from mobile pause time exponential distribution. After the pause time is up, the mobile node selects the next location based on the transition matrix and moves there not in a straight line but following data that supports movements along popular roads and turns. The mobile node repeats the pattern 'pause-select next location - move there' until allotted mobile simulation end time.

RealMobGen shows that wireless nodes tend to cluster around popular locations, i. e., cafeteria, gym, classes, and library. We believe that RealMobGen is the first mobility model (we are not aware of any other models) that implements the dynamic characteristic of wireless devices in NS 2, devices join and leave the network at different times. RealMobGen addresses the drawbacks present in the RWM by implementing the following new features:

Feature 1: Wireless Nodes are clustered around popular hotspots. For example, Figure 3.1 shows a snapshot of RealMobGen with 150 nodes, which are clustered around 14 hotspots.

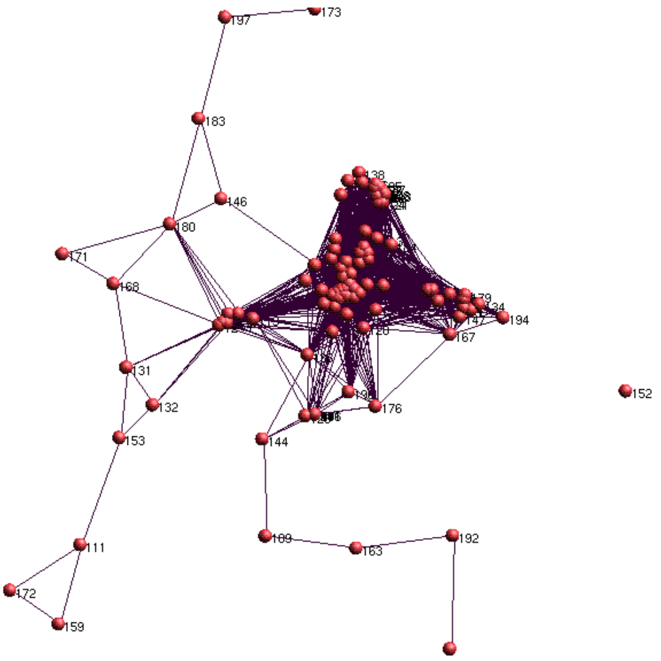


FIG. 3.1. *150 Nodes cluster around 14 Hotspots.*

Feature 2: Wireless Nodes posses dynamic membership. For example, Figure 3.2 shows the dynamic membership of 60 nodes.

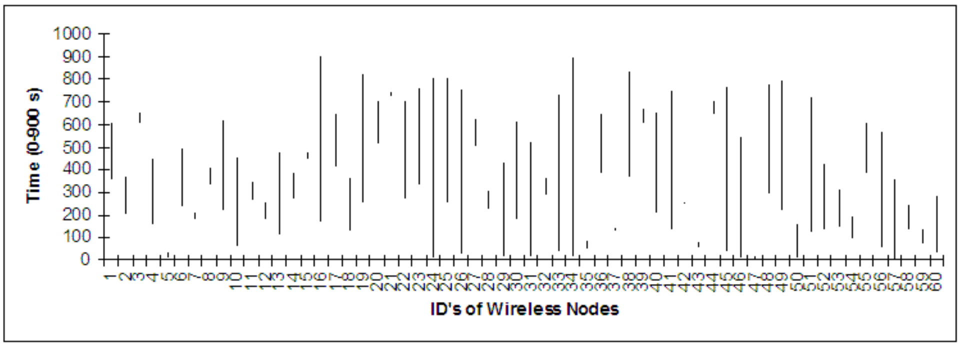


Fig. 3.2. *Dynamic Membership of 60 Nodes.*

Feature 3: Nodes are classified on two flavors, namely stationary and mobile (stationary (46%)of the nodes and mobile (54% of the nodes). The ratio of stationary vs. mobile nodes was borrowed from the Dartmouth model.

Feature 4: Moving from one point to another is done via waypoints, instead of a straight line.

3.3. Real Vehicular Mobility Model. MixMobGen [26] is based upon the data collected by the Indian Institute of Technology [13] and from the Battelle Memorial Institute [14]. Both data sets have automatically collected mixed traffic data. MixMobGen is the first mobility model, we are not aware of any other one, that presents mixed traffic conditions by realistically implementing the speed as a bimodal distribution, the direction of movement based on a probability transition matrix, and the wireless nodes to poses the dynamic membership property (join and leave the simulation at some random time).

3.3.1. Data Sets. The first data set [13] was collected on 17 different data sections of the national and state highways in different parts of India, which were chosen to have a wide variation of fast vs. slow moving vehicles. The data collected on each section involved 2 hours of a typical weekday. (Refer to Figure 3.3 for a summary of the traffic sample size on each data section). For example, as shown in the Figure 3.3, the sample size on data section 6 is $1,407$ vehicles.

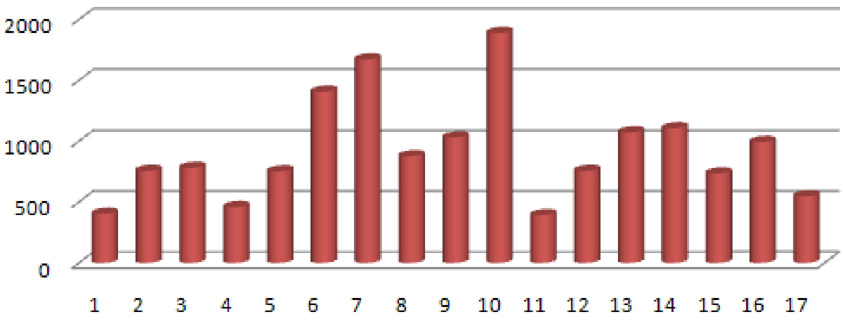


Fig. 3.3. *Number of the vehicles sampled on 17 data sections.*

On each data section, different ratios of fast moving vehicles vs. slow moving vehicles were captured. The hypothesis of the study was:

Hypothesis: Speed Data distribution on mixed traffic conditions does not follow normal distribution.

The collected data supported the hypothesis on 13 out of the 17 (77%) data sections (As shown in Figure 3.4). For example, the red bars that represent the data sections 1, 3, 8, 12 show that the null hypothesis was rejected on only four data sections due to the fact that data was better modeled by unimodal distribution, but supported on the other 13 data sections. Furthermore, the graph shows that the bimodality of the data is not correlated to the volume of the fast vs. slow moving vehicles. For example, bimodality is supported when the ratio of slow moving vehicles was 14% on data section 11 or as high as 31% on data sections 5, 6, and 7. Analogously, the distribution was unimodal at low ratio of slow moving, i. e., 19% on data section 1 or at high ratio, i. e., 39% on data section 8.

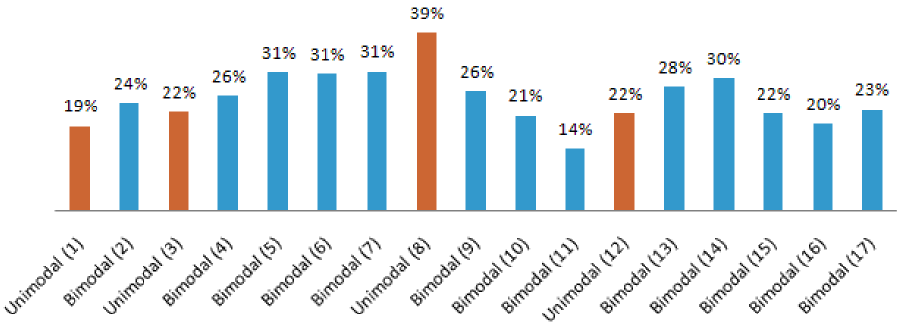


FIG. 3.4. *Ratio of the Slow Moving Vehicles and the Speed Distributions.*

The second data set [14] covered the Lexington area of approximately 461 square miles with a total population of approximately 350, 000. The sample was comprised of 100 households and included data collected via GPS mounted systems in the cars, which provides useful information for the mobility parameters that could not be extracted from the first data set, including direction of movement, trip start times, and dynamic membership properties.

The second data set reveals that the direction of movement is not random, but rather it is based on person's activities. For example, on the case of traveling on local streets the purpose of trips was classified as follow:
Work Place 10%
Social/Recreational Activity 13.8%
Eat Out 15.8%
Shopping 14.9%

In addition, it emphasizes that nodes posses dynamic membership, thus are not in the simulation during the entire simulation time, but rather a fraction of the simulation time.

3.3.2. MixMobGen Parameters. In this section we discuss the parameters of the MixMobGen and the implementation choices in NS 2.

Speed Distribution. Speed distribution is extracted from the first data set [13], which shows that on the 13 out of the 17 of the data sections the speed of mixed traffic is best modeled by bimodal distribution. For example, on the Table 3.1 we show 13 data points collected at data section 6.

We assessed many distributions to graph the data, however we adopted the interpolation [27] method, which is the process of defining a function that takes on specified values at specified points (As shown in Figure 3.5). The figure and the study supports that the speed distribution is bimodal with the mean to be *12.5* on the first peak and the mean to be *37.5* on the second peak.

Direction of Movement. Wireless devices are carried by humans, thus the human movements would be the best approximation to the mobility patterns of the mobile nodes. We are aware that humans do not move at random, but rather based on activities. The data collected on Lexington area supports that humans move based on activities and the findings are summarized in Figure 3.6. We implemented this feature by introducing a $Transitional\_Destination\_Prob\_Matrix$, which places a weight of 0.1 on the Shopping and Other Errands activities, or a weight of 0.26 on the Return Home activity.

Dynamic Membership. The length of the trips, as well as, the start times of the trips were collected. In Figure 3.7 we see that 50% of the trips were on length of $[0, 9]$ minutes, 30% of the trips were on length of $[10, 19]$ minutes, 10% of the trips were on length of $[20, 29]$ minutes, 4% of the trips were on length of $[30, 39]$ minutes, 2% of the trips were on length of $[40, 49]$ minutes, and 4% of the trips were on length of $[50, +]$ minutes.

Table 3.1
Speed Distribution Data

Speed (kmph)	Probability Density
8	8
13	52
16	33
20	8
24	6
28	33
33	36
38	44
40	26
44	18
48	2
52	3
56	2

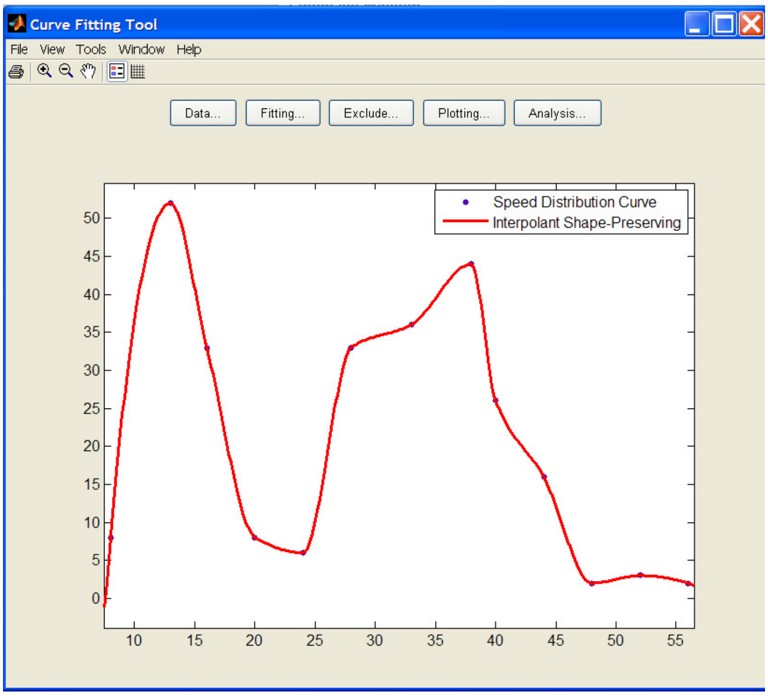


Fig. 3.5. *Shape-preserving interpolation curve on the data collected in data section* 6.

In the implementation phase we introduced the array *Start_time_of_Trip*, which has the percentage of the nodes that become active at time 0 (of the simulation). For example, 27% of the nodes become active at time 0), 5, 10, and (increments of 5 until the full hour is reached). In addition, we introduce the array the *Active_time_of_Nodes*, which presents the weights of the trip lengths.

3.3.3. Algorithm of MixMobGen. The simulation duration time (T) and the number of nodes (N) are the inputs entered by the user. First, 11 popular locations are defined, i.e, Shopping, Errands, University, and Work. In MixMobGen the nodes are distributed based on the weights defined on the *Transitional_Destination_Prob_Matrix*. The nodes active state is determined by the values on the arrays *Start_time_of_Trip* and *Active_time_of_Nodes*. The algorithm of the *MixMobGen* is presented by Algorithm 1.

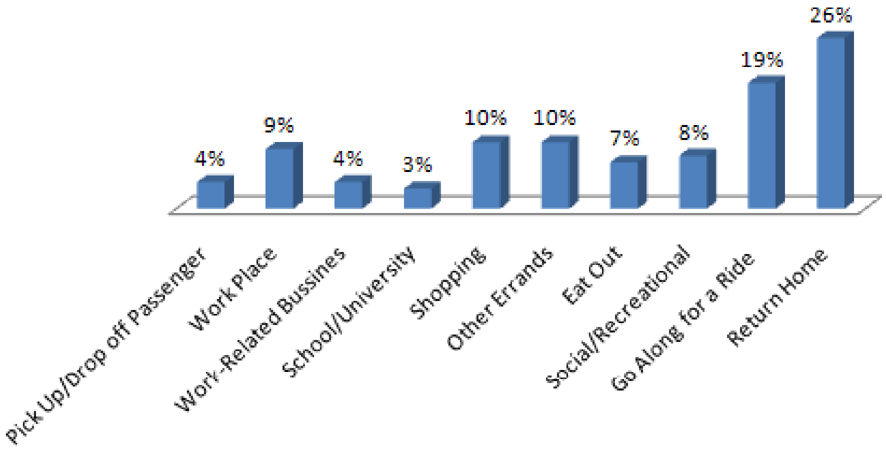


Fig. 3.6. *Destinations (As % of trips).*

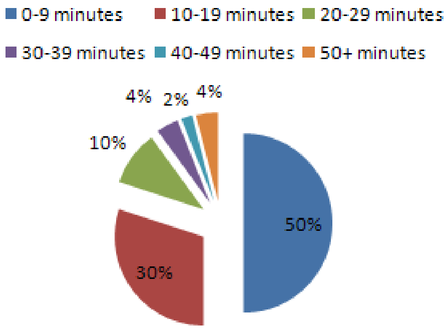


Fig. 3.7. *Length of Trips.*

Algorithm 1 : *MixMobGen.*

```
Input:  Simulation Time (T); Number of Nodes (N);
Compute Transitional_Destination_Prob_Matrix
Compute Start_time_of_Trip as a function of N
Compute Active_time_of_Nodes as a function of N, T
INITIALIZATION
for each node $\epsilon$ N do
   InitialLocation from the Transitional_Destination_Prob_Matrix
   Speed (S) from the BimodalSpeed Distribution
   ActiveTime from the Active_time_of_Nodes
   TripStartTime from Start_time_of_Trip
end for
for each node $\epsilon$ N do
   Select Destination (D) to move to from the Transitional_Destination_Prob_Matrix
   Move toward D with speed S from Initial Location
   if upon reaching D the node is still ACTIVE then
      Select new Destination and Speed
      Move toward the new destination with the new speed
   end if
end for
Output:  Mobility Patterns File
```

4. Protocol Performance Evaluations. The MixMobGen mobility was used to generate the movement pattern of the wireless nodes. mobility. In the routing layer AODV [24] and DSR [21] were selected, since they are the most used ones in the performance evaluations studies. The propagation model is the two-ray-ground [25]. The parameters that were not varied in the simulations were the number of nodes set to 40, simulation area $900m \times 1200m$, simulation time set to $900s$, the IEEE 802.11 [28] as the protocol for the medium access control (MAC) layer model.

We summarize in Table 4.1 the parameters used in the simulation.

Table 4.1
Simulation Parameters.

Parameter	Value (s)
Routing	AODV and DSR
MAC	802.11
Number of Nodes	40
Simulation Area	$1200m \times 900m$
Simulation Time	900 s
Propagation Model	Two-Ray-Ground
Radio Range	250 m
Traffic	Constant Bit Rate
Mobility	MixMobGen

In addition, the derived parameters that are calculated from the number of nodes (40); the simulation area ($900m \times 1200m$); and the transmission range (R=250m) are provided below (for further explanations on each of the derived parameters we refer the reader to [29].)

Node Density: Number of nodes divided by the simulation area. In our case it is (900 × 1200)/40, thus 1 node for $27,000m^2$.

Coverage Area: Area with the transmission range as radius. In our case it is $\Pi * R^2$=$196,349m^2$.

Maximum Path Length: The diameter of the rectangle $900m \times 1200m$ equals to 1500.

The network Diameter: The maximum path length divided by the transmission range, which in our case turns out to be 6 *Hops*.

Network connectivity no edge effect: The coverage area by the node density, which turns out to be 7.27 *Hops*.

The performance metric used in this simulation evaluations is *Availability*, which is a performance metric that takes into account the dynamic membership and we define it hereby.

Availability: We define Availability as the ratio between the number of packets sent by the source and the number of packets received by the destination, while the node is active.

MixMobGen shows that nodes are clustered around the main activities and their probabilities are defined by the $Transitional\_Destination\_Prob\_Matrix$. For example, Figure 4.1 shows the visualization of MixMobGen on 40 nodes.

The traffic file is generated with three different sources $(10, 20, 30)$ sources, respectively, 40 nodes, and the rate of generating packets was set to 4 packets. Each data point represents an average of *fifty* runs with different traffic and different randomly generated mobility scenarios. The results of the experiments are summarized in Table 4.2. The results, also, include the 95% confidence intervals (CI) for validation of the experiments.

Table 4.2
Performance evaluation under MixMobGen.

Number of Sources	Availability: AODV	95% CI: AODV 95% CI: AODV	Availability: DSR	95% CI: DSR 95% CI: DSR
30	55.65%	55.65 ± 4.03	56.11%	56.11 ± 4.16
20	66.64%	66.64 ± 4.22	66.48%	66.48 ± 4.68
10	68.49%	68.49 ± 4.14	70.02%	70.02 ± 4.00

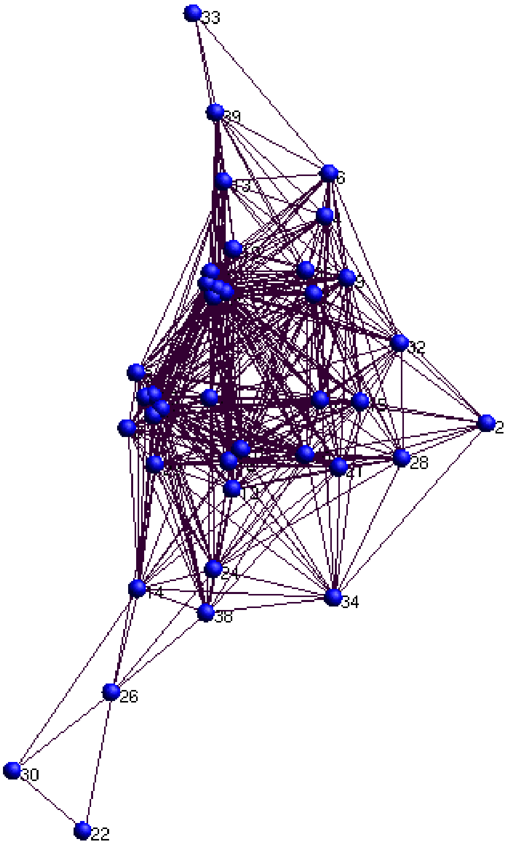


FIG. 4.1. *MixMobGen on 40 Nodes.*

5. Conclusions. In this paper we first described the synthetic mobility and traffic models. Specifically, we described in detail three real simulation models, namely: RealTrafficGen, RealMobGen, and MixMobGen. We believe that the mobility generator should adjust automatically to pedestrian mobility patterns, fast moving traffic, and mixed traffic. Therefore, in the future plan to design Knob Mobility Generator, which is based on the context can automatically adjust to the proper mobility model.

Furthermore, MixMobGen and RealMobGen suggest that mobility models that are extracted from real user data posses mobility characteristics that are rather different from the synthetic mobility models. When we evaluate the protocol performance using realistic mobility models, the performance drops significantly from the evaluations done when using synthetic mobility models.

The realistic mobility models shed new light into the ad hoc protocol design, as well. For example, the realistic mobility models show that nodes tend to cluster around popular locations. When the nodes are within the cluster they tend to be less mobile, but when they are between the clusters the nodes tend to be more mobile. However, none of the popular protocols captures this reality. In the future, we are going to address ad hoc protocol design based from on the real data sets collected for real world scenarios.

Lastly, we believe that realistic simulation models significantly improve the credibility and validity of the simulation models. This paper shows that when using more realistic simulation models, the protocol performance more closely reflects the one in the real test beds. In addition, the paper points out that there are similarities between pedestrian mobility models and mixed traffic condition mobility models. The main similarities are:

- Direction of movement: It is based on people activities.
- Dynamic Membership: Wireless Nodes join and leave the network dynamically.
- Transition Matrix : Moving from one location to another is based on weighted probabilities.

The main reason for the similarities between the simulation models are based on the fact that wireless nodes are carried by people, thus will closely model the people behavior and movements.

REFERENCES

[1] T. Camp, J. Boleng, and V. Davies, *A survey of mobility models for ad hoc network research*, Wireless Communications and Mobile Computing, vol. 2, no. 5, pp. 483–502, 2002.

[2] E. Nordström, P. Gunningberg, C. Rohner, and O. Wibling, Evaluating wireless multi-hop networks using a combination of simulation, emulation, and real world experiments. In *MobiEval'07: Proceedings of the 1st International Workshop on System Evaluation for Mobile Platforms*, pages 29–34, 2007.

[3] C. Walsh, A. Doci, and T. Camp, *A call to arms: it's time for real mobility models*, vol. 12, no. 1, pp. 34–36, 2008.

[4] A. Doci, *Interconnected traffic with real mobility tool for ad hoc networks*,International Conference on Parallel Processing Workshops, International Conference on, vol. 0, pp. 204–211, 2008.

[5] A. Doci and F. Xhafa, *WIT: A Wireless Integrated Traffic Model* Mobile Information Systems Journal (4), pp 1–17, IOS Press, 2008.

[6] F. K. Karnadi, Z. H. Mo, and K.C. Lan, *Rapid generation of realistic mobility models for Vanet*, pp. 25062511, 2007.

[7] D. R. Choffnes and F. E. Bustamante, *An integrated mobility and traffic model for vehicular wireless networks*, in VANET '05: Proceedings of the 2nd ACM international workshop on Vehicular ad hoc networks, pp. 69–78, 2005.

[8] R. Mangharam, D. S. Weller, D. D. Stancil, R. Rajkumar, and J. S. Parikh, *Groovesim: a topographyaccurate simulator for geographic routing in vehicular networks*, Proceedings of the 2nd ACM international workshop on Vehicular ad hoc networks, pp. 5968, 2005.

[9] A. Jardosh, E. M. Belding-Royer, K. C. Almeroth, and S. Suri, *Towards realistic mobility models for mobile ad hoc networks*, Proceedings of the 9th annual international conference on Mobile computing and networking, pp. 217–229, 2003.

[10] A. K. Saha and D. B. Johnson, *Modeling mobility for vehicular ad-hoc networks*, Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks, pp. 91–92, 2004.

[11] N. Potnis and A. Mahajan, *Mobility models for vehicular ad hoc network simulations*, Proceedings of the 44th annual Southeast regional conference, pp. 746–747, 2006.

[12] R. Baumann, S. Heimlicher, and M. May, *Towards realistic mobility models for vehicular ad-hoc networks*, In Mobile Networking for Vehicular Environments, pp. 73–78, 2007.

[13] P. Dey, S. Chandra, and S. Gangopadhaya, *Speed distribution curves under mixed traffic conditions*, Journal of transportation engineering, vol. 132, no. 6, pp. 475–481, 2006.

[14] *Lexington area travel data collection test, final report: Global positioning systems for personal travel surveys*, Battelle Memorial Institute, URL: `http://www.fhwa.dot.gov/ohim/lextrav.pdf`, 1997.

[15] J. Broch, D. A. Maltz, D. B. Johnson, Y.C. Hu, and J. Jetcheva, A performance comparison of multi-hop wireless ad hoc network routing protocols. In *MobiCom'98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 85-97. ACM Press, 1998.

[16] S. R. Das, C. E. Perkins, and E. E. Royer, Performance comparison of two on-demand routing protocols for ad hoc networks. In *INFOCOM (1)*, pages 3–12, 2000.

[17] T. Karagiannis, M. Molle, M. Faloutsos, and A. Broido, A nonstationary poisson view of internet traffic. In *INFOCOM 2004: Proceedings of twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1558 - 1569, 2004.

[18] L. Kleinrock, *Queueing Systems, Vol. II Computer Applications.* Wiley, New York, USA, 1976.

[19] T. V. Group, The network simulator—NS 2. URL: `http://www.isi.edu/nsnam/ns/` Page accessed as of May 30th, 2006.

[20] W. J. Hsu, K. Merchant, H. W. Shu, C. H. Hsu, and A. Helmy, Weighted waypoint mobility model and its impact on ad hoc networks. *SIGMOBILE Mobile Computer Communications Review*, 9(1): 59–63, 2005.

[21] D. Johnson and D. A. Maltz, Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[22] M. Kim, D. Kotz, and S. Kim, Extracting a mobility model from real user traces. In *INFORCOM: Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1–13, 2006.

[23] X. G. Meng, S. H. Y. Wong, Y. Yuan, and S. Lu, Characterizing flows in large wireless data networks. In *MobiCom'04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 174–186, New York, NY, USA, 2004. ACM.

[24] C. Perkins and E. Royer, Ad-hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.

[25] T. Rappaport, *Wireless Communications: Principles and Practice.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[26] A. Doci, L. Barolli, and F. Xhafa, *MixMobGen—a realistic Mixed traffic Mobility Generator for ad hoc network simulations* CISIS: Wireless and Mobile Networking, 2009.

[27] D. Kahaner, C. Moler, and S. Nash, *Numerical methods and software.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[28] Wireless lan medium access control (mac) and physical layer (phy)specifications. Technical report, IEEE Computer Society LAN MAN Standards Committee, 1997.

[29] J. Boleng, W. Navidi, and T. Camp, Metrics to enable adaptive protocols for mobile ad hoc networks. In *International Conference onWireless Networks*, pages 293–298, 2002.

Edited by: Fatos Xhafa, Leonard Barolli
Received: September 30, 2008
Accepted: December 15, 2008

Scalable Computing: Practice and Experience
Volume 10, Number 1, pp. 13–23. http://www.scpe.org

ISSN 1895-1767
© 2009 SCPE

SIMULATION FRAMEWORK FOR THE EVALUATION OF DEPENDABLE DISTRIBUTED SYSTEMS

CIPRIAN DOBRE[*], FLORIN POP[†], AND VALENTIN CRISTEA[‡]

Abstract. The use of discrete-event simulators in the design and development of distributed systems is appealing due to their efficiency and scalability. Their core abstractions of process and event map neatly to the components and interactions of modern-day distributed systems and allow designing realistic simulation scenarios. MONARC, a multithreaded, process oriented simulation framework designed for modelling large scale distributed systems, allows the realistic simulation of a wide-range of distributed system technologies, with respect to their specific components and characteristics. In this paper we present an innovative solution to the problem of evaluating the dependability characteristic of distributed systems. Our solution is based on several proposed extensions to the simulation model of the MONARC simulation framework. These extensions refer to fault tolerance and system orchestration mechanisms being added in order to asses the reliability and availability of distributed systems. The extended simulation model includes the necessary components to describe various actual failure situations and provides the mechanisms to evaluate different strategies for replication and redundancy procedures, as well as security enforcement mechanisms.

Key words: dependable distributed systems, grid computing, fault tolerance, simulation, performance analysis.

1. Introduction. Nowadays there is an increasing interest in large scale distributed systems, in both academic and industrial environments. Today more than ever distributed systems represent the preferred instruments for developing a wide range of new applications. Due to the offered technological opportunities, the domains of usage of Grid computing in particular have been extending during the past years from scientific to commercial applications.

Together with the extension of the application domains, new requirements have emerged for large scale distributed systems; among these requirements, reliability, safety, availability, security and maintainability, in other words dependability [1], are needed by more and more modern distributed applications, not only by the critical ones. However, building dependable distributed systems is one of the most challenging research activities. The characteristics of distributed systems make dependability a difficult problem from several points of view. A first aspect is the geographical distribution of resources and users that implies frequent remote operations and data transfers; these lead to a decrease in the system's safety and reliability and make it more vulnerable from the security point of view. Another problem is the volatility of the resources, which are usually available only for limited periods of time; the system must ensure the correct and complete execution of the applications even in the situations when the resources are introduced and removed dynamically, or when they are damaged.

The management of distributed systems is also complicated by the constraints that the applications and the owners of the resources impose; in many cases there are conflicts between these constraints. For example, an application needs a long execution time and performs database operations, while the owner of the machine on which the application could be run only makes it available in a restricted time interval and does not allow database operations.

Solving these issues still represents a research domain and, although numerous projects have obtained significant results, no complete solution has been fond yet to integrate all requirements involved in obtaining a

[*] Ciprian Dobre received his PhD in Computer Science at the University "Politehnica" of Bucharest in 2008. He received his MSc in Computer Science in 2004 and the Engineering degree in Computer Science in 2003, at the same University. His main research interests are Grid Computing, Monitoring and Control of Distributed Systems, Modeling and Simulation, Advanced Networking Architectures, Parallel and Distributed Algorithms. He is member of the RoGrid consortium and is involved in a number of national projects (CNCSIS, GridMOSI, MedioGRID, PEGAF) and international projects (MonALISA, MONARC, VINCI, VNSim, EGEE, SEE-GRID, EU-NCIT). His research activities were awarded with the Innovations in Networking Award for Experimental Applications in 2008 by the Corporation for Education Network Initiatives (CENIC). (ciprian.dobre@cs.pub.ro).

[†] Florin POP is a PhD student at Computer Science department in University "Politehnica" of Bucharest. His research interests are oriented to: scheduling in Grid environments (his PhD research), distributed system, parallel computation, communication protocols. He received engineering degree in Computer Science, in 2005, on Decentralized Scheduling Methods for Grid and MSc in 2006. He is member of RoGrid consortium and developer in the national projects (CNCSIS, GridMOSI, MedioGRID) and international projects (EGEE, SEE-GRID, EU-NCIT). (florin.pop@cs.pub.ro).

[‡] Valentin CRISTEA is a Professor at Computer Science department in University "Politehnica" of Bucharest. His main fields of expertise are Grid Computing, e-Learning, e-Business, e-Government, Distributed Systems and Web-based Application Development. He is the Director of the National Center for Information Technology. Prof. Valentin Cristea has a long experience in the development, management and/or coordination of research national and international projects on distributed computing, Grid computing, High Performance Computing, and e-Learning. (valentin.cristea@cs.pub.ro).

dependable system. The current research in this domain considers several aspects like control of complexity, early defect detection, rigorous translation, ease of simulation, model checking, productivity gains for teams, wide applicability [1]. It means collecting informal requirements, requirements translation, build requirement behavior trees, making requirements integration and integrated behavior tree, and then make simulation, verification and implementation [2]. In this paper we present a solution to evaluate the correctness and performance of various dependability-related technologies for distributed systems using the formalism provided by the modeling and simulation domain. Simulation is a powerful method to perform system analysis, even when the involved system does not physically exist or if it is very expensive to use it. Our proposed solution is based on an extension proposal of the simulation model being provided by MONARC, a generic simulation framework designed for modeling large scale distributed systems.

As described next, no existing simulator that addresses the evaluation of distributed systems related technologies is mature enough to consider the problem of dependability evaluation in the generic sense, considering all entities part of such systems, together with the corresponding characteristics. Our model considers the generic aspects of a wide-range of distributed architectures, preserving their components and characteristics. Because of this, the model can be used to validate a wide-range of dependability solutions, such as a networking protocol that can cope with faults in transmissions or a scheduling algorithm designed to consider the case of resource faults.

The rest of this paper is structured as follows. Section 2 presents related work to the problem of analyzing dependable distributed systems using simulation. The next section presents the MONARC architecture and the simulation model. Sections 4 and 5 present the proposed solutions for simulating dependable distributed systems. Finally, in Section 6 we present some conclusions and future work.

2. Related Work. SimGrid [7] is a simulation toolkit that provides core functionalities for the evaluation of scheduling algorithms in distributed applications in a heterogeneous, computational Grid environment. It aims at providing the right model and level of abstraction for studying Grid-based scheduling algorithms and generates correct and accurate simulation results.

GridSim [8] is a grid simulation toolkit developed to investigate effective resource allocation techniques based on computational economy.

OptorSim [9] is a Data Grid simulator project designed specifically for testing various optimization technologies to access data in Grid environments. OptorSim adopts a Grid structure based on a simplification of the architecture proposed by the EU DataGrid project.

ChicagoSim [10] is a simulator designed to investigate scheduling strategies in conjunction with data location. It is designed to investigate scheduling strategies in conjunction with data location.

None of these projects present general solutions to modeling dependability technologies for large scale distributed systems. They tend to focus on providing evaluation methods for the traditional research in this domain, which up until recently targeted the development of functional infrastructures. However, lately, the importance of dependable distributed systems was widely recognized and this is demonstrated by the large number of research projects initiated in this domain. Our solution aims to provide the means to evaluate a wide-range of solutions for dependability in case of large scale distributed systems.

Another issue is related to the generic evaluation of dependable distributed systems. A fault occurring in such systems could lead to abnormal behavior of any of the system's components. For this reason we argue that a correct evaluation of dependability in distributed systems should provide a complete state of the entire distributed system.

Because of the complexity of the Grid systems, involving many resources and many jobs being concurrently executed in heterogeneous environments, there are not many simulation tools to address the general problem of Grid computing. The simulation instruments tend to narrow the range of simulation scenarios to specific subjects, such as scheduling or data replication. The simulation model provided by MONARC is more generic that others, as demonstrated in [11]. It is able to describe various actual distributed system technologies, and provides the mechanisms to describe concurrent network traffic, to evaluate different strategies in data replication, and to analyze job scheduling procedures.

3. A taxonomy of distributed system-based simulation tools. Simulation is applied in many application areas, one of which is the design and evaluation of parallel and distributed systems (PDSs). Simulation tools for PDSs are then differentiated by four taxonomies [5]. Figure 3.1 provides a categorization of simulation tools. PDS taxonomy identifies the type of target systems to be simulated; usage taxonomy illustrates how the tool is used; simulation taxonomy highlights the characteristics of the simulation; design taxonomy describes the components and features of the simulation tools.

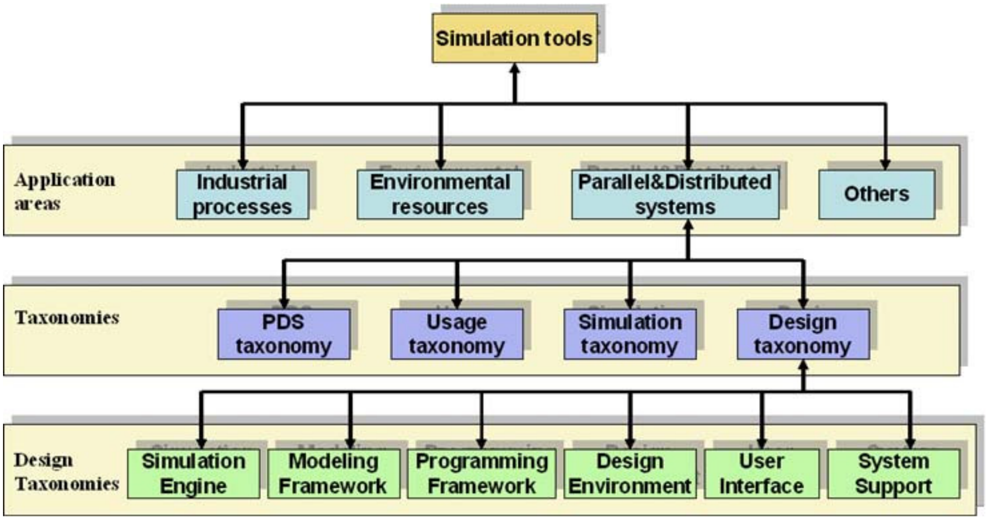


Fig. 3.1. *Categorization of simulation tools.*

The PDS taxonomy, presented in Figure 3.2, separates the target systems to be simulated into parallel systems and distributed systems. A parallel system consists of multiple processors in close communication, normally located within the same machine.

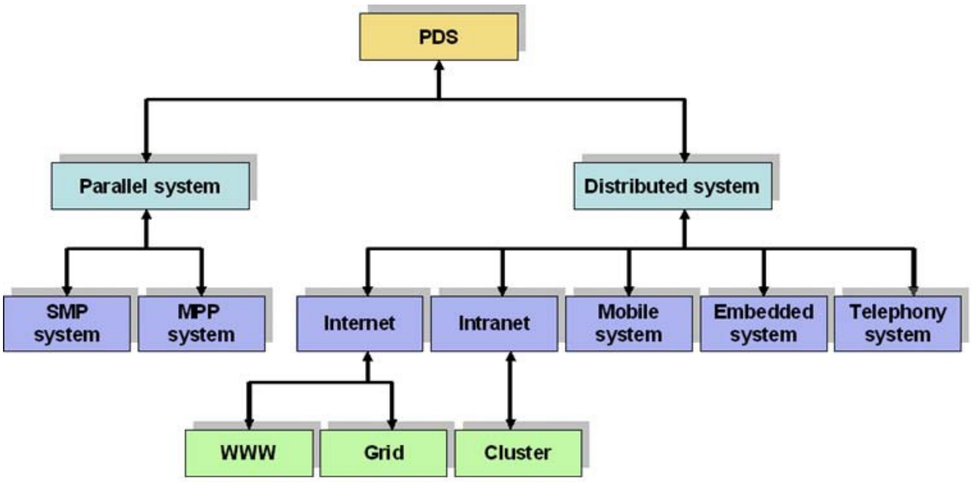


Fig. 3.2. *PDS taxonomy.*

A distributed system spreads out the computation among autonomous computers, which are physically distributed in general. Each distributed computer accesses its own local memory and communicates with other computers through networks such as the Internet. A distributed system supports both resource sharing and load sharing, and is fault-tolerant through the replication of devices and data in separate physical locations. It also provides a good price/performance ratio. Simulation tool can be used as a simulator or an emulator (see Figure 3.3).

A simulator is a tool that can model and represent the actual system. Simulation runs at any speed relative to the real world and saves information for the entire simulation to facilitate analysis of the simulated behavior

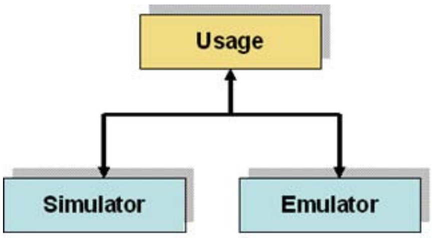


Fig. 3.3. *Usage taxonomy.*

of the actual system. Conversely, an emulator is a tool that acts like the actual system. Emulation executes like the actual system itself and is useful for accurate and reliable testing without having the real system. For example, the Grid simulation tools described in the next sections are simulators, except MicroGrid [6], which emulates Globus-based Grid applications.

The characteristics of the simulation highlighted by the simulation taxonomy (see Figure 3.4) consist of three main properties. Presence of time indicates whether the simulation of a system encompasses the time factor. A static simulation does not have time as part of the simulation, in contrast to a dynamic simulation. Basis of value specifies the values that the simulated entities can contain. A discrete simulation has entities only possessing one of many values within a finite range, but a continuous simulation has entities possessing one of many values within an infinite range. Behavior defines how the simulation proceeds. A deterministic simulation has no random events occurring, so repeating the same simulation will always return the same simulation results. In contrast, a probabilistic simulation has random events occurring, so repeating the same simulation often returns different simulation results.

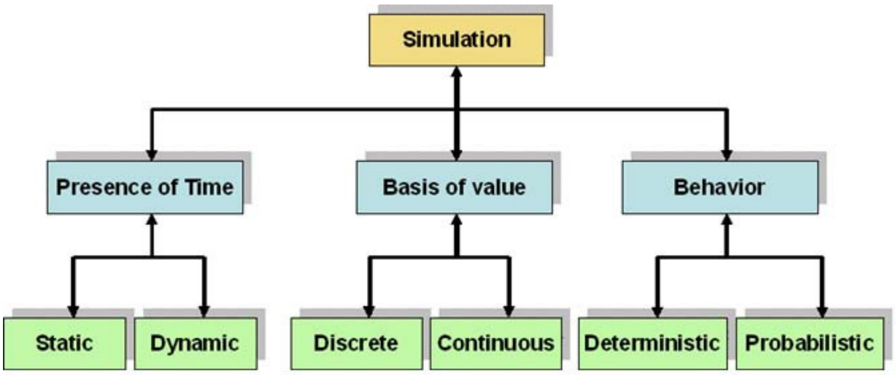


Fig. 3.4. *Simulation taxonomy.*

Every simulation can be classified based on these three properties. An example of a dynamic, discrete and probabilistic simulation is to generate a path that a data packet moves from a source host to a destination host in a network. However, simulating the path of a missile given an initial firing velocity and fixed wind resistance is an example of a dynamic, continuous and deterministic simulation. A chess game simulation comprises static, discrete and probabilistic properties.

Finally the design taxonomy (Figure 3.1) categorizes simulation tools based on desired components and features that are necessary to provide simulation of a PDS. These features and components not only provide a framework for understanding the design of existing simulation tools, but also provide possible ways of improving the design.

4. MONARC Architecture. MONARC is built based on a process oriented approach for discrete event simulation, which is well suited to describe concurrent running programs, network traffic as well as all the stochastic arrival patterns, specific for such type of simulations [3]. Threaded objects or “Active Objects” (having an execution thread, program counter, stack…) allow a natural way to map the specific behavior of distributed data processing into the simulation program. However, due to various optimizations considered, the threaded implementation of the simulator can be used to experiment with scenarios consisting of thousands of processing nodes executing a large number of concurrent jobs, as demonstrated in [12].

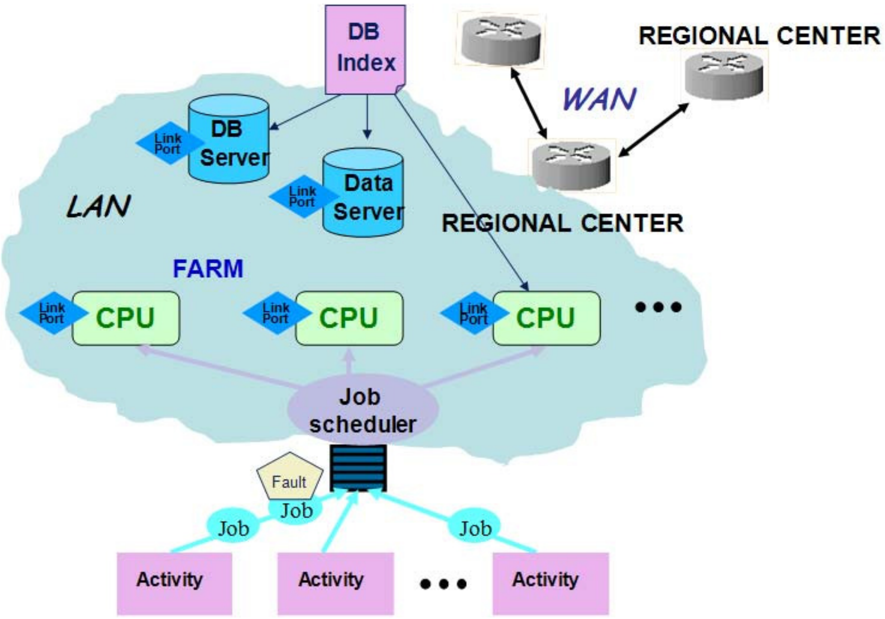


Fig. 4.1. *The Regional center model.*

In order to provide a realistic simulation, all the components of the system and their interactions were abstracted. The chosen model is equivalent to the simulated system in all the important aspects. A first set of components was created for describing the physical resources of the distributed system under simulation. The largest one is the regional center (Figure 4.1), which contains a site of processing nodes (CPU units), database servers and mass storage units, as well as one or more local and wide area networks. Another set of components model the behavior of the applications and their interaction with users. Such components are the “Users” or “Activity” objects which are used to generate data processing jobs based on different scenarios.

The job is another basic component, simulated with the aid of an active object, and scheduled for execution on a CPU unit by a “Job Scheduler” object. Any regional center can dynamically instantiate a set of users or activity objects, which are used to generate data processing jobs based on different simulation scenarios. Inside a regional center different job scheduling policies may be used to distribute jobs to corresponding processing nodes.

One of the strengths of MONARC is that it can be easily extended, even by users, and this is made possible by its layered structure. The first two layers contain the core of the simulator (called the “simulation engine”) and models for the basic components of a distributed system (CPU units, jobs, databases, networks, job schedulers etc.); these are the fixed parts on top of which some particular components (specific for the simulated systems) can be built. The particular components can be different types of jobs, job schedulers with specific scheduling algorithms or database servers that support data replication. The diagram in Figure 4.2 presents the MONARC layers and the way they interact with a monitoring system. In fact, one other advantage that MONARC have over other existing simulation instruments covering the same domain is that the modeling experiments can use real-world data collected by a monitoring instrument such as MonALISA, an aspect demonstrated in [13]. This is useful for example when designing experiments that are evaluating the behavior of existing real distributed infrastructures using various assumed theoretical conditions.

Using this structure it is possible to build a wide range of models, from the very centralized to the distributed system models, with an almost arbitrary level of complexity (multiple regional centers, each with different hardware configuration and possibly different sets of replicated data).

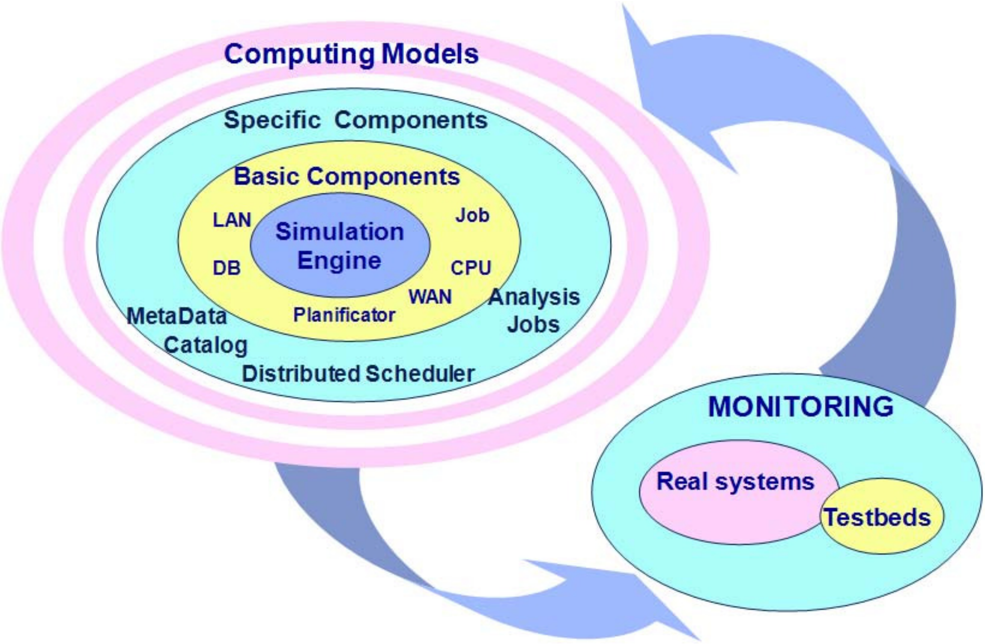


FIG. 4.2. *The layers of MONARC.*

The superior performance of the simulator was demonstrated in terms of experimenting with a large number of concurrent threads, evaluation of the interrupt model and in terms of time needed to conduct simulation experiments of large scale [11]. For example, Figure 4.3 presents the performance results of the simulator in terms of concurrent threads. The series represents the different architectures on which the evaluation experiment was conducted. In MONARC all the jobs being concurrently processed by the same processing node are handled by one thread, and all the messages being concurrently generated from the same host are also managed by one thread. The obtained results demonstrate the capability of the simulator to cope with experiments describing distributed systems consisting of thousands of processing nodes and concurrent network links.

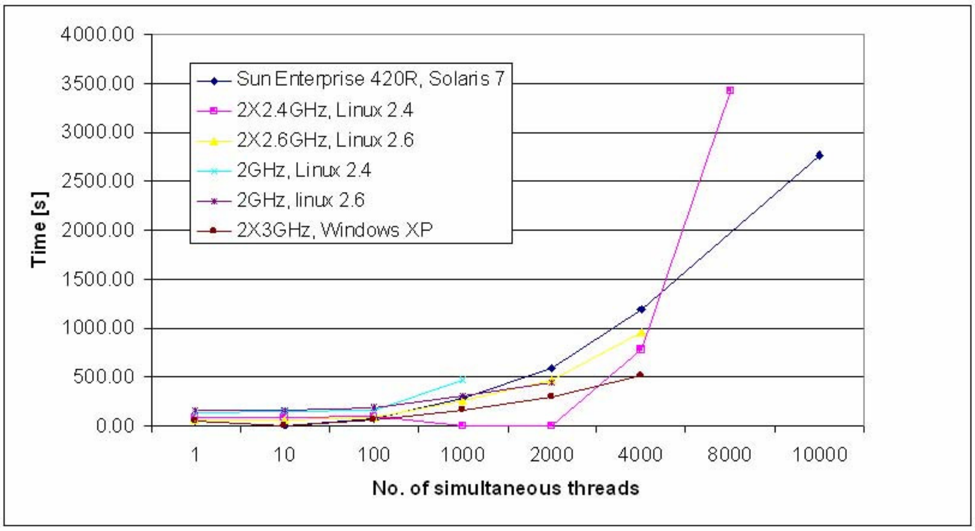


FIG. 4.3. *Performance of the MONARC simulator in terms of concurrent threads*

The maturity of the simulation model was demonstrated in previous work. For example, a number of data replications experiments were conducted in [3], presenting important results for the future LHC (Large Hadron

Collider) experiments, which will produce more than 1 PB (petabyte) of data per experiment and year, data that needs to be then processed. A series of scheduling simulation experiments were presented in [3], [4] and [5]. In [5] we successfully evaluated a simple distributed scheduler algorithm, proving the optimal values for the network bandwidth or for the number of CPUs per regional centre. In [4] the simulation model was used to conduct a series of simulation experiments to compare a number of different scheduling algorithms.

Probably the most extensive simulation scenario is the one described in [6]. The experiment tested the behavior of the tier architecture envisioned by the two largest LHC experiments, CMS and ATLAS. The simulation study described several major activities, concentrating on the data transfer on WAN between the T0 at CERN and a number of several T1 regional centers. The experiment simulated a number of physics data production specific activities (the RAW data production, Production and DST distribution, the re-production and the new DST distribution and the detector analysis activity). We simulated the described activities alone and then combined. The obtained results indicated the role of using a data replication agent for the intelligent transferring of the produced data. The obtained results also showed that the existing capacity of 2.5 Gbps was not sufficient and, in fact, not far afterwards the link was upgraded to a current 40 Gbps.

5. Simulation of Failure Models in MONARC. The characteristics of large scale distributed systems make the problem of assuring dependability a difficult issue because of several aspects. A first aspect is the geographical distribution of resources and users that implies frequent remote operations and data transfers; these lead to a decrease in the system's safety and reliability and make it more vulnerable from the security point of view. Another problem is the volatility of the resources, which are usually available only for limited periods of time; the system must ensure the correct and complete execution of the applications even in situations such as when the resources are introduced and removed dynamically, or when they are damaged. Another issue relates to the management of distributed resources. In many cases these are conflicts between the constraints that the applications and owners of the resources impose. One could consider, for example, the case when the execution needs of an application are sometimes conflicting with the restrictions imposed by the owners of the machines on which it should execute.

Solving these issues still represents a research domain and, although numerous projects have obtained significant results, no complete solution has yet been found to integrate all requirements involved in obtaining a dependable system.

The means to achieve dependability are fault prevention, fault tolerance, fault removal and fault forecasting [1]. Among these fault tolerance is the most difficult to achieve because of the complexity of the distributed systems. We consider that the simulation model of MONARC is adequate to be augmented as presented in this paper to testing various methods, techniques and technologies related to the different phases of fault tolerance achievement in distributed systems: error and fault detection, recovery and fault masking (redundancy) [2]. Figure 5.1 presents the components of the proposed dependable modeling layer.

The first extension to the simulation model relates to modeling faults appearing inside the modeled distributed system. In a distributed system failures can be produced at hardware level (abnormalities in the functionality of hardware components) or software level (the middleware or application deviating from their normal functionality or delivery of services). We propose extending the simulation model to account for both hardware, as well as software failures, modeling their occurrences and detection, as well as recovery and masking (redundancy) mechanisms [1].

At hardware level different distributed components can be modeled as failing: the processing unit, the network connectivity as well as the storage devices. At software level we consider the faults occurring in a middleware component (the scheduler behavior could be erroneous, the database server could return wrong values, etc.) or in the higher-level distributed application (for example the jobs could fail to return correct results). For all the modeled components being considered by the simulation model we propose adding the mechanisms to inject faults.

The injection of faults will affect primarily the behavior of the components. In this way we will be able to model all types of faults: crash faults, omission faults, time faults, as well as Byzantine faults [2]. In order to cover all possible faults, we propose the use of two mechanisms, as follows.

In the first approach the user will input into the simulation model values for the MTTF (mean time to failure) parameter in case of the various components involved in a specific simulation experiment. In the simulation this parameter represents the basis for simulating the haphazardness stimuli coming from external and internal sources of influence that affect the characteristics of the modeled components and is seen as a probability measure of fault occurrence that is supposedly computed for each component prior to its deployment in the system.

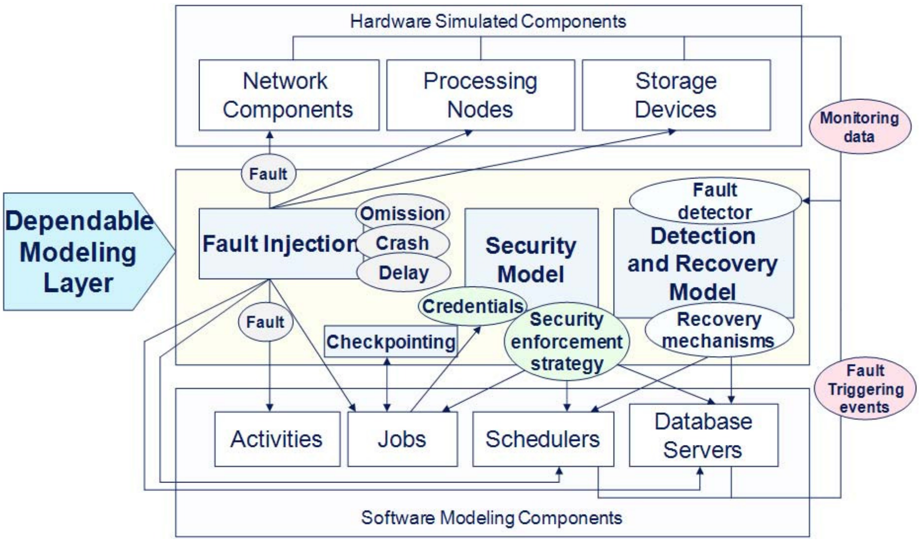


Fig. 5.1. *The layers of MONARC.*

For modeling the fault injection mechanisms we will use the MTTF together with some mathematical probability distributions (the simulation model already uses distributions such as binomial, Poisson, Gaussian, standard uniform, exponential to implement the stochastic behavior of distributed systems). In this way, at random intervals of time, a component will experience faulty behavior (failures), as well as possible recovery. Regarding the failures, a component will experience complete crash (the component will not recover anymore and will not be accessible by modeled entities), omissions (the network model will deliver only partial messages for example) or delays (the component will experience modeled delays).

The second proposed approach considers a completely random occurrence of fault events, without the user specifying any value. This is useful in modeling the most disruptive faults, the Byzantine failures that occur arbitrary in the simulated system, such as in the case of transient hardware faults for example: they appear, do the damage, and then apparently just go away, leaving behind no obvious reason for their activation in the first place [14]. For these types of errors the simulation model will allow the resuming of the normal behavior of the affected component, as it is usually enough to allow successful completion of its execution.

This is also useful for example when modeling the intervention of the human operator restarting or replacing a faulty component. For modeling this behavior the user usually inputs into the simulation model a MTTR (mean time to repair) value that, together with a probability distribution, represents the basis of a triggering mechanism that allows the modeling of a resume operation event. In addition, in order to model a wider range of possible fault-tolerance based scenarios, a second approach to resuming will consider that the timestamp of the resuming event can also be based on a completely random decision.

A different faulty situation that needs to be addressed is represented by users or programs trying to gain access to secured components.

For modeling security technologies we propose adding an extra layer that injects security problems. This layer includes the mechanisms that are used to construct a special job that communicates with various components in the attempt to gain access to restricted data or evaluate security technologies (single authentication, credentials protection, interoperability with local security solutions, etc.).

6. Simulation of Dependable Distributed System Technologies using MONARC. The fault injection mechanisms presented above will be used together with various fault detection and recovery mechanisms. For that we propose the addition of an external monitoring component in the simulation model. The monitoring component will be responsible for receiving data of interest (such as fault occurrence triggered events), for taking actions to update the state of the distributed system being modeled and possible inform interested components of the events occurrences.

For the purpose of the current proposal one of the roles of this component will be to keep track of the resource faults in the system and generate appropriate controlling actions. For example, as described before, based on various fault injection mechanisms, we could trigger the generation of a crash of a processing unit. The trigger will translate in the generation of a special simulation event that will be received by the monitoring component. Upon receive of such event the monitoring unit will remove the processing unit from the modeled system (it will no longer be visible and all its on-going tasks will be forced to stop the state update action) and inform all interested components of the occurrence of the event. In this approach a scheduler that is interesting in monitoring the state of the processing unit on which it deployed a task for execution would register with the monitoring component and, upon triggering of the crash event, will be informed of the failure so that to take the appropriate correction actions). In this model an omission fault example is the simulation of a crushed network link, where the monitoring unit will be responsible with generating corresponding interrupt events for the still-running tasks that were using that modeled link.

For modeling timing faults the monitoring unit will also play an important role. In the simulation model the boundaries of the action (start of the execution of a task, termination of a task, etc.) are modeled using simulation events. When the monitoring unit receives a timing fault triggering event it will simply modify termination events so that to be triggered at a later time in the future. In this way we simply change the default behavior so that a task will not end when it was supposed to end, but sometimes later in the future.

A modeled fault-tolerant software component could then use timing checks (deadlines) to look for deviations from the acceptable behavior. For example, the scheduler will also implement a fault-tolerant mechanism, according to which whenever a new job is submitted the scheduler will also produce a special simulation event that will be triggered when the timeout occurs (where by timeout we mean an amount of time dependant on the user specification that will be triggered if the job fails to return results in due time).

The scheduler will then be interrupted by one of two actions: either the job finishes or the timeout event occurs (which one happens faster in the simulation timeline). The same mechanisms will be implemented to the network simulation model, a job being informed if a transfer failed to finished in a specified amount of time (possible due to network congestion for example) in order to consider taking appropriate measures (such as canceling the transfer for example).

We state that the described extension to MONARC's simulation model will be useful for testing both reactive and proactive fault tolerance existing techniques [15]. In case of the reactive fault tolerant systems the fault monitoring systems is informed after the detection of the failure in order to start the corresponding recovery processes. This is in concordance with the presented functionality of the proposed modeled monitoring component. The predictive (proactive) fault tolerant systems predict a fault before its occurrence. Predictive systems are based on the assumption that faults do show some disruptive effect on the performance of the system [16]. Much effort has been put recently into developing proactive fault tolerant systems [17], [18] and [19]. In our approach the user could evaluate the performance of the distributed system in the presence of various predictive fault tolerance techniques by augmenting the monitoring component with various prediction algorithms.

In order to allow the evaluation of a wide-range of dependability technologies, the simulation model will also include the necessary mechanisms to allow the modeling of check-pointing or logging of the system's state. These mechanisms will be implemented based on the simulation events and the state of the objects used to simulate the modeled components. For that the job interface will provide a method that, when called, will result in the saving of the serialized objects as well as the state of the simulation on the local disk storage. This will be useful in experiments dealing with both static and dynamic check-pointing strategies.

The simulation model will also include the evaluation of various replication and redundancy mechanisms. The replication provides mechanisms for using multiple identical instances of the same system or subsystems and choosing the result based on quorum. The simulation model already allows the simulation of DAG distributed activities. This construction is also useful in modeling the replication of the jobs, where the same job would be executed on multiple processing units and another job will be used to receive the outputs and select the correct result. Another approach could be to model special voter jobs acting at the input of the replicated jobs and selecting the correct results (as the one described in [15]). We propose augmenting the simulation model with these approaches as well.

The redundancy is another characteristic that is important to be considered when evaluating dependable distributed system strategies. The fault-tolerance techniques applicable to software, either single version of multi-version, are based on their vast majority on the use of redundancy, on the use of resources beyond the minimum needed to deliver the specified services [19]. For example, a strategy to achieve reliability could be that when a storage component fails the system would select a different replica of that storage still functioning and query it for the data of interest. This is already possible in MONARC, as demonstrated by the experiments described in [3]. In the experiments we describe how the simulation model already deals with replicating database storages. We propose to add replication mechanism in case of the simulated jobs and scheduler as well.

The simulation of security technologies also plays an important part in assessing dependable systems. We propose adding mechanisms to evaluate various security procedures that are essential in dependable distributed systems. These mechanisms refer to providing availability for authorized users only, confidentiality and integrity.

The networking model being provided by MONARC can be easily extended to provide the modeling of all these aspects. The message being transferred among simulated entities can carry inside a wide-range of information. This can be used to implement various authorization enforcement mechanisms, such as for example certificates that are provided as input to the modeling experiments and that are automatically verified upon the receipt of the message. We propose extending the network and job models to incorporate several such security technologies.

We also propose adding an extra layer to the simulation model, a security layer that could be used to simulate various security-related procedures. The security layer will provide mechanisms to encrypt the data that could be used in conjunction with the storage components for example. It will also include various security protocols (such as SSL and TLS for example, or group key exchange protocols) that could be used of-the-shelf by users trying to run their own experiments in the presence of various security mechanisms.

7. Conclusions. As society increasingly becomes dependent of distributed systems (Grid, P2P, network-based), it is becoming more and more imperative to engineer solutions to achieve reasonable levels of dependability for such systems. Simulation plays an important part in the building process of dependable distributed systems.

MONARC is a generic simulation framework designed for modeling a wide range of distributed systems technologies, with respect to their specific components and characteristics. Its simulation model incorporates the necessary components to model various modern-day distributed systems technologies, providing the mechanisms to describe concurrent network traffic and to evaluate different strategies in data replication or in the job scheduling procedures. In this paper we described a solution to simulating dependable distributed systems using MONARC. The solution is able to model failures in distributed systems at hardware level (abnormalities in the functionality of hardware components) or software level (the middleware or application deviating from their normal functionality or delivery of services). We aim at extending MONARC to account for types of failures (at hardware and software levels), modeling their occurrences and detection, as well as recovery and masking (redundancy) mechanisms.

REFERENCES

[1] A. Avizienis, J. C. Laprie, B. Randell, *Fundamental Concepts of Dependability,* Research Report No 1145, LAAS-CNRS, April 2001.

[2] A. Avizienis, V. Magnus, J. C. Laprie, B. Randell, *Fundamental Concepts of Dependability,* presented at ISW-2000, Cambridge, MA, 2000.

[3] I. C. Legrand, H. Newman, C. Dobre, C. Stratan, *MONARC Simulation Framework,* International Workshop on Advanced Computing and Analysis Techniques in Physics Research, Tsukuba, Japan, 2003.

[4] F. Pop, C. Dobre, G. Godza, V. Cristea, *A Simulation Model for Grid Scheduling Analysis and Optimization,* Proc. of Parelec, Bialzsrok, Poland, pp. 133–138, September, 2006.

[5] C. Dobre, C. Stratan, *MONARC Simulation Framework,* Proc. of 3rd Edition of RoEduNet International Conference, Timisoara, Romania, May, 2004.

[6] I. C. Legrand, C. Dobre, R. Voicu, C. Stratan, C. Cirstoiu, L. Musat, *A Simulation Study for T0/T1 Data Replication and Production Activities,* The 15th International Conference on Control Systems and Computer Science, Bucharest, Romania, pp. 127–130, 2005.

[7] H. Casanova, A. Legrand, M. Quinson, *SimGrid: a Generic Framework for Large-Scale Distributed Experimentations,* Proc. of the 10th IEEE International Conference on Computer Modelling and Simulation (UKSIM/EUROSIM'08), pp. 126–131, 2008.

[8] R. Buyya, M. Murshed, *GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing,* The Journal of Concurrency and Computation: Practice and Experience (CCPE), Volume 14, pp. 1507–1542, 2002.

[9] W. Venters et al, *Studying the usability of Grids, ethongraphic research of the UK particle physics community,* UK e-Science All Hands Conference, Nottingham, 2007.

[10] K. Ranganathan, I. Foster, *Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications,* Int. Symposium of High Performance Distributed Computing, Edinburgh, Scotland, p. 352, 2002.

[11] C. Dobre, V. Cristea, *A Simulation Model for Large Scale Distributed Systems,* Proc. of the 4th International Conference on Innovations in Information Technology, Dubai, United Arab Emirates, pp. 526–530, November 2007.

[12] C. Dobre, *Advanced techniques for modeling and simulation of Grid systems,* PhD Thesis presented to University Politehnica of Bucharest, January 2008.

[13] C. Dobre, C. Stratan, V. Cristea, *Realistic simulation of large scale distributed systems using monitoring,* in Proc. of the 7th International Symposium on Parallel and Distributed Computing (ISPDC 2008), Krakow, Poland, July 2008.

[14] J. Gray, *Why Do Computers Stop and What Can Be Done About It?* Proc. of the Fifth Symposium On Reliability in Distributed Software and Database Systems, Los Angeles, CA, pp. 3-12, 1986.

[15] N. Naik, *Simulating Proactive Fault Detection in Distributed Systems,* Proposal Draft for Capstone Project, 2007.

[16] T. A. Dumitras, P. Narsimhan, *Fault-Tolerant Middleware and the Magical 1 percent,* ACM/IFIP/USENIX Conference on Middleware, Grenhole, France, 2005.

[17] L. Yawei, Z. Lan, *Exploit Failure Prediction for Adaptive Fault-Tolerance in Cluster Computing,* Proc. of the sixth IEEE International Symposium on Cluster Computing and Grid, Vol. 1, p. 8, May 2006.

[18] P. Narasimhan, R. Rajkumar, G. Thaker, P. Lardieri, *A Versatile, Proactive Dependability Approach to Handling Unanticipated Events in Distributed Systems,* Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium, pp. 136a, April, 2005.

[19] W. Torres-Pomales, *Software Fault Tolerance: A Tutorial,* Langley Research Center, Hampton, Virginia, NASA/TM-2000-210616, October 2000.

Edited by: Fatos Xhafa, Leonard Barolli
Received: September 30, 2008
Accepted: December 15, 2008

Scalable Computing: Practice and Experience
Volume 10, Number 1, pp. 25–34. http://www.scpe.org

ISSN 1895-1767
© 2009 SCPE

IMPACT OF THE DYNAMIC MEMBERSHIP IN THE CONNECTIVITY GRAPH OF THE WIRELESS AD HOC NETWORKS

ARTA DOCI[*], WILLIAM SPRINGER[†], AND FATOS XHAFA[‡]

Abstract. Mobility Metrics of the connectivity graphs with fixed number of nodes have been presented in several research studies over the last three years. More realistic mobility models that are derived from real user traces challenge the assumption of the connectivity graph with fixed number of nodes, but they rather show that wireless nodes posses dynamic membership (nodes join and leave the simulation dynamically based on some random variable). In this paper, we evaluate the mobility metrics of the dynamic connectivity graph on both the nodes and the links. The contributions of this paper are two-fold. First, we introduce the algorithm that computes the Maximum Node Degree. We show that the Maximum Node Degree, which characterizes the dynamic membership property, impacts the connectivity mobility metrics of the Link and Path Durations of the dynamic connectivity graphs. Second, we provide the lower and the upper bounds for these mobility metrics.

Key words: wireless ad hoc networks, mobility metrics, connectivity, simulation.

1. Introduction. Many studies have demonstrated that mobility has a significant impact on the performance of ad hoc network protocols. Specifically, the authors of [5] provide a framework, which is helpful in understanding *how* and *what* mobility metrics affect the protocol performance. For example, the mobility metrics that effect the performance are average shortest path hop count ($A_{sp}Hops$) [11], average link durations [2] and average path durations [13]. In addition, other path and link metrics have been proposed by the authors of [7, 9, 15] and have been shown to effect performance. All these mobility metrics are derived from synthetic mobility models, which face two main issues.

First, the connectivity graph of the synthetic mobility models, where the mobile nodes are the vertices and the communication links are the edges, have been assumed to *have static number of mobile nodes for the simulation duration*. The number of vertices $|V|$ represent the order and the number of edges $|E|$ represent the size of the connectivity graph. In general, the running time of the algorithms are measured in terms of the order and size of the graph, which is relatively easy to estimate when the number of nodes does not change, but becomes a cumbersome task when the graph is dynamic on both the vertices and edges.

Second, the current implementations of the synthetic mobility models place the wireless nodes to start the simulation at time 0 and remain in the simulation until the allotted simulation time is over. On the other hand, real mobility models [16], which are extracted from real user traces, show that wireless nodes posses dynamic membership, that is, they join and leave the simulation based on an exponentially distributed random variable. Under dynamic membership of the nodes, the connectivity graph is *dynamic on the number of vertices and edges*. When adding the new dimension of the dynamic membership to simulation mobility models, then the mobility metrics need to be re evaluated under the new dimension. The focus of this paper is to assess the mobility metrics of the dynamic connectivity graph.

Maximum Node Degree mobility metric, which represents the maximum number of neighbors for each node (in graph theory terms it is the number of edges incident to it), can be used to account for the dynamic membership of the wireless nodes. The contributions of this paper are:

1. Design and implement algorithms that compute the Maximum Node Degree, Average Links, and Average Path Durations mobility metrics for dynamic topology connectivity graphs.
2. Present the lower and upper bounds for the Maximum Node Degree, Link Durations, and Path Durations mobility metrics.

2. RealMobGen: A More Realistic Mobility Model. The main characteristics of a mobility model are speed, pause distribution, and direction of movement. For example, the most used synthetic mobility model is Random Walk Model (RWM) [3]. In RWM each node is assigned a randomly distributed initial location $(x0; y0)$. Then, each node randomly picks up a destination independent of their initial positions and moves toward it with speed chosen uniformly on the interval $(v0; v1)$. Nodes pause upon reaching each destination. The process is repeated until the allotted simulation time is reached. There are several issues with RWM, which are addressed in RealMobGen.

[*] Colorado School of Mines, Colorado, USA (adoci@mines.edu).
[†] Colorado State University, Colorado, USA (wmspring@CS.ColoState.EDU).
[‡] Technical University of Catalonia, Barcelona, Spain (fatos@lsi.upc.edu).

RealMobGen is a hybrid model that is based on Dartmouth's model of mobile network traces [10] and USC's WWP [6] survey collected from the students on campus. The model closely mimics the environments where ad hoc networks will likely be deployed, since it borrows its characteristics from models derived from real user traces. Another feature of RealMobGen, that is not existent in any other current mobility model, is the classification of nodes as stationary (46% of the nodes) and mobile (54% of the nodes). The ratio of stationary vs. mobile nodes was borrowed from the Dartmouth model.

The stationary nodes select a location based on a transition matrix that defines the probabilities for moving from one point to another. Once a location is selected, a node is turned on for a time drawn from the exponential distribution of start time for the stationary nodes. Stationary node stays at the selected location until the allotted stationary end time. The mobile nodes, also, select a start location based on the transition matrix. The mobile node enters the simulation at a time drawn from the mobile node start time. The node pauses at the selected location until the allotted pause time from mobile pause time exponential distribution. After the pause time is up, the mobile node selects the next location based on the transition matrix and moves there not in a straight line but following data that supports movements along popular roads and turns. The mobile node repeats the pattern 'pause-select next location - move there' until allotted mobile simulation end time.

RealMobGen shows that wireless nodes tend to cluster around popular locations, i. e., cafeteria, gym, classes, and library. We believe that RealMobGen is the first mobility model (we are not aware of any other models) that implements the dynamic characteristic of wireless devices in NS 2, devices join and leave the network at different times. RealMobGen addresses the drawbacks present in the RWM by implementing the following new features:

Feature 1: Wireless Nodes are clustered around popular hotspots. For example, Figure 2.1 shows a snapshot of RealMobGen with 100 nodes, which are clustered around 14 hotspots.

Feature 2: Wireless Nodes posses dynamic membership. For example, Figure 2.2 shows the dynamic membership of 60 nodes.

Feature 3: Nodes are classified on two flavors, namely stationary and mobile (stationary (46%)of the nodes and mobile (54% of the nodes). The ratio of stationary vs. mobile nodes was borrowed from the Dartmouth model.

Feature 4: Moving from one point to another is done via waypoints, instead of a straight line.

3. Connectivity graph and Mobility Metrics. In more realistic mobility models the connectivity graph $G = (V, E)$ is a dynamic graph on vertices that join and leave and edges that appear and disappear due to mobility. In graph theory terms, when taking into account the dynamic membership of the nodes, there is an edge between any two nodes (i, j), if the following two conditions are met:

1. $D_{i,j} \leq R$, where D is the euclidian distance between nodes (i, j) and R is the transmission range of the wireless nodes.
2. $I_i \cap I_j \neq \emptyset$, where I represents the active intervals of the nodes.

Thus, when interested on the mobility metrics, i. e., path and link durations, we are basically posing the shortest path problem in the case of the dynamic topology and edges. Basically, this can be represented as a graph with edges that are added and removed, as the vertices turn 'on' and 'off'. While the method of calculating the full graph is polynomial in the size of the input, it is quite inefficient as each movement of a node can potentially cause $n - 1$ edge insertions and the same number of deletions. We can obtain better bounds by restricting the algorithm to only *processing the nodes, which are currently relevant.*

In order to consider only the nodes that are relevant, first we process the dynamic membership information. For example, consider two nodes i and j. Since all communication is instantaneous, any communication between the nodes must occur while both nodes are turned on. If the communication is indirect, then every node that participates in routing the messages must also be turned on during that period of time. While, we should note that, this can be $O(n)$ nodes in the worst case, in most cases this allows us to consider only a fraction of the vertices of the full graph (See Section *Maximum Node Degree Mobility Metric*).

3.1. Dynamic Membership Graph is an Interval Graph. Given a set of n intervals (which represent the number of wireless nodes) $I = I_1, I_2, \ldots, I_n$, the corresponding interval graph $G = (V, E)$ has the set of vertexes $V = v_1, v_2, \ldots, v_n$ and there is an edge E linking nodes (v_i, v_j) if and only if $I_i \cap I_j \neq \emptyset$ (See Figure 3.1). We will use the interval graphs to calculate the $MaximumNodeDegree$.

4. Maximum Node Degree Mobility Metric. In order to define the upper bounds of the number of nodes that are neighbors, we formulate the problem as a coloring problem, where the goal of the algorithm is

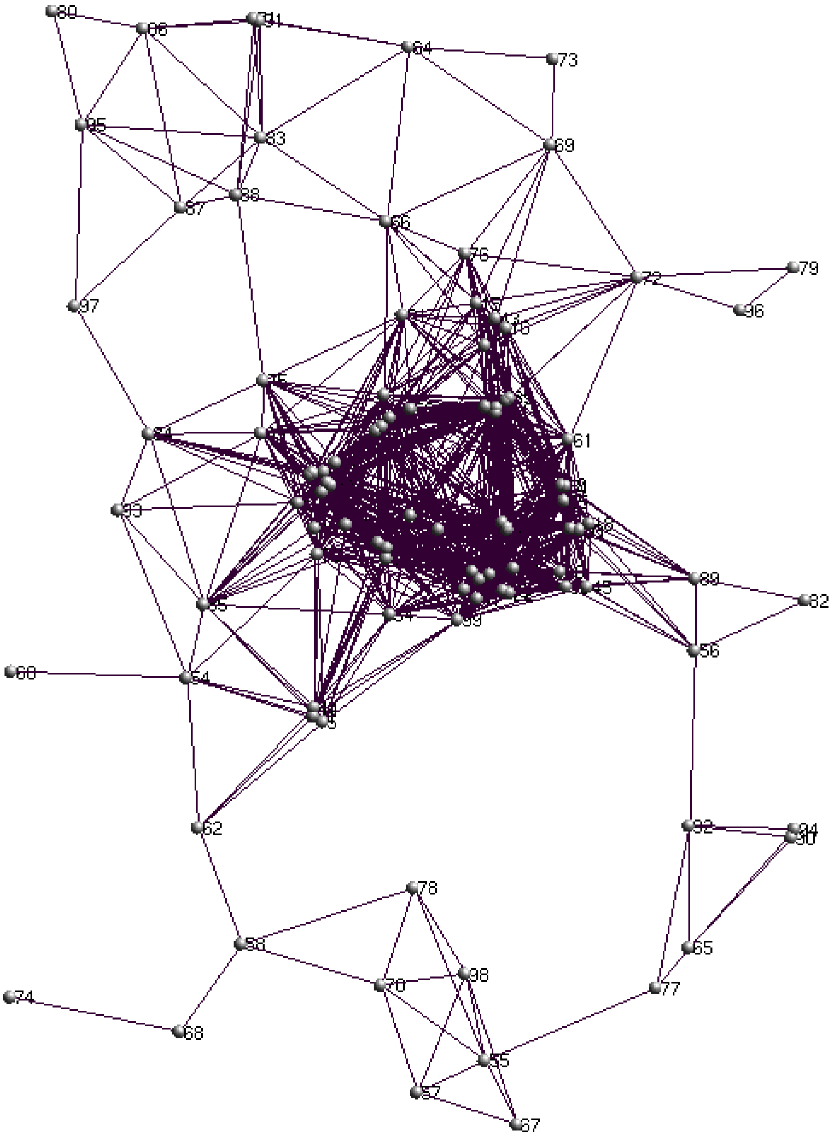


Fig. 2.1. *100 Nodes cluster around 14 Hotspots.*

to color each interval with the minimal number of colors in such a way that no two overlapping intervals are colored with the same color. The minimal number of colors represents the upper bound of the Maximum Node Degree[1] In Figure 4.1, we graph the dynamic membership information, as an interval graph, which is parsed from the output of RealMobGen with 40 wireless nodes.

In this section we present the algorithm for the Maximum Node Degree. The algorithm reads each interval and sorts them by the beginning interval times (See Figure 4.1).

The *MaximumNodeDegree* is derived by the *GreedlyMaximumNodeDegree* Algorithm (See Algorithm 2). The input to the algorithm is the dynamic membership of the nodes, which is a series of intervals that are determined by their beginning and ending time of active nodes. Our goal is to find all the overlapping active time intervals. The number of overlapping intervals gives us the upper bound of the number of neighbors for each node. We set the problem as a coloring problem. The goal is to color each interval with the minimal number of colors in such a way that no two overlapping intervals are colored with the same color.

[1] The algorithm in this section is similar to independent work in the context of channel routing problem [14].

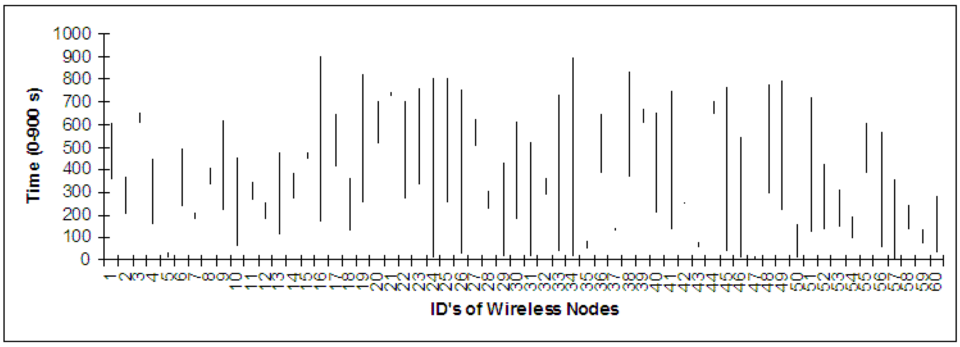


FIG. 2.2. *Dynamic Membership of 60 Nodes.*

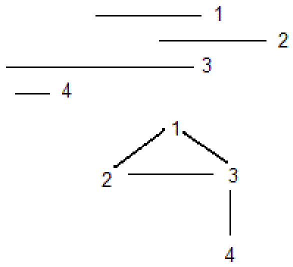


FIG. 3.1. *A set of four intervals and the corresponding interval graph.*

First, the algorithm reads all the intervals and sorts them by their beginning times. After it assigns to the first interval the Color 1, the algorithm looks at the second interval, in order to figure out whether it overlaps with any other intervals. If it overlaps, then it calculates what colors have already been used and assigns the next available color to the interval. On the other hand, if the answer is “no”, then the algorithm assigns the color of the overlapping interval.

```
Algorithm 2 : GreedyMaximumNodeDegree.
Input:   Active Intervals(”TimeOn”, ”TimeOff”); (N) Intervals
sortedIntervals ← sortByTimeOn (Active Intervals).
GreedilyIntervalColoring (sortedIntervals).
loopCounter ← 1
colorCounter ← 1
ColorIntervals(sortedIntervals[loopCounter]) ← Colour[colorCounter].
repeat
  loopCounter ← loopCounter + 1
  nextInterval ← sortedIntervals [loopCounter].
  if nextInterval OVERLAPS previousIntervals then
    colorCounter ← colorCounter + 1
    ColorIntervals(sortedIntervals[loopCounter]) ← (Colour[colorCounter])
  else
    ColorIntervals(sortedIntervals [colorCounterOfOverlapping]) ← (Colour[colorCounterOfOverlapping])
  end if
until $loopCounter \leq N$
Output:   Active Intervals: ("TimeOn","TimeOff"), Interval Color
```

We illustrate the algorithm by running it in the same example of the 40 nodes used in the previous sections. For example, in Table 4.1 we show the partial output of the algorithm. The table illustrates that the first seven intervals overlap, thus the color of the intervals is incremental. On the other hand, the eighth interval

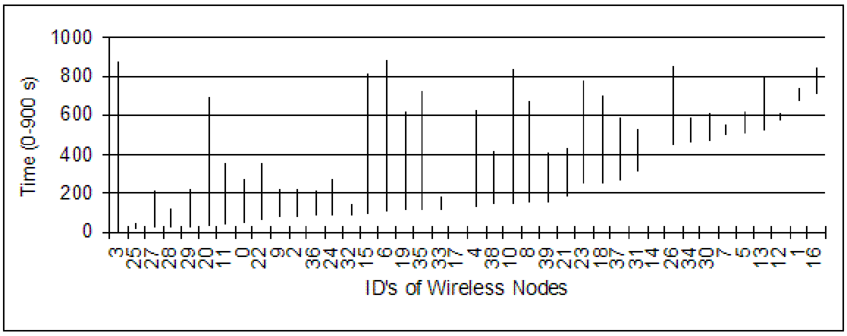


Fig. 4.1. *Dynamic Membership Graph on 40 nodes: Sorted by beginning interval times.*

Table 4.1
The partial output of the Algorithm $GreedyMaximumNodeDegree$.

Time node turns “ON”	Time node turns “OFF”	Assigned Interval Color
1.28239834	874.2507994	1
9.112384965	35.76696803	2
12.33443828	206.6781021	3
13.96709987	116.2303641	4
15.42727166	212.4065595	5
22.61315476	686.9883725	6
28.87034892	349.2144535	7
39.14919133	269.6973103	2

defined by $(39.14919133, 269.6973103)$ does not overlap with the second one $(9.112384965, 35.76696803)$, thus gets assigned the same color (Color 2).

The algorithm $GreedyMaximumNodeDegree$ falls in the greedy class of the algorithms, since it takes the best immediate, or local, solution while finding a global one. The global solution represents the upper bound of the node neighbors degree, while the average of the local optimum(s) and the global one provides us with the average node neighbors degree. It can be shown that the algorithm is optimal (following a well known result from greedy scheduling algorithms [14]).

So far, we have provided the algorithm for the upper bounds of the Maximum Node Degree mobility metric. Now, we will derive the lower bounds for the metric. The input to the algorithm for the lower bounds is the ***Active Intervals***: $("TimeOn","TimeOff")$ and the *Interval Color*. For each color from the Interval Color, we compute the pairwise Euclidian distance of the nodes that belong to that color. If the distance is greater than the transmission range, then the nodes are not neighbors. Else, the nodes are neighbors, thus we increment the $MaximumNodeDegree$ counter for this pair of nodes. The algorithm is summarized below:

```
INPUT Active Intervals: $("TimeOn","TimeOff")$, Interval Color
For each Color from the Interval Color do
        • Compute the pairwise Euclidian distance of the nodes that belong to that color.
        • If $D_{i,j} \leq R$, increment the $MaximumNodeDegree$ counter for this pair of nodes.
        • Else, process to the next pair of nodes.
```

We should mention that in NS 2, in order to maintain the lower bounds over the entire simulation time the snapshot of the network should be taken every $0.5s$ and compute the metric for each snapshot. The average of the metric should be computed in the and over the entire simulation time and used as the lower bound of the average maximum node degree.

5. Link and Path Durations Mobility Metrics. We make use of dynamic programming to compute the mobility metrics of link and path durations. In order to compute these metrics, we make optimal solutions sequentially during the simulation, thus ensuring a global solution in the continuous time. In this section again we use the fact that ad hoc network simulations evolve with time; thus, to compute the upper and lower bounds of link and path durations over the entire simulation time we take a snapshot of the network every $0.5s$ and compute the metrics for each snapshot. Then, we average the results of all the individual snapshots and present the average Maximum Node Degree over the entire simulation. For example, if the simulation time is $900s$, then the first step is to divide the simulation time in small time intervals (for illustration purposes we selected to divide it into $0.5s$ time intervals) and take the snapshot of the network for each time interval, which results into $\frac{900s}{0.5s} = 1800snapshots$.

First, we present the algorithm that provides the lower bounds for the path duration metric. So far, the research on this area considers the graph static on the number of nodes. Therefore, the run time of the shortest path(s) on N nodes for each snapshot is $O(V^3)$ (over the simulation for the $900s$ simulation time it will be $1800 * O(V^3)$ (where $|V| = N$).

In this section we improve the algorithm complexity to $O((V^*)^2 logV^* + V^*E^*)$ for each snapshot (V^* represents the active nodes only and E^* represents the active edges for each snapshot, thus both are dynamic). The algorithm takes as input the dynamic membership of the nodes, which is represented by Active Intervals("TimeOn", "TimeOff"). On each snapshot we include only the nodes that are active at that time, thus allowing us to perform the algorithm only for those nodes adjacent to the previously selected nodes [8]. In this paper we implemented Johnson's sequential single-source shortest paths algorithm (As shown in Algorithm 3).

Algorithm 3 : $JohnsonSingleSourceShortestPaths$.

```
Input: Active Vertices;Active Edges;s
PriorityQueue=ActiveVertices
for all $v \epsilon ActiveVertices$ do
    $l[v] \Leftarrow \infty$
end for
$l[s] \Leftarrow 0$
while $PriorityQueue \neq 0$ do
    $u \Leftarrow findminimum(PriorityQueue)$
    for each $v \epsilon Adjacent[u]$ do
        if $(v \epsilon PriorityQueue)$ and $(l[u] + w(u,v) < l[v]))$ then
            $l[v] \Leftarrow l[u] + w(u,v)$
        end if
    end for
end while
Output: ShortestPathsMatrix
```

Let's illustrate the output of the algorithm through an example. Let's assume that we are given the undirected graph and its weights (See Figure 5.1).

We run the algorithm $JohnsonSingleSourceShortestPaths$, which outputs the matrix

$$ShortestPathsMatrix = \begin{pmatrix} 0 & 53 & 53 & 45 & 21 & 82 \\ 53 & 0 & 51 & 66 & 32 & 40 \\ 53 & 51 & 0 & 15 & 32 & 29 \\ 45 & 66 & 15 & 0 & 36 & 38 \\ 21 & 32 & 32 & 36 & 0 & 61 \\ 82 & 40 & 29 & 38 & 61 & 0 \end{pmatrix}.$$

In the implementation phase we designed matrix multiplication as addition, while the addition as a minimization operators. In addition, the product of adjacency matric with itself returns shortest pair of length 2 for any pair of nodes. Thus, to get the n shortest paths we compute by doubling powers of the $Adjacency$, $Adjacency^2$, $Adjacency^4$,... $Adjacency^n$. Furthermore, we have added the parallel implementation of Johnson's Algorithm, which extracts simultaneously p nodes from the priority queue. Each of the nodes will update the neighbors costs, as well as, augment to the shortest path. The improvement with the parallel algorithm is $\frac{|V^*|}{p} * log\frac{|V^*|}{p}$ on p processors for each snapshot.

The Link Durations is computed by the algorithm $LinkDurations$ (See Algorithm 4).

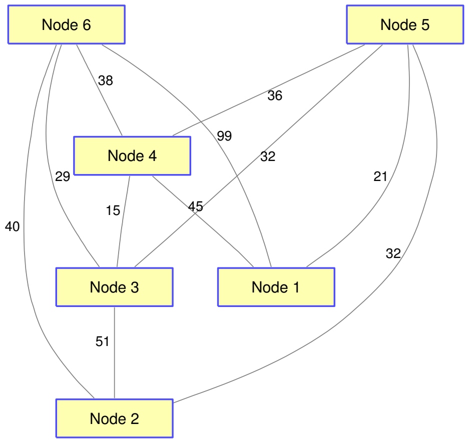


Fig. 5.1. *Illustrate all-pair shortest path algorithm.*

```
Algorithm 4 : $LinkDurations$.
Input:  Active Vertices;Active Edges;
for all $v \epsilon ActiveVertices$ do
    for each $v \epsilon Adjacent[u]$ do
        if $(voverlapsu)$ and $(D(u, v) < R))$ then
            $link[u, v] \Leftarrow OverlapTime[u, v]$
        end if
    end for
end for
Output:  LinkDurationsMatrix
```

So far in this section, we have provided the lower bounds for the Link and Path durations mobility metric. In order to get the upper bounds of the maximum node degree the algorithm takes as input the $GreedyMaximumNodeDegree$ algorithm's output. For each overlapping interval we compute the amount of the time they overlap. The overlapping time represents the link durations between the two wireless nodes. We repeat the computation until there are no more overlapping intervals. For the path durations we compute all the possible paths of the overlapping intervals and add the corresponding link durations. Lastly, we average all the individual snapshot results and present the average Link and Path Durations over the entire simulation.

6. Case Study: Efficiency using Maximum Node Degree. In order to demonstrate how to use the Maximum Node Degree mobility metric to improve algorithm efficiency we illustrate it by a case study that is the incentive protocols in ad hoc networks.

6.1. Incentive Ad Hoc Protocols. Ad hoc networks are self-organizing and multi-hop networks with no central authority. Thus, every aspect of the configuration and operation of an ad-hoc network is distributed. Another characteristic is that nodes are power and energy constrained. Thus, each node running a distributed protocol must make its own decisions (possibly relying on information from other nodes). Those decisions maybe constrained by the rules or algorithms of a protocol, but ultimately each node would have some leeway in setting parameters or changing the mode of operation. These nodes are autonomous agents, making decisions about transmit power, packet forwarding, back off time, and so on.

In the presence of the selfish nodes, the goal of each wireless device is to maximize its welfare: $WELFARE = Profit - Costs$, where profit is the payments received for forwarding traffic and the costs are the incurred energy loss of the node by transmitting packets for other nodes and sending its own traffic; and the payments to others that forwarded its own traffic. On the other hand the goal of the incentive protocol designer is to provide

Table 6.1
RealMobGen Simulation Parameters.

Parameter	Value
Simulation Duration	900 s
Simulation Area	900 × 1200 m
Number of Hotspots	14
Number of Nodes (nn)	$nn \in (40, 60, 90)$

incentives to the wireless nodes to relay traffic, such that nodes will have no incentives to deviate from the protocol, since doing so will not bring them a higher welfare.

In the presence of selfish nodes most of the proposed protocols are based on the well known Vickrey-Clarke-Groves (VCG) mechanism. Below we describe one such protocol based on VCG. Vickery auction is most familiar because it is the foundation of eBay's auction design. In the Vickrey auction the high bidder wins, but pays the second-highest bid. This is why the Vickrey auction is called a second-price auction: the price is not the highest bid, but the second-highest bid. Desirable properties of auction protocols are given in [12], which include Strategy-Proofness, Pareto Efficiency, Individual Rationality, and Budget-Balance.

In [1, 4, 17] the authors propose incentive protocols that ensure that the participating wireless nodes will have no reason to deviate from the protocol, since doing so will not bring them a higher welfare. For example, in [1] the authors propose the Ad-Hoc VCG protocol, which is a reactive routing protocol that achieves the design objectives of truthfulness. Reactive protocols seek to set up routes on-demand, thus topology information is only transmitted by nodes on-demand. In this protocol vertexes represent the nodes and weighted directed edges represent the payments an emitting node has to receive. Nodes are awarded payments for forwarding a message, thus cover the cost for forwarding a unit-size packet: $Payment = c_i * P^{emit}$, where c_i is the cost-of energy of $\$/Watt$ and P^{emit} is emission signal strength in $watt$.

$$Payment = DeclaredCost + (LCP_{withthenode} - LCP_{withoutthenode}). \tag{6.1}$$

The protocol can be thought as it is run on two phases. Firstly, route discovery, where nodes communicate to destination P^{emit} and c_i. Then, the destination computes Lowest Cost Path (LCP). Secondly, data transmission, packets are forwarded along the shortest path route and payments are made to the intermediate nodes.

The algorithm complexity for the proposed protocols [1, 17] is $O(N^3)$, for each time snap of the network during simulation time. In [4] the authors try to reduce the complexity to $O(M^2 * d)$, where d is the diameter and M is some upper bound for the node degree. However, the value of M was not computed. We computed M by using the $GreedyMaximumNodeDegree$ Algorithm. Next, we present the efficiency introduced by taking into account the value of M, which is the maximum node degree.

6.2. Efficiency. We propose the metric *Efficiency*, as a metric to evaluate the gains in the algorithm complexity of incentive protocols.

In this section, we run the *GreedlyDynamicMembership* with several scenarios. The scenarios are generated using RealMobGen mobility model. The inputs to the RealMobGen are the simulation duration set to 900 seconds; simulation area of 900 × 1200 meters; number of hotspots set to 14, while the number of nodes was varied 40, 60, 100 (the parameters are summarized in Table 6.1).

For each set of nodes, we run the simulation 10 times and present the results in Table 6.2. We define a new metric, namely *Efficiency*, that calculates the improvement of real mobility metric algorithms over the synthetic ones, i. e., in terms of relevant comparisons (See Eq.(6.2)).

$$\mathit{Efficiency} = \left(\frac{\frac{\sum_T MaxNodeD_{egree}}{T}}{N} \right) * 100. \tag{6.2}$$

For example, from the table we have the $Efficiency$ for $40, 60, 100$ number of nodes to be respectively $54.75, 54.33, 52.30$, which demonstrates that we need to compare only half of the nodes, instead of the full graph when computing the mobility metrics (N^2).

Table 6.2
Number of necessary comparisons on 40, 60, 100 nodes.

$MaxNodeD_{egree}$ on 40 Nodes	$MaxNodeD_{egree}$ on 60 Nodes	$MaxNodeD_{egree}$ on 100 Nodes
24	33	50
21	27	55
16	28	49
25	35	55
21	32	53
25	34	55
20	37	48
23	32	51
21	29	54
23	39	53
Efficiency = 54.75%	*Efficiency* = 54.33%	*Efficiency* = 52.30%

7. Conclusions. In this paper we demonstrated that when introducing dynamic topology in wireless *adhoc* networks, the current mobility metrics need to be re evaluated. We introduced Maximum Node Degree as a mobility metrics, which can be used to improve algorithm efficiency in wireless *adhoc* networks. In addition, we re-evaluate the link and path durations mobility metrics and provided their lower and upper bounds. The usefulness of the Maximum Node Degree mobility metric for improving the efficiency is illustrated by a case study of incentive protocols in ad hoc networks. We have defined a new metric, namely *Efficiency*, to characterize the improvement of real mobility metric algorithms over the synthetic ones. The results are validated trough simulations.

REFERENCES

[1] L. Anderegg and S. Eidenbenz, *Ad hoc-vcg: a truthful and cost-efficient routing protocol for mobile ad hoc networks with selfish agents*, in MobiCom, Septmber 2003.

[2] J. Boleng, W. Navidi, and T. Camp, *Metrics to enable adaptive protocols for mobile ad hoc networks*, in International Conference onWireless Networks, 2002, pp. 293-298.

[3] J. Broch, D. Maltz, D. Johnson, Y. Hu, and J. Jetcheva, *Multihop wireless ad hoc network routing protocols*, in MobiCom 1998: Proceedings of the Fourth Annual ACM International Conference on Mobile Computing and Networking, 1998, p. 8597.

[4] S. Eidenbenz, G. Resta, and P. Santi, *Commit: A sender-centric truthful and energy-efficient routing protocol for ad hoc networks with selfish nodes*, in IPDPS, April 2005.

[5] F.Bai, N. Sadagopan, and A. Helmy, *Important: a framework to systematically analyze the impact of mobility on performance of routing protocols for adhoc networks*, in INFOCOM, 2003, pp. 825–835.

[6] W. J. Hsu, K. Merchant, H. W. Shu, C. H. Hsu, and A. Helmy, *Weighted waypoint mobility model and its impact on ad hoc networks*, SIGMOBILE Mobile Computer Communications Review, 9 (2005), pp. 59–63.

[7] S. Jiang, *An enhanced prediction-based link availability estimation for manets*, IEEE Transactions on Communications, 52 (2004), pp. 183–186.

[8] D. B. Johnson, *Efficient algorithms for shortest paths in sparse networks*, J. ACM, 24 (1977), pp. 1–13.

[9] H. M. Jones, S. Xu, and K. Blackmore, *Link ratio for ad hoc networks in a rayleigh fading channel*, in WITSP, 2004, pp. 252–255.

[10] M. Kim, D. Kotz, and S. Kim, *Extracting a mobility model from real user traces*, in INFORCOM: Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies, 2006, pp. 1–13.

[11] S. Kurkowski, W. Navidi, and T. Camp, *Constructing manet simulation scenarios that meet standards*, in MASS, 2007, p. To Appear.

[12] A. Mas-Colell, M. D. Whinston, and J. R. Green, *Microeconomic theory*, Oxford University Press, (1995).

[13] N. Sadagopan, F. Bai, B. Krishnamachari, and A. Helmy, *Paths: analysis of path duration statistics and their impact on reactive manet routing protocols*, in MobiHoc, 2003, pp. 245–256.

[14] M. Sarrafzadeh and C. K. Wong, *An Introduction to VLSI Physical Design*, McGraw-Hill Higher Education, 1996.

[15] W. Su, S. Lee, and M. Gerla, *Mobility prediction and routing in ad hoc wireless networks*, International Journal of Network Management, 11 (2001), pp. 3–30.

[16] C. Walsh, A. Doci, and T. Camp, *A call to arms: its time for real mobility models*, vol. 12, no. 1, pp. 3436, 2008.

[17] S. Zhong, L. E. Li, Y. G. Liu, and Y. R. Yang, *On designing incentive-compatible routing and forwarding protocols in wireless ad-hoc networks: an integrated approach using game theoretical and cryptographic techniques*, in MobiCom, 2005, pp. 117–131.

Edited by: Fatos Xhafa, Leonard Barolli
Received: September 30, 2008
Accepted: December 15, 2008

Scalable Computing: Practice and Experience
Volume 10, Number 1, pp. 35–47. http://www.scpe.org

ISSN 1895-1767
© 2009 SCPE

AUTOMATIC PERFORMANCE MODEL TRANSFORMATION FROM A HUMAN-INTUITIVE TO A MACHINE-EFFICIENT FORM

SABRI PLLANA, SIEGFRIED BENKNER[*] FATOS XHAFA[†] AND LEONARD BAROLLI[‡]

Abstract. We address the issue of the development of performance models for programs that may be executed on large-scale computing systems. The commonly used approaches apply non-standard notations for model specification and often require that the software engineer has a thorough understanding of the underlying performance modeling technique. We propose to bridge the gap between the performance modeling and software engineering by incorporating UML. In our approach we aim to permit the graphical specification of performance model in a human-intuitive fashion on one hand, but on the other hand we aim for a machine-efficient model evaluation. The user specifies graphically the performance model using UML. Thereafter, the transformation of the performance model from the human-usable UML representation to the machine-efficient C++ representation is done automatically. We describe our methodology and illustrate it with the automatic transformation of a sample performance model. Furthermore, we demonstrate the usefulness of our approach by modeling and simulating a real-world material science program.

Key words: performance modeling, model transformation, UML, simulation.

1. Introduction.

It is impractical and costly to use a large-scale computing system for performance tuning during the program development. Furthermore, in the case of large-scale computing systems the program developer commonly has access to only a part of the computing system resources and for only a limited time. The model-based performance analyzes may be used to overcome these obstacles [17]. Based on the model, the performance can be predicted and design decisions can be influenced without time-consuming modifications of large portions of an implemented program. In the past the performance evaluation of computing systems was a preoccupation of many computer scientists [8, 17]. However, most of approaches [1, 9, 6, 13, 21, 10, 7] for the performance modeling of parallel and distributed programs are of limited use to support performance-oriented software engineering because of the following reasons: (1) the use of a notation that is not based on widely accepted standards, and (2) the requirement that the software engineer has a thorough understanding of the underlying performance modeling technique.

In our approach we aim to bridge the gap between the performance modeling and the software engineering by using the Unified Modeling Language (UML) [2]. We have developed an extension of UML for the domain of performance-oriented parallel and distributed programs [18, 19]. Our UML extension provides a set of UML building blocks that model some of the most important concepts of message passing and shared memory programming paradigms, which can be used to develop models for large and complex parallel and distributed programs. To provide tool support for our approach we have developed the Performance Prophet [16], which is a performance modeling and prediction system. Performance Prophet provides a UML based graphical user interface, which alleviates the problem of specification and modification of the performance model. In the context of Performance Prophet we aim to permit the graphical specification of performance model in a human-intuitive fashion on one hand, but on the other hand we aim for a machine-efficient model evaluation. The user specifies graphically the performance model using UML. Afterwards, Performance Prophet automatically transforms the performance model from UML to C++ and evaluates it by simulation.

In this paper we describe our methodology for automatic transformation of performance models from UML to C++. We show how we may develop a UML-based performance model for a given program code, and thereafter we explain how the UML representation of the performance model is transformed to the corresponding C++ representation. Furthermore, we present and explain our algorithm for the performance model transformation from UML to C++, which is implemented in the Performance Prophet. We illustrate our methodology with the transformation of a sample performance model using the Performance Prophet. The usefulness of our approach is demonstrated by modeling and simulating LAPW0, which is a real-world material science program that comprises about 15,000 lines of code.

[*] University of Vienna, Department of Scientific Computing, Nordbergstrasse 15/C/3, 1090 Vienna, Austria, ({pllana, sigi}@par.univie.ac.at).

[†] Polytechnic University of Catalonia, Department of Languages and Informatics Systems, C/Jordi Girona 1-3, 08034 Barcelona, Spain, (fatos@lsi.upc.edu).

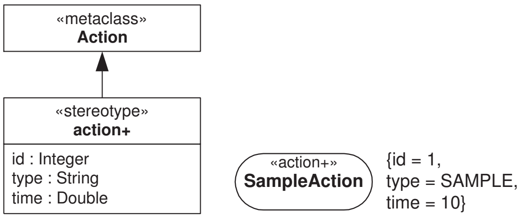
[‡] Fukuoka Institute of Technology, Department of Information and Communication Engineering, 3-30-1 Wajiro-Higashi, Higashi-ku, Fukuoka 811-0295, Japan, (barolli@fit.ac.jp).

The rest of this paper is organized as follows. Section 2 describes how we customized the UML for performance modeling and outlines the architecture of the Performance Prophet. Our performance model transformation methodology is presented in Section 3. Section 4 exemplifies the model transformation using the Performance Prophet. The usefulness of our approach is demonstrated in Section 5 by modeling and simulating a real-world material science program. Finally, Section 6 concludes the paper and briefly describes the future work.

2. Preliminaries. In this section we describe our approach for customization of the UML for performance modeling of parallel and distributed programs and give an overview of the architecture of Performance Prophet.

2.1. UML-Based Performance Modeling. UML [2, 14] is a graphical language that is primarily used for visualizing, specifying, and documenting the software-intensive systems. In order to make possible the modeling of different types of systems, UML modeling elements are defined in UML specification in an abstract manner without conceptual connection with a particular domain. For instance, the UML specification defines the modeling element Action as follows: "*an action is the fundamental unit of behavior specification*" [14]. Such an abstract definition allows us to use an action to model various kinds of behavior such as *addition* of two numbers in a computer system, or *acceleration* of a vehicular system. However, too generic semantics of UML modeling elements may present an obstacle for using UML in a specific domain. For this reason, UML specification defines the mechanisms for specializing semantics of modeling elements for a particular domain. UML extension mechanisms include *stereotypes*, *tagged values*, and *constraints*.

The UML may be extended by defining new modeling elements, *stereotypes*, based on existing elements, *base classes* (i. e. metaclasses). A stereotype is defined as a subclass of an existing UML metaclass, with the associated *tagged values* (i. e. metaattributes) and *constraints*. Stereotypes are notated by the stereotype name enclosed in guillemets `<<StereotypeName>>`, or by a specific graphic icon. Stereotypes may improve the readability of models by distinguishing modeling elements of the same shape with different stereotype names.



(a) Definition (b) Usage

FIG. 2.1. *Definition and usage of the stereotype* `<<action+>>`.

Figure 2.1(a) depicts the definition of stereotype `<<action+>>` based on the UML metaclass *Action*. The list of tag definitions includes *id*, *type*, and *time*. Tag *id* can be used to uniquely identify the modeling element `<<action+>>`; tag *type* specifies the type of `<<action+>>`, and tag *time* the time spent to complete `<<action+>>`. We are using `<<action+>>` (see example in Figure 2.1(b)) to model various types of single-entry single-exit code regions. Commonly we use tags to describe performance relevant information, such as the estimated or the measured execution time (see the tag *time* in Figure 2.1(b)). The set of tag definitions is not limited to those shown in Figure 2.1(a), but it can be arbitrarily extended to meet the modeling objective. In this manner we have extended the UML for performance modeling of parallel and distributed programs [18, 19].

2.2. Performance Prophet. Performance Prophet [16] is a performance modeling and prediction system for parallel and distributed computing systems. The architecture of Performance Prophet is depicted in Figure 2.2. The main components of Performance Prophet are *Teuta* and *Performance Estimator*. Teuta is a platform independent tool for graphical modeling of parallel and distributed programs. The role of Performance Estimator, in the context of Performance Prophet, is to estimate the performance of a program on a computing machine.

Teuta comprises the following parts: Model Checker, Model Traverser, Graphical User Interface (GUI), and the components for Performance Visualization (see Figure 2.2). The GUI of Teuta is used for the development of performance model based on the UML [14]. The Model Checker is used to verify whether the model conforms

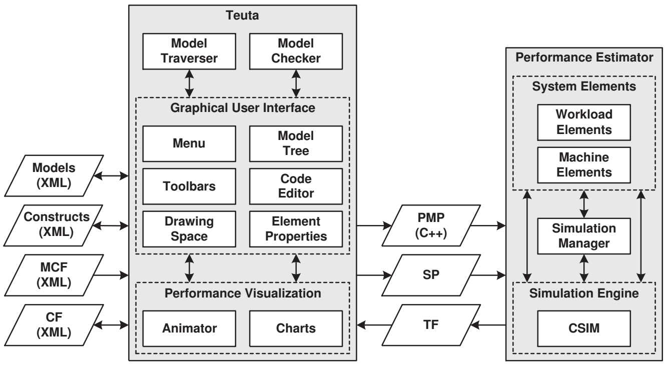


Fig. 2.2. *The architecture of Performance Prophet. Abbreviations: Model Checking File (MCF), Configuration File (CF), Performance Model of Program (PMP), System Parameters (SP), Trace File (TF).*

to the UML specification. The Model Traverser is used for generation of different model representations (XML and C++). The Performance Visualization components are used for visualization of the performance results.

Element MCF indicates the XML file, which is used for the model checking. The XML files that are used for the configuration of Teuta are indicated with the element CF.

The Performance Estimator estimates the performance of a parallel and distributed program on a target computer architecture. As input for the Performance Estimator serve the program model and architectural parameters that are specified in Teuta. The Performance Estimator generates automatically the machine model based on the specified architectural parameters. The program model is integrated with the machine model to create the model of the whole computer system. The Performance Estimator evaluates the integrated model of computing system and generates the corresponding performance results.

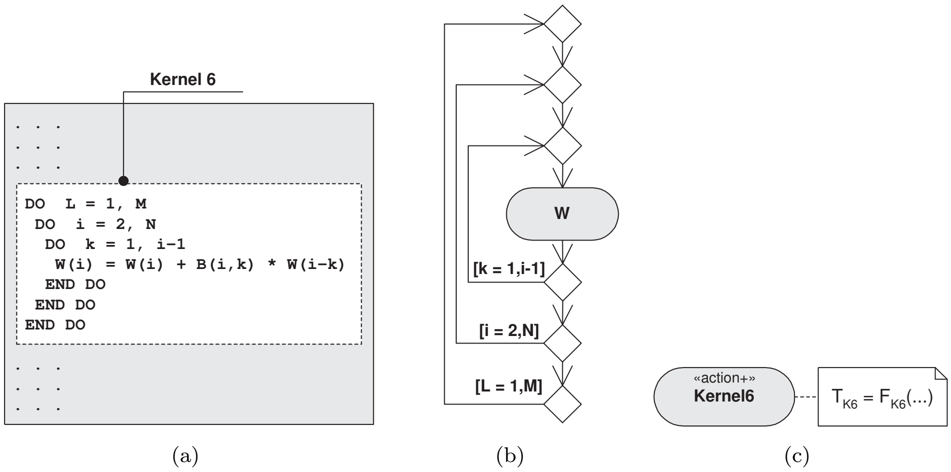
The communication between Teuta and the Performance Estimator is done via elements PMP, SP and TF. Element PMP indicates the C++ representation of the program's performance model. PMP is generated by Teuta and serves as input information for the Performance Estimator. Element SP indicates a set of system parameters. The parameters of system include the number of computational nodes, the number of processors per node, the number of processes, and the number of threads. The Performance Estimator uses SP for building the model of system, whose performance is estimated. Element TF represents the trace file, which is generated by the Performance Estimator as a result of the performance evaluation. Teuta uses TF for the visualization of performance results.

3. Methodology. In this section we describe conceptually the transition: (1) from the program code to the UML based performance model, and (2) from the UML representation to the C++ representation of performance model. Thereafter, we present our algorithm that takes as input the UML representation, and automatically generates the C++ representation of the performance model.

Commonly, scientific programs are written in imperative languages such as *Fortran* or *C*. This type of programs is executed on parallel and distributed computing systems, which may consist of multiple nodes (each node may have multiple processors), in order to solve large problems or to reduce the time to solution for a single problem [4]. The MPI [22, 12] is usually used to express the inter-node parallelism, whereas OpenMP [3, 15] is used to express the intra-node parallelism. We have identified that UML activity diagrams are suitable for modeling scientific imperative programs [19]. Therefore, we usually model a scientific program with one or more activity diagrams. Activity diagrams may be annotated with performance-relevant information. For instance, cost functions that model the execution time of program actions may be associated with *ActionNodes* of the activity diagram.

During the process of performance modeling are considered only the *code blocks* that strongly influence the overall performance of program. We may identify, for an existing program, code blocks that determine the overall program performance by using a profiling tool.

Figure 3.1(a) shows a code block of a *Fortran* program. This code block is known as *kernel 6* of the Livermore Fortran kernels [11]. Since the performance of a scientific program is strongly influenced by loops, it is important to consider loops during the development of performance model. Figure 3.1(b) shows the UML model of kernel 6, which is a fragment of an activity diagram. But, this detailed UML representation of the

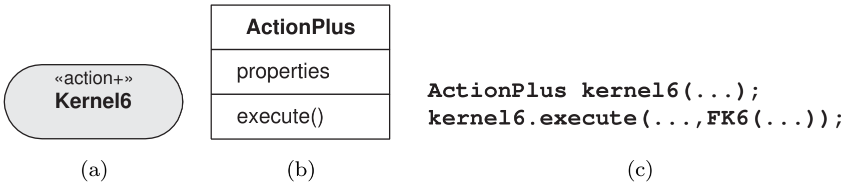


```
DO  L = 1, M
 DO  i = 2, N
  DO  k = 1, i-1
   W(i) = W(i) + B(i,k) * W(i-k)
  END DO
 END DO
END DO
```

FIG. 3.1. *From the program code to the UML based performance model.*

kernel 6 is not necessary, since we are interested on the rough performance estimation. Therefore, we model the performance of the kernel 6, that is depicted in Figure 3.1(a), with the action *Kernel6*, which is an instance of the stereotype `action+` (see Figure 3.1(c)). The associated cost function $F_{K6}(\dots)$ models the execution time T_{K6} of kernel 6.

The UML based representation of performance model of the kernel 6, that is depicted in Figure 3.1(c), is simple and intuitive. We have developed this graphical representation to streamline the specification process of performance models. However, while the UML representation is suitable as human-usable notation for performance model specification, it is not adequate for an efficient model evaluation. Therefore, we need to transform the UML representation to a form that is suitable for evaluation.



```
ActionPlus kernel6(...);
kernel6.execute(...,FK6(...));
```

FIG. 3.2. *From the UML representation to the C++ representation of performance model.*

Figure 3.2 depicts an example of transition from the UML representation to the C++ representation of performance model. For the illustration of this transformation process serves the model of kernel 6 that we introduced in Figure 3.1. We use the stereotype `action+` to represent a code block of a program. In Figure 3.2(a) the action *Kernel6*, which is an instance of stereotype `action+`, represents the kernel 6. For the modeling element `action+` we have defined the corresponding class *ActionPlus* (see Figure 3.2(b)). In the context of Performance Prophet, the class *ActionPlus* is implemented as a C++ class. The properties of modeling element `action+` are mapped to *properties* of the class *ActionPlus*. The performance behavior of the modeling element `action+` is defined in the method *execute()* of the class *ActionPlus*. Figure 3.2(c) depicts the textual representation of the model of kernel 6. We may observe that the name of the instance of modeling element (in our example *Kernel6*) is mapped to the name of the instance of class *ActionPlus* (in our example `kernel6`).

Figure 3.3 depicts our algorithm for the automatic model transformation from UML to C++ representation. As input serves the UML model of a program. The algorithm generates C++ representation of the model. The UML model, with its diagrams and modeling elements, forms a tree data structure. During the model transformation process the tree is programmatically traversed, which makes possible to visit each modeling element and read its properties (see Figure 3.4). Lines 1–8 of the algorithm determine the performance relevant modeling elements of the UML model based on the element's property *stereotype name*. For instance, modeling elements with the stereotype name `action+` are used to model the performance of sequential code blocks. In

```
Input: uml_mod_rep, UML based model representation

Output: c++_mod_rep, C++ based model representation

Method: A tree structure, which contains the model
with its diagrams and modeling elements, is
traversed during the model transformation process.
Performance relevant modeling elements of the UML
model are identified based on the stereotype name
for instance, <<action+>>).

 1: // Identify and select performance modeling
       elements
 2: FORALL(is diagram of uml_mod_rep) DO
 3:  FORALL(is element of diagram) DO
 4:    IF(element is performance modeling element)
 5:      add element to perf_elements;
 6:    ENDIF
 7:  ENDFOR
 8: ENDFOR
 9: // Globals
10: FORALL(variable of uml_mod_rep is global) DO
11:  add variable to c++_mod_rep;
12: ENDFOR
13: // Cost functions
14: FORALL(is element of perf_elements) DO
15:  IF(element has function)
16:    add function to c++_mod_rep;
17:  ENDIF
18: ENDFOR
19: // Program
20: // Locals
21: FORALL(variable of uml_mod_rep is local) DO
22:  add variable to c++_mod_rep;
23: ENDFOR
24: // Declare performance modeling elements
25: FORALL(is element of perf_elements) DO
26:  identify the type of element;
27:  add element declaration to c++_mod_rep;
28: ENDFOR
29: // Define performance modeling elements and
       their control flow
30: FORALL(is diagram of uml_mod_rep) DO
31:  FORALL(is element of diagram) DO
32:    identify the type of element;
33:    add the corresponding c++ representation to
       c++_mod_rep;
34:  ENDFOR
35: ENDFOR
```

Fig. 3.3. *The algorithm for model transformation from UML to C++.*

the C++ model representation are included the *global variables*, *cost functions*, and the *model structure* (that is performance modeling elements and their flow). Lines 9–12 of the algorithm are responsible for generation of C++ representation of the global variables. Lines 13–18 generate C++ representation of the cost functions (for instance, `double FA1(){ ...};`). The model structure is defined in the lines 19–35. If there are local variables defined in the UML model, then their C++ representation is generated in the lines 20–23. Lines 24–28 declare the performance modeling elements (for instance, `ActionPlus A1(...);`). In the lines 25–35 it is defined the execution flow of modeling elements. The algorithm generates the C++ code that for each performance modeling element invokes its *execute()* method (for instance, `A1.execute(uid, pid, tid, FA1());`). The execution order of performance modeling elements is in accordance with the specified flow in the UML model.

Figure 3.4 shows the UML communication diagram of the model traversing procedure, which provides the possibility to walk programmatically through the model, to visit each modeling element, and to access its properties. We use the model traversing for the generation of various model representations. Model traversing involves three entities: the *Traverser*, the *Navigator* and the *ContentHandler*. During the model traversing procedure, first, the Traverser sends the navigation command to the Navigator. Then, the Traverser obtains the current element *ce* from the Navigator. Finally, the Traverser asks the ContentHandler to visit the element *ce* and generate the corresponding code.

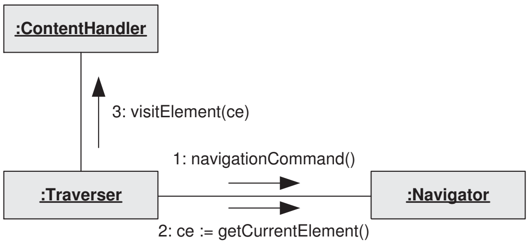


FIG. 3.4. *The UML communication diagram of the Performance Prophet model traverser.*

The Navigator, the Traverser, and the ContentHandler are independent of each other in the sense that they only communicate via well-defined interfaces (see Figure 3.4). Therefore, each implementation of one of these components can be combined with any implementation of the other two components. Performance Prophet provides the necessary interfaces and base classes and default implementations of the Navigator, Traverser and ContentHandler. Commonly, the extension of Performance Prophet for the generation of a specific model representation involves only a specific implementation of the ContentHandler interface.

In the following section we illustrate our methodology with an example using the Performance Prophet.

4. Example. Figure 4.1 depicts the process of UML based specification of a *sample performance model* for a hypothetical program. The user specifies the type of performance modeling elements and their flow. Furthermore, the user may associate a code fragment and a cost function to each performance modeling element. Each performance modeling element corresponds to a code block of a hypothetical program, whose performance is modeled. The example in Figure 4.1(a) illustrates the hierarchical modeling capabilities of Performance Prophet. On the left hand side of Figure 4.1(a) is depicted the main activity diagram, which comprises a set of instances of stereotypes `action+` and `activity+`. After the execution of action *A1* is completed, based on the value of variable *GV*, it is decided whether to execute the activity *SA* or the action *A2*. While an *action* is not further decomposed into other elements, an *activity* contains a set of elements. The content of an activity is described with an activity diagram. The content of activity *SA*, which is an instance of stereotype `activity+`, is depicted in the undocked diagram *SA* in Figure 4.1(a). Activity *SA* comprises performance modeling elements *SA1* and *SA2*.

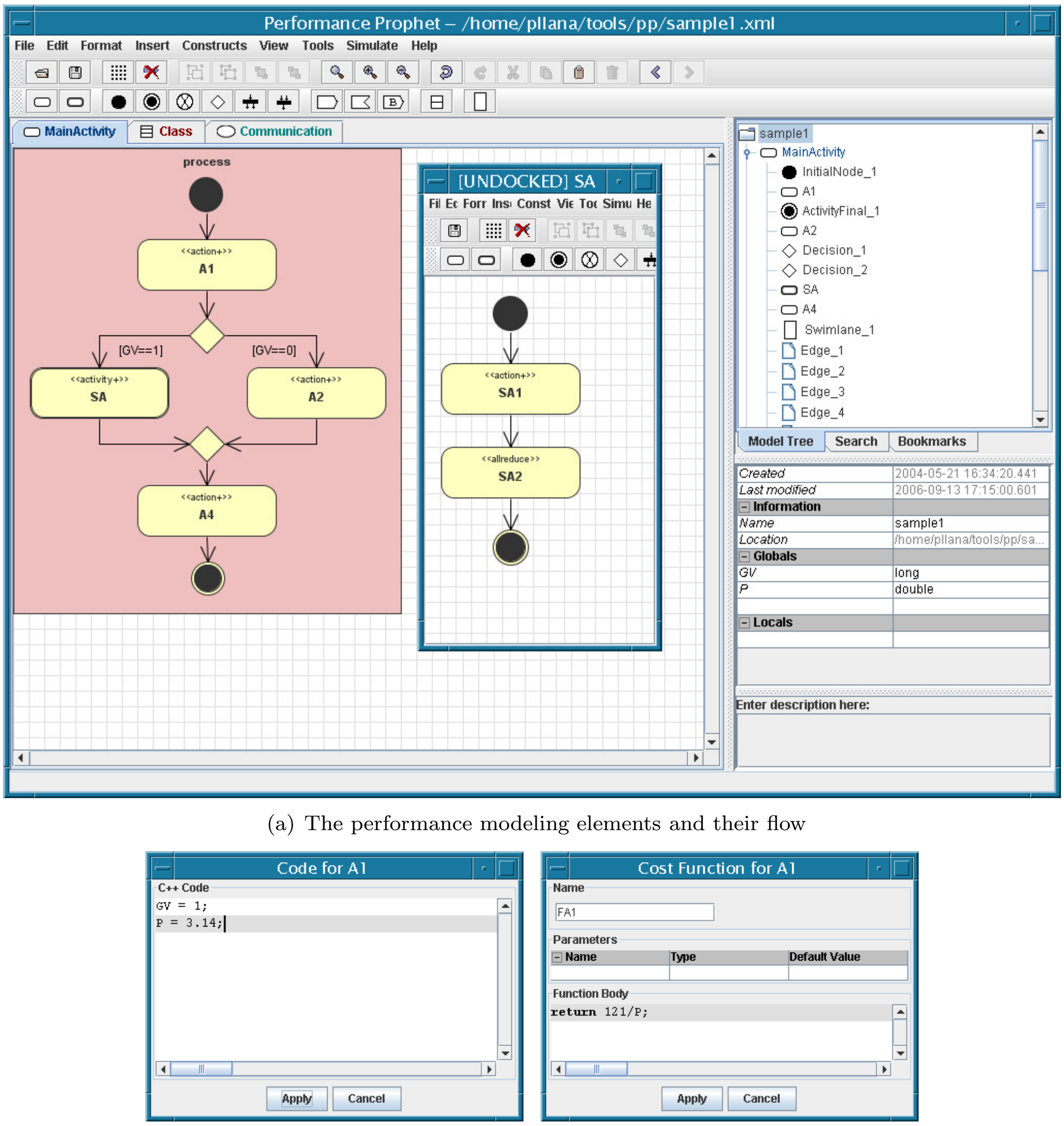
It is possible to associate global and local variables to the model. The name and the type of global and local variables may be specified as properties of the model. On the right-down corner of Figure 4.1(a), in the list of properties of sample model, we may observe that variables GV and P are specified as global variables of the model. We are aware that this sample model could be expressed without the use of global variables, but nevertheless we opted for this solution for illustration purposes.

Figure 4.1(b) depicts an example of the code association to a performance modeling element. This feature can be used to complement C++ representation of the performance model. In this example we have associated a code fragment to the performance modeling element *A1*, which assigns values to global variables GV and P.

Figure 4.1(c) depicts an example of the association of a cost function to a performance modeling element. A cost function models the execution time of the code block that is represented by the performance modeling element. A cost function may use local or global variables as parameters. Moreover, a cost function may be composed using other functions that are defined in the performance model. In this example we have associated a simple parameterized cost function to the performance modeling element *A1*.

Figure 4.2 depicts the C++ representation of the *sample model*, which is automatically generated by the Performance Prophet based on the UML representation. The model transformation from UML to C++ representation is based on the algorithm that we presented in Figure 3.3.

Figure 4.2(a) depicts an excerpt of the C++ model representation that includes two code sections: (1) global variables, and (2) cost functions. Lines 24–25 declare the global variables GV and P. In our sample model the variable GV is used to make the decision whether to execute activity *SA* or action *A2* (see Figure 4.1(a)). The variable P is used as a parameter of cost functions. We may observe that, in this example, for each performance



Created	2004-05-21 16:34:20.441
Last modified	2006-09-13 17:15:00.601
⊟ **Information**	
Name	sample1
Location	/home/pllana/tools/pp/sa...
⊟ **Globals**	
GV	long
P	double
⊟ **Locals**	

(a) The performance modeling elements and their flow

```
GV = 1;
P = 3.14;
```

Name: FA1

Parameters

Name	Type	Default Value

Function Body

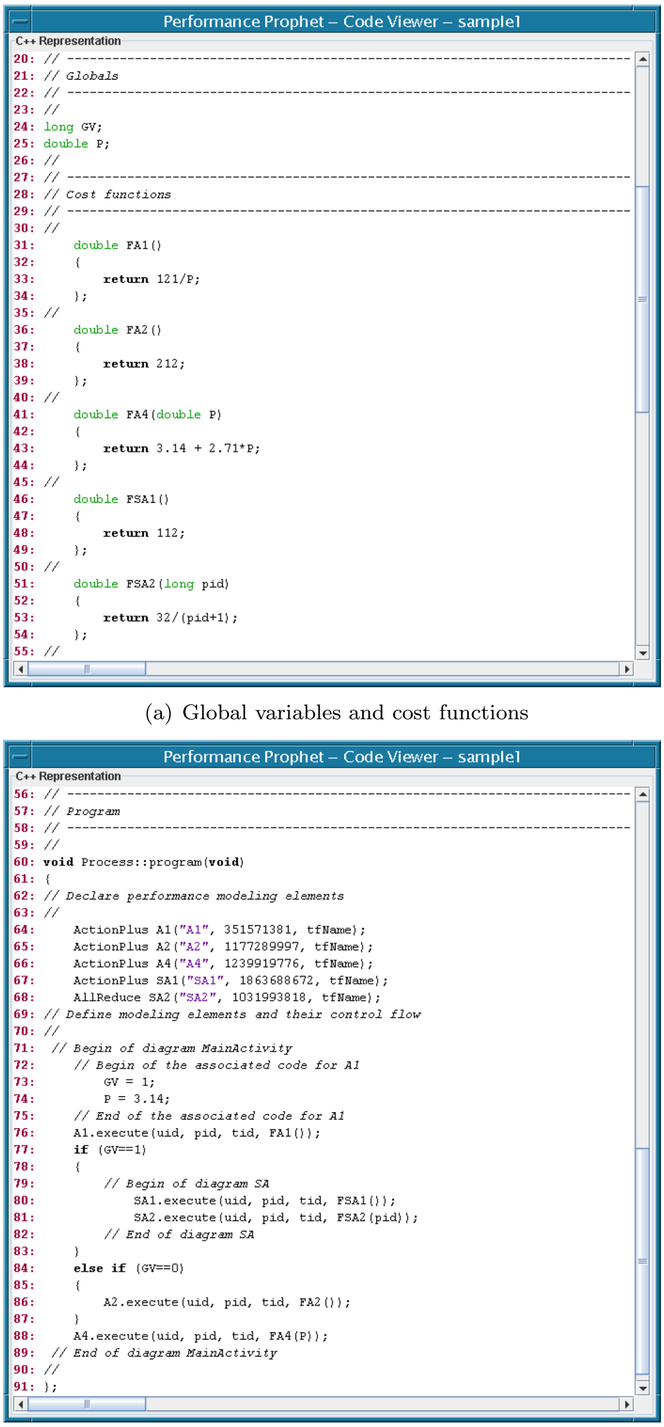
```
return 121/P;
```

(b) Code fragment association (c) Cost function association

FIG. 4.1. *The UML based specification of a sample model.*

modeling element {*A1, A2, A4, SA1, SA2*} it is defined a cost function {*FA1, FA2, FA4, FSA1, FSA2*} (code lines 31–54). As parameters of cost functions may be used the properties of system components (such as number of processors, or the ID of process). For instance, the cost function *FSA2* takes *pid* as a parameter, which is the process ID. Please note that the cost functions presented here serve the purpose of illustration of various forms of expressing cost functions, and that these cost functions are not derived from a real-world program.

Figure 4.2(b) depicts the declaration of performance modeling elements and their execution flow. Lines 64–68 declare the performance modeling elements {*A1, A2, A4, SA1, SA2*}. We may observe that the C++ code that represents activity *SA* (lines 79–82) is nested within the C++ code of the *main activity* (lines 71–89). Lines 72–75 represent the code that is associated with the element *A1* (the code association is depicted in Figure 4.1(b)). A performance modeling element is *executed* by invoking its *execute()* method. The execution of a performance modeling element models the performance behavior of a code block during the program execution. Each performance modeling element corresponds to a code block of the program, whose performance is modeled. For instance, the line 76 in Figure 4.2(b) executes the performance modeling element *A1*. We may observe that one of the parameters of method *execute()* is the name of the cost function *FA1* that is associated with the


Performance Prophet – Code Viewer – sample1
C++ Representation

```
20: // ---------------------------------------------------------------------------
21: // Globals
22: // ---------------------------------------------------------------------------
23: //
24: long GV;
25: double P;
26: //
27: // ---------------------------------------------------------------------------
28: // Cost functions
29: // ---------------------------------------------------------------------------
30: //
31:     double FA1()
32:     {
33:         return 121/P;
34:     };
35: //
36:     double FA2()
37:     {
38:         return 212;
39:     };
40: //
41:     double FA4(double P)
42:     {
43:         return 3.14 + 2.71*P;
44:     };
45: //
46:     double FSA1()
47:     {
48:         return 112;
49:     };
50: //
51:     double FSA2(long pid)
52:     {
53:         return 32/(pid+1);
54:     };
55: //
```

(a) Global variables and cost functions

Performance Prophet – Code Viewer – sample1
C++ Representation

```
56: // ---------------------------------------------------------------------------
57: // Program
58: // ---------------------------------------------------------------------------
59: //
60: void Process::program(void)
61: {
62: // Declare performance modeling elements
63: //
64:     ActionPlus A1("A1", 351571381, tfName);
65:     ActionPlus A2("A2", 1177289997, tfName);
66:     ActionPlus A4("A4", 1239919776, tfName);
67:     ActionPlus SA1("SA1", 1863688672, tfName);
68:     AllReduce SA2("SA2", 1031993818, tfName);
69: // Define modeling elements and their control flow
70: //
71:  // Begin of diagram MainActivity
72:     // Begin of the associated code for A1
73:         GV = 1;
74:         P = 3.14;
75:     // End of the associated code for A1
76:     A1.execute(uid, pid, tid, FA1());
77:     if (GV==1)
78:     {
79:         // Begin of diagram SA
80:             SA1.execute(uid, pid, tid, FSA1());
81:             SA2.execute(uid, pid, tid, FSA2(pid));
82:         // End of diagram SA
83:     }
84:     else if (GV==0)
85:     {
86:         A2.execute(uid, pid, tid, FA2());
87:     }
88:     A4.execute(uid, pid, tid, FA4(P));
89:  // End of diagram MainActivity
90: //
91: };
```

(b) Execution flow of performance modeling elements

FIG. 4.2. *The C++ representation of the sample model.*

element *A1*. The branch control flow of the UML model representation (see Figure 4.1(a)) is mapped to the `if-else-if` statement in C++ model representation (lines 77–87).

The C++ representation that is presented in Figure 4.2 is used as input for the Performance Estimator.

5. Case Study. In this section we demonstrate the usefulness of our approach by modeling and simulating a real-world material science program. For our case study we use LAPW0, which is a part of WIEN2k package [20]. WIEN2k is a program package for calculation of the electronic structure of solids based on the density-functional theory. It is worth to mention that the 1998 Nobel Prize in Chemistry was awarded to Walter Kohn for his development of the density-functional theory. LAPW0 calculates the effective potential within a unit cell of a crystal. The code of LAPW0 program is written in Fortran 90 and MPI [5]. LAPW0 comprises about 15,000 lines of code.

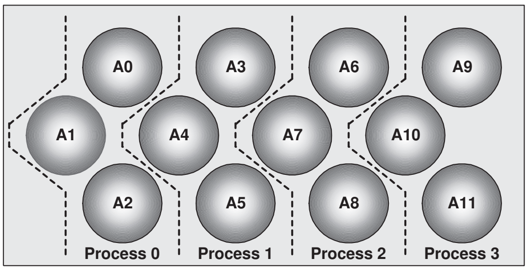


FIG. 5.1. *An instance of LAPW0 domain decomposition. Number of atoms (NAT) is 12; number of atoms per process (PNAT) is 3.*

LAPW0 is executed in SPMD fashion (all processors execute the same program) on a multiprocessor computing system. A domain decomposition approach is used for parallelization of LAPW0 (see Figure 5.1). The unit of material, for which LAPW0 calculates the effective potential, comprises a certain number of atoms (NAT). Atoms are evenly distributed to the available processes. This means that each process is responsible for calculation of the effective potential for a subset of atoms. For NP available processes, each process obtains $PNAT = NAT/NP$ atoms. LAPW0 uses an algorithm that aims to distribute a similar (if not the same) number of atoms to each process for any given positive integer values of the number of atoms and the number of processes.

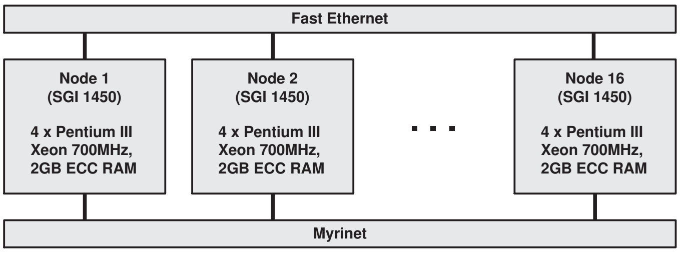
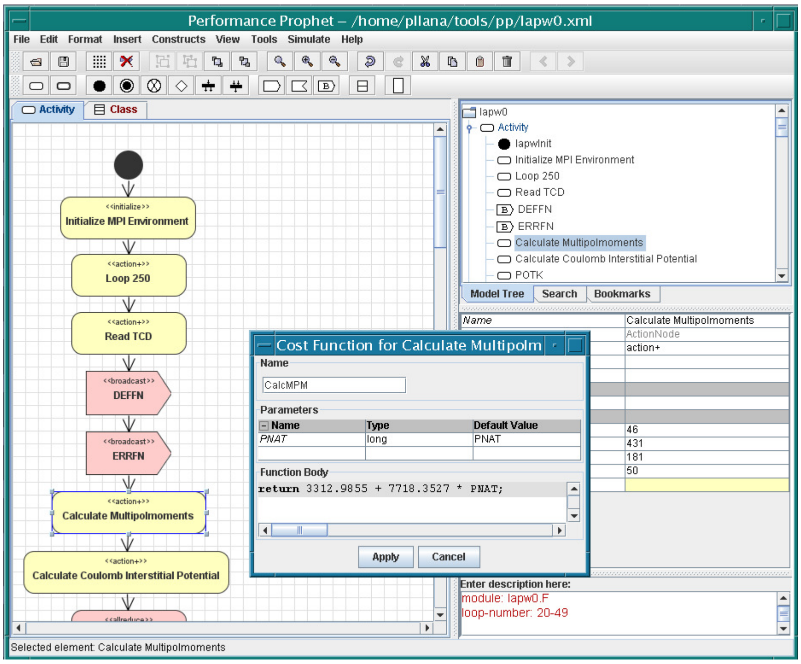


FIG. 5.2. *Experimentation platform. Gescher cluster has 16 SMP nodes. Each node has four processors.*

Figure 5.2 depicts the architecture of Gescher cluster, which is located at Institute of Scientific Computing, University of Vienna. Gescher is a 16 node SMP cluster. All nodes of Gescher are of type SGI 1450. Each node of the cluster has four Pentium III Xeon 700MHz processors , and 2GB ECC RAM. The nodes of Gescher are interconnected via a 100Mbit/s Fast Ethernet network and a Myrinet network. For our experiments we have used the Fast Ethernet network. Gescher serves as our platform for performance measurement experiments of LAPW0.

In what follows in this section we develop and evaluate the model of LAPW0 with Performance Prophet. We validate the model of LAPW0 by comparing simulation results with measurement results.

Figure 5.3 illustrates the procedure for the development of performance model of LAPW0 with Performance Prophet. Due to space limitations, in Figure 5.3 it is depicted just a fragment of the UML model of LAPW0. We developed the model of LAPW0 by using the modeling elements that are available in the toolbar of Performance Prophet. Basically, Performance Prophet permits to associate to each modeling element a cost function. A cost function models the execution time of the code block that is represented by the performance modeling element. Figure 5.3 depicts the association of cost function *CalcMPM* to action *Calculate Multipolmoments.*


Performance Prophet – /home/pllana/tools/pp/lapw0.xml
File Edit Format Insert Constructs View Tools Simulate Help
Activity
Class
<<initialize>>
Initialize MPI Environment
<<action+>>
Loop 250
<<action+>>
Read TCD
<<broadcast>>
DEFFN
<<broadcast>>
ERRFN
<<action+>>
Calculate Multipolmoments
<<action+>>
Calculate Coulomb Interstitial Potential
<<allreduce>>
lapw0
Activity
lapwInit
Initialize MPI Environment
Loop 250
Read TCD
DEFFN
ERRFN
Calculate Multipolmoments
Calculate Coulomb Interstitial Potential
POTK
Model Tree
Search
Bookmarks
Selected element: Calculate Multipolmoments

Cost Function for Calculate Multipolm

Name

CalcMPM

Parameters

Name	Type	Default Value
PNAT	long	PNAT

Function Body

```
return 3312.9855 + 7718.3527 * PNAT;
```

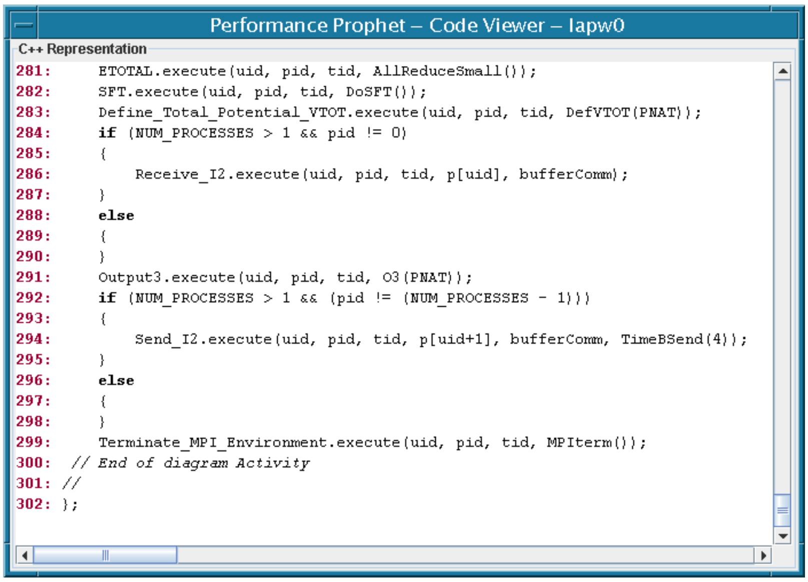
Apply Cancel

Name	Calculate Multipolmoments
	ActionNode
	action+
	46
	431
	181
	50

Enter description here:
module: lapw0.F
loop-number: 20-49

FIG. 5.3. *Performance modeling of LAPW0.*

This cost function was generated based on measurement data by using regression. Regression is a technique for fitting a curve through a set of data values using some goodness-of-fit criterion.



```
    ETOTAL.execute(uid, pid, tid, AllReduceSmall());
    SFT.execute(uid, pid, tid, DoSFT());
    Define_Total_Potential_VTOT.execute(uid, pid, tid, DefVTOT(PNAT));
    if (NUM_PROCESSES > 1 && pid != 0)
    {
        Receive_I2.execute(uid, pid, tid, p[uid], bufferComm);
    }
    else
    {
    }
    Output3.execute(uid, pid, tid, O3(PNAT));
    if (NUM_PROCESSES > 1 && (pid != (NUM_PROCESSES - 1)))
    {
        Send_I2.execute(uid, pid, tid, p[uid+1], bufferComm, TimeBSend(4));
    }
    else
    {
    }
    Terminate_MPI_Environment.execute(uid, pid, tid, MPIterm());
  // End of diagram Activity
//
};
```

FIG. 5.4. *An excerpt of C++ representation of performance model of LAPW0.*

Figure 5.4 depicts an excerpt of C++ representation of performance model of LAPW0 that is generated by Performance Prophet based on the specified UML model. We may observe that C++ representation of performance model includes 302 lines of code. This C++ representation is used as input for the Performance Estimator of Performance Prophet.

We validated the performance model of LAPW0 by comparing simulation results with measurement results for two problem sizes and four system configurations. The problem size is determined by the parameter NAT, which indicates the number of atoms in a unit of the material. We have validated the performance model of

TABLE 5.1

Simulation and measurement results for LAPW0. In NxPy, x denotes the number of nodes N, and y denotes the total number of processes P. T_s is simulated time, T_m is measured time, and T_e evaluation time. All times are expressed in seconds [s].

NAT = 32					
System	T_s [s]	T_m [s]	T_e [s]	T_m/T_e	Error [%]
N1P4	280	264	0.01	26,400	6
N2P8	170	166	0.02	8,300	2
N4P16	126	131	0.04	3,275	3
N8P32	98	113	0.08	1,413	13
NAT = 64					
System	T_s [s]	T_m [s]	T_e [s]	T_m/T_e	Error [%]
N1P4	543	501	0.01	50,100	8
N2P8	314	264	0.02	14,700	7
N4P16	211	197	0.04	4,925	7
N8P32	184	164	0.09	1,822	12

LAPW0 for $NAT = 32$ and $NAT = 64$. The system configuration is determined by the number of nodes and the number of processing units. We have validated the performance model of LAPW0 for the following system configurations: one node and four processes (N1P4), two nodes and eight processes (N2P8), four nodes and 16 processes (N4P16), eight nodes and 32 processes (N8P32). Each node comprises four processors. On each processor is mapped one process.

Table 5.1 depicts simulation and measurement results for LAPW0. The second column of table, which is indicated with T_s, shows the performance prediction results for LAPW0 that we have obtained by simulation. Measurement results of LAPW0 are presented in the third column, which is indicated with T_m. The column that is indicated with T_e presents the CPU time needed for evaluation of the performance model of LAPW0 by simulation. All simulations were executed on a Sun Blade 150 (UltraSPARC-IIe 650MHz) workstation. We compare the time needed to execute the real LAPW0 program on our SMP cluster with the time needed to evaluate the performance model on a Sun Blade 150 workstation in the column that is indicated with T_m/T_s. We may observe that model-based performance evaluation of LAPW0 with Performance Prophet was several thousand times faster than the corresponding measurement-based evaluation. The rightmost column of the table shows the percentage error, which serves to quantify the prediction accuracy of Performance Prophet. We have calculated the percentage error using the following expression,

$$Error[\%] = \frac{|T_s - T_m|}{T_m} 100,$$

where T_s is the simulated time and T_m is the measured time. We may observe that the prediction accuracy of Performance Prophet for LAPW0 was between 2% and 13%. The average percentage error was 7%. Simulation and measurement results for LAPW0 are graphically presented in Figure 5.5.

6. Conclusions. In this paper we have described our methodology for the development of performance models of programs. Our approach supports the graphical specification of performance models in a human-intuitive fashion on one hand, and on the other hand is amenable to the machine-efficient model evaluation. The model transformation, from the graphical human-intuitive form (that is, UML representation), to the form that can be efficiently evaluated by machine (that is, C++ representation), is performed automatically. We have illustrated our methodology with the transformation of a sample performance model using the Performance Prophet modeling system.

The usefulness of our approach has been demonstrated by modeling and simulating LAPW0, which is a real-world material science program that comprises about 15,000 lines of code. In our case study, the model evaluation with Performance Prophet on a single processor workstation was several thousand times faster than the execution time of the real program on our SMP cluster. We validated the model of LAPW0 by comparing the simulation results with measurement results for two problem sizes and four system configurations.

In future we plan to extend our approach to enable the automatic generation of the program code based on the UML model.

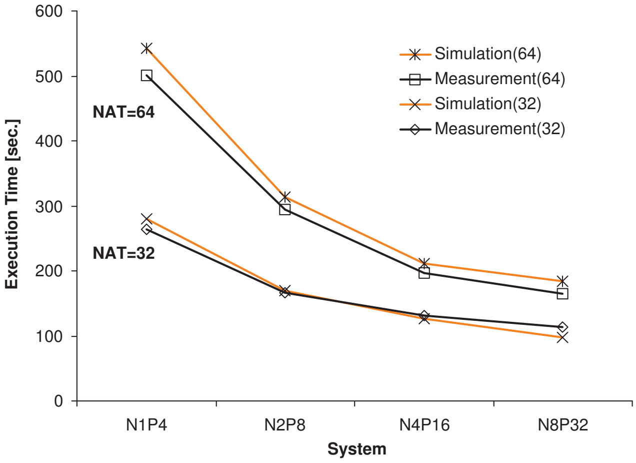


Fig. 5.5. *Simulation and measurement results for LAPW0.*

REFERENCES

[1] V. Adve, R. Bagrodia, J. Browne, E. Deelman, A. Dube, E. Houstis, J. Rice, R. Sakellariou, D. Sundaram-Stukel, P. Teller, and M. Vernon, *POEMS: End-to-End Performance Design of Large Parallel Adaptive Computational Systems*, IEEE Transactions on Software Engineering, 26 (2000), pp. 1027–1048.

[2] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide, Second Edition*, Addison Wesley Professional, 2005.

[3] L. Dagum and R. Menon, *OpenMP: An Industry-Standard API for Shared-Memory Programming*, IEEE Computational Science and Engineering, 5 (1998), pp. 46–55.

[4] S. Graham, M. Snir, and C. Patterson, *Getting Up to Speed: The Future of Supercomputing*, The National Academies Press, 2004.

[5] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI—2nd Edition: Portable Parallel Programming with the Message Passing Interface (Scientific and Engineering Computation)*, MIT Press, 1999.

[6] D. Grove and P. Coddington, *Performance Modeling and Evaluation of High-Performance Parallel and Distributed Systems*, Performance Evaluation, 60 (2005), pp. 165–187.

[7] C. Hughes, V. Pai, P. Ranganathan, and S. Adve, *RSIM: Simulating Shared-Memory Multiprocessors with ILP Processors*, IEEE Computer, 35 (2002), pp. 40–49.

[8] H. Karatza, *Applied System Simulation: Methodologies and Applications*, Springer, 2003, ch. Simulation of Parallel and Distributed Systems Scheduling, Concepts, Issues and Approaches.

[9] D. Kerbyson, A. Hoisie, and H. Wasserman, *Use of Predictive Performance Modeling During Large-Scale System Installation*, Parallel Processing Letters, 15 (2005).

[10] D. Kvasnicka, H. Hlavacs, and C. Ueberhuber, *Simulating Parallel Program Performance with CLUE*, in International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Orlando, Florida, USA, July 2001, The Society for Modeling and Simulation International, pp. 140–149.

[11] F. H. McMahon, *The Livermore Fortran Kernels: A Computer Test of the Numerical Performance Range*, Tech. Report UCRL-53745, Lawrence Livermore National Laboratory, Livermore, California, December 1986.

[12] *Message Passing Interface Forum (MPIF)*. `http://www.mpi-forum.org/`

[13] G. Nudd, D. Kerbyson, E. Papaefstathiou, S. Perry, J. Harper, and D. Wilcox, *PACE—A Toolset for the Performance Prediction of Parallel and Distributed Systems*, International Journal of High Performance Computing Applications, 14 (2000), pp. 228–251.

[14] Object Management Group (OMG), *UML 2.0 Superstructure Specification*. `http://www.omg.org` August 2005.

[15] *Open specifications for Multi Processing (OpenMP)*. `http://www.openmp.org/`

[16] S. Pllana, S. Benkner, F. Xhafa, and L. Barolli, *Hybrid Performance Modeling and Prediction of Large-Scale Computing Systems*, in International Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2008), Barcelona, Spain, March 2008, IEEE Computer Society.

[17] S. Pllana, I. Brandic, and S. Benkner, *A Survey of the State of the Art in Performance Modeling and Prediction of Parallel and Distributed Computing Systems*, International Journal of Computational Intelligence Research (IJCIR), 4 (2008).

[18] S. Pllana and T. Fahringer, *On Customizing the UML for Modeling Performance-Oriented Applications*, in UML 2002, "Model Engineering, Concepts and Tools", LNCS 2460, Dresden, Germany, Dresden, Germany, October 2002, Springer-Verlag.

[19] ———, *UML Based Modeling of Performance Oriented Parallel and Distributed Applications*, in Proceedings of the 2002 Winter Simulation Conference, San Diego, California, USA, December 2002, IEEE.

[20] K. Schwarz, P. Blaha, and G. Madsen, *Electronic structure calculations of solids using the WIEN2k package for material sciences*, Computer Physics Communications, 147 (2002), pp. 71–76.

[21] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha, *A Framework for Performance Modeling and Prediction*, in The 2002 ACM/IEEE conference on Supercomputing, Baltimore, Maryland, USA, November 2002, ACM.

[22] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*, MIT Press, 1998.

Edited by: Fatos Xhafa, Leonard Barolli
Received: September 30, 2008
Accepted: December 15, 2008

Scalable Computing: Practice and Experience
Volume 10, Number 1, pp. 49–60. http://www.scpe.org

ISSN 1895-1767
© 2009 SCPE

REPLAY-BASED SYNCHRONIZATION OF TIMESTAMPS IN EVENT TRACES OF MASSIVELY PARALLEL APPLICATIONS

DANIEL BECKER[*], JOHN C. LINFORD[†], ROLF RABENSEIFNER[‡], AND FELIX WOLF[*]

Abstract. Event traces are helpful in understanding the performance behavior of message-passing applications since they allow in-depth analyses of communication and synchronization patterns. However, the absence of synchronized hardware clocks may render the analysis ineffective because inaccurate relative event timings can misrepresent the logical event order and lead to errors when quantifying the impact of certain behaviors. Although linear offset interpolation can restore consistency to some degree, inaccuracies and time-dependent drifts may still disarrange the original succession of events—especially during longer runs. In our earlier work, we have presented an algorithm that removes the remaining violations of the logical event order postmortem and, in addition, have outlined the initial design of a parallel version. Here, we complete the parallel design and describe its implementation within the Scalasca trace-analysis framework. We demonstrate its suitability for large-scale applications running on more than thousand application processes and evaluate its accuracy by showing that it eliminates inconsistent inter-process timings while preserving the length of local intervals.

1. Introduction. Event tracing is a popular technique for the postmortem performance analysis of message-passing applications because it can be used to investigate temporal relationships between concurrent activities. Obviously, the accuracy of the analysis depends on the comparability of timestamps taken on different processors. Inaccurate timestamps may cause a given interval to appear shorter or longer than it actually was, or change the logical event order, which requires a message to be received only after it has been sent. This is also referred to as the *clock condition*. Besides leading to false conclusions during performance analysis when the impact of certain behaviors is quantified, clock condition violations may confuse the user of trace visualization tools such as Vampir [30] by causing arrows representing messages to point backward in time-line views (Figure 1.1(a)). Moreover, automatic trace-analysis tools such as Scalasca [19] may produce not only inaccurate but outright inconsistent results when calculating certain global metrics that rely on the correct logical order of message events. For example, the output of Scalasca shown in Figure 1.1(b) suggests that the difference between the total time spent in barrier synchronization (i. e., collective synchronization) and the time spent in the barrier before and immediately after the actual synchronization has taken place is negative, which cannot be true.

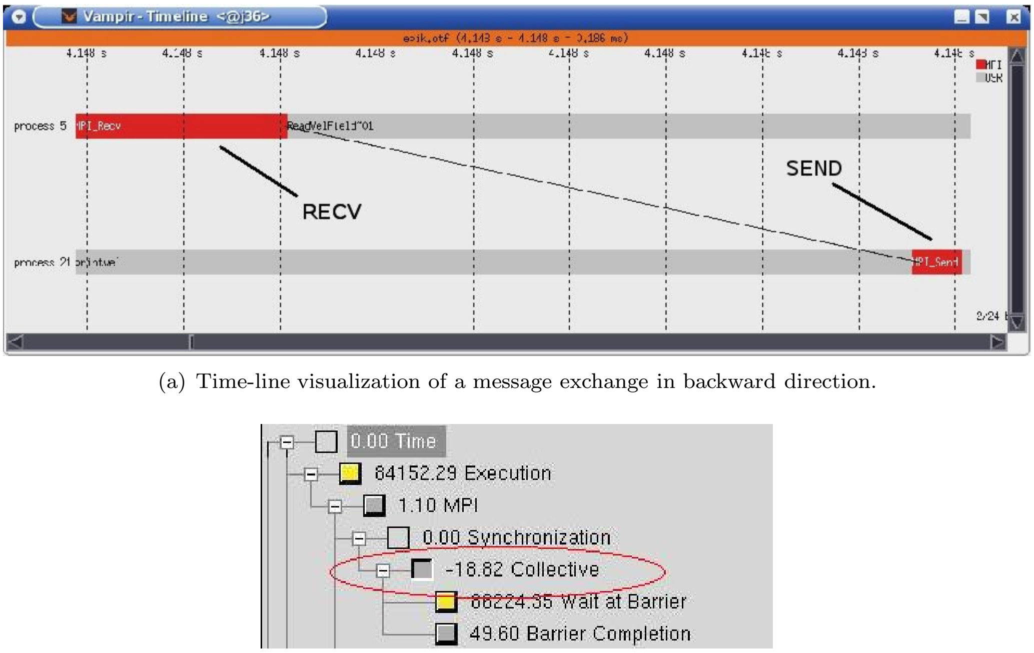
To avoid clock-condition violations, the error of timestamps should ideally be smaller than one half of the message latency. A typical clock quartz with a drift of only 1 min/year will cause a deviation of 2 μs already after 1 second. While some systems such as IBM Blue Gene offer a relatively accurate global clock, many other systems including most PC clusters provide only processor-local clocks that are either entirely non-synchronized or synchronized only within disjoint partitions (e.g., SMP-node or multicore-chip). Clock synchronization protocols such as NTP [29] can align the clocks to a certain degree, but are often not accurate enough for our purposes. Assuming that every local clock on a parallel machine runs at a different but constant speed (i. e., with a different but constant drift), the (global) time of an arbitrarily chosen master clock can be calculated locally as a linear function of the local time. However, as the assumption of constant drifts is only an approximation, violations of the clock condition may still occur - especially when the offset measurements needed for the interpolation are taken with long intervals in between. For example, temperature variations may cause variations of the drift rate of more than 10^{-8}, which will cause synchronization errors of more than 1 μs after 100 seconds execution time. Figure 1.2 shows clock offsets after linear end-to-end interpolation measured using a simple benchmark program that was executed for 500 seconds. One can see that the non-linearity of local clocks caused clock errors larger than the send-receive and allreduce latency.

While the errors of single timestamps are hard to assess, clock-condition violations can be easily detected and offer a toehold to increase the fidelity of inter-process timings. In our earlier work [3], we have presented an algorithm that retroactively corrects timestamps violating the clock condition in event traces of MPI applications. However, in view of rapidly increasing parallelism combined with advances in scalable trace-analysis technology [5, 19, 6], it is crucial that the algorithm scales to large numbers of application processes. For

[*] Jülich Supercomputing Centre, Forschungszentrum Jülich, 52425 Jülich, Germany; Department of Computer Science, RWTH Aachen University, 52056 Aachen, Germany, {d.becker, f.wolf}@fz-juelich.de

[†] Department of Computer Science, Virginia Tech, Blacksburg, VA 24061, USA, jlinford@vt.edu

[‡] High-Performance Computing Center, University of Stuttgart, 70550 Stuttgart, Germany, rabenseifner@hlrs.de



(a) Time-line visualization of a message exchange in backward direction.

0.00 Time
84152.29 Execution
1.10 MPI
0.00 Synchronization
-18.82 Collective
68224.35 Wait at Barrier
49.60 Barrier Completion

(b) A global time metric that must not be negative.

FIG. 1.1. *Inconsistent trace analyses due to insufficiently synchronized timestamps.*

this reason, we have designed and implemented a parallel version of the algorithm and integrated it into the Scalasca performance-analysis framework [1, 18]. Instead of sequentially processing a single global trace file, we follow Scalasca's scalable trace-analysis approach [19] and process separate process-local trace files in parallel by *replaying* the original communication on as many CPUs as have been used to execute the target application itself. Since trace processing capabilities (i. e., processors and memory) grow proportionally with the number of application processes, we can achieve good scalability at very large scales.

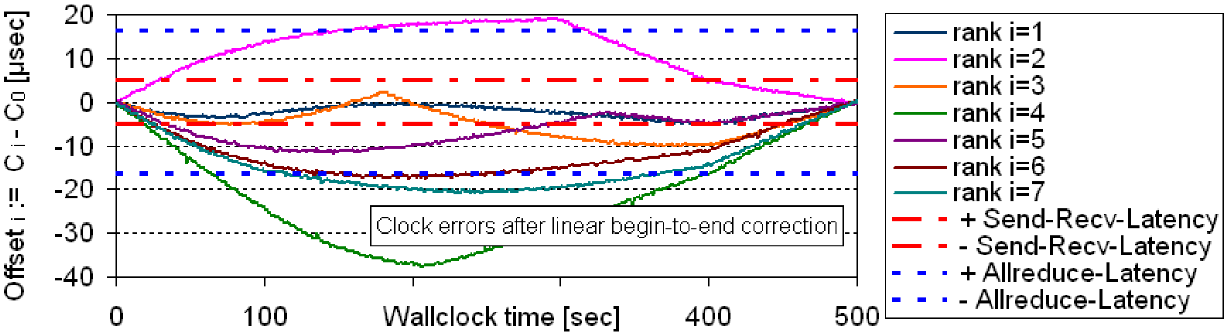


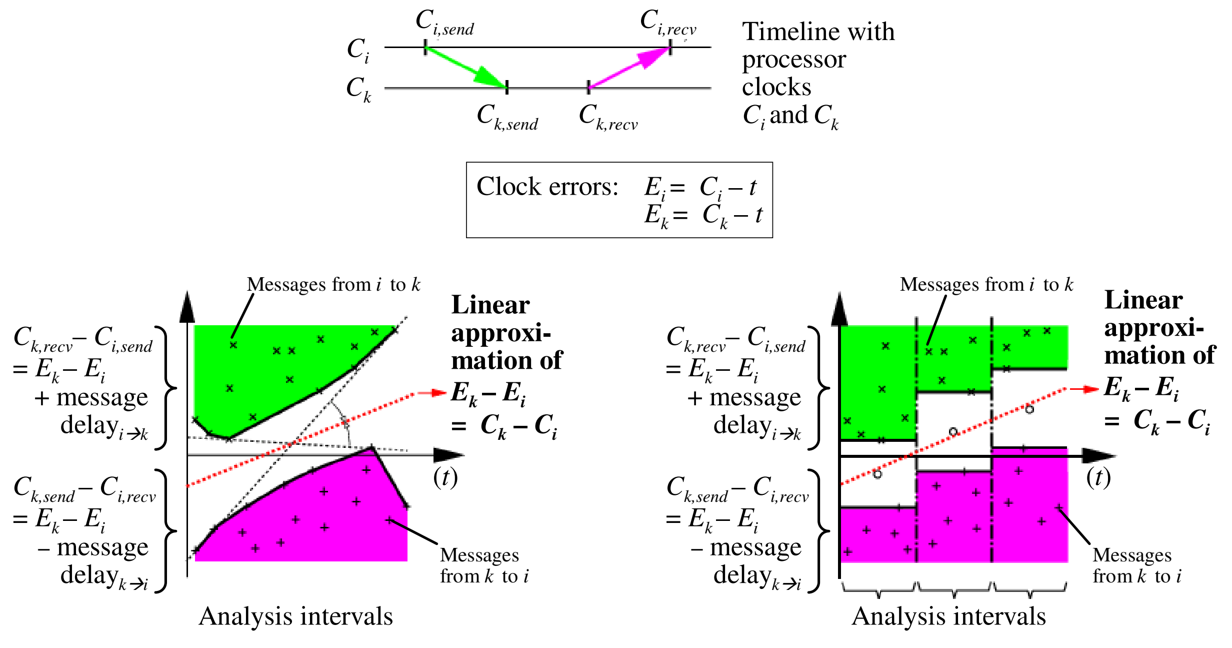
FIG. 1.2. *Non-linear offsets of physical clocks measured on an Infiniband cluster in comparison to the send-receive and allreduce latency.*

The outline of this article is as follows: After reviewing related work in Section 2, we briefly describe the serial version of the algorithm in Section 3. In Section 4, we complete the parallel design outlined in [3] and explain its implementation within the Scalasca trace-analysis infrastructure. We evaluate the scalability of the parallel version in Section 5, where we also show that the collaterally introduced deviations of local interval lengths remain within acceptable limits. Finally in Section 6, we conclude our paper and give an outlook on future work.

2. Related Work. This article describes an approach to retroactively synchronizing timestamps in event traces of parallel programs for the purpose of accurate program observation. While a detailed introduction to event-based program observation as discussed here can be found in Riek et al. [35], we restrict our account of related work to the topic of clock synchronization.

Network-based synchronization protocols aim at synchronizing distributed clocks before reading them. Clocks distributed across the network query the global time from reference clocks, which are often organized in a hierarchy of servers. For instance, NTP [29] uses widely accessible and already synchronized primary time servers. Secondary time servers and clients can query time information via both private networks and the Internet. To reduce network traffic, the time servers are accessed only at regular intervals to adjust the local clock. Jumps are avoided by changing the drift while leaving the actual time unmodified. Unfortunately, varying network latencies limit the accuracy of NTP to about one millisecond compared to a few microseconds required to satisfy the clock condition for MPI applications running on clusters equipped with modern interconnect technology.

Time differences among distributed clocks can be characterized in terms of their relative offset and drift (i. e., the rate at which the offset changes over time). In a simple model assuming different but constant drifts, global time can be established by measuring offsets to a designated master clock using Cristian's probabilistic remote clock reading technique [7]. After estimating the drift, the local time can be mapped onto the global (i. e., master) time via linear offset interpolation. Offset values among participating clocks are measured either at program initialization [13] or at initialization and finalization, and are subsequently used as parameters of the linear correction function [22, 27]. So as not to perturb the program, offset measurements in between are usually avoided, although a recent approach proposes periodic offset measurements during global synchronization operations while limiting the effort required in each step by resorting to indirect measurements across several hops [10]. While linear offset interpolation might prove satisfactory for short runs (or interpolation intervals), measurement errors and time-dependent drifts may create inaccuracies and clock-condition violations during longer runs. Additionally, repeated drift adjustments caused by NTP may impede linear interpolation, as they deliberately introduce non-constant drifts.


$C_{i,send}$
$C_{i,recv}$
C_i
C_k
$C_{k,send}$
$C_{k,recv}$
Timeline with processor clocks C_i and C_k
Clock errors: $E_i = C_i - t$
$E_k = C_k - t$
Messages from i to k
$C_{k,recv} - C_{i,send}$ $= E_k - E_i$ + message delay$_{i \to k}$
$C_{k,send} - C_{i,recv}$ $= E_k - E_i$ – message delay$_{k \to i}$
Linear approximation of $E_k - E_i$ $= C_k - C_i$
(t)
Messages from k to i
Analysis intervals
Messages from i to k
$C_{k,recv} - C_{i,send}$ $= E_k - E_i$ + message delay$_{i \to k}$
$C_{k,send} - C_{i,recv}$ $= E_k - E_i$ – message delay$_{k \to i}$
Linear approximation of $E_k - E_i$ $= C_k - C_i$
(t)
Messages from k to i
Analysis intervals

(a) Duda et al.

(b) Hofmann.

FIG. 2.1. *Algorithms that calculate the clock errors through the differences of the message transfer time in both directions between two processes.*

If linear interpolation alone turns out to be inadequate to achieve the desired level of accuracy, error estimation allows the retroactive correction of clock values in event traces after assessing synchronization errors among all distributed clock pairs. First, difference functions among clock values are calculated from the difference between clock values of receive and send events (plus the minimum message latency). Second, a medial smoothing function can be found and used to correct local clock values because for each clock pair two difference functions exist. Regression analysis and convex hull algorithms have been proposed by Duda et al. [12] to determine the smoothing function. Using a minimal spanning tree algorithm, Jezequel [25] adopted Duda's algorithm for arbitrary processor topologies. In addition, Hofmann [21] improved Duda's algorithm using a simple minimum/maximum strategy and further proposed that the execution time should be divided into several intervals to compensate for different clock drifts in long running applications. Figure 2.1 shows the principles underlying Duda's and Hofmann's algorithms. Using a graph-theory algorithm to calculate the shortest paths, Hofmann and Hilgers [23] simplified Jezequel's algorithm for handling multi-processor topologies. A modification aimed at handling cases of non-existing communication relations between some of the application processes is described in [33]. Biberstein et al. [4] rewrote Hofmann's and Hilgers' algorithm for use on the Cell BE architecture using a short and intelligible notation. Their version solves the clock condition problem only for short intervals (i. e., without splitting into sub-intervals for handling a non-linear drift of the physical clocks). Babaoğlu and Drummond [2, 11] have shown that clock synchronization is possible at minimal cost if the application makes a full message exchange between all processors at sufficiently short intervals. However, jitter in message latency, nonlinear relations between message latency and message length, and one-sided communication topologies limit the usefulness of error estimation approaches. References to additional error estimation approaches can be found in a survey by Yang and Marsland [37].

In contrast, logical synchronization uses happened-before relations among send and receive pairs to synchronize distributed clocks. Lamport introduced a discrete logical clock [26] with each clock being represented as a monotonically increasing software counter. As local clocks are incremented after every local event and the updated values are exchanged at synchronization points, happened-before relations can be exploited to further validate and synchronize distributed clocks. If a receive event appears before its corresponding send event, that is, if a *clock-condition violation* occurs, the receive event is shifted forward in time according to the clock value exchanged. Lamport's discrete logical clock [26] can be used directly for monitoring [8]. Moreover, an algorithm to prevent the drift between the logical clocks has been proposed by Raynal [34]. As an enhancement of Lamport's discrete logical clock, Fidge [15, 16] and Mattern [28] proposed a vector clock. In their scheme, each processor maintains a vector representing all processor-local clocks. While the local clock is advanced with each local event as before, the local vector is updated after receiving a message using an element-wise maximum operation between the local vector and the remote vector that has been sent along with the message. The vector clock is used in some monitoring tools [9, 14] and, in a modified form, also to distinguish in event traces between primary wait states and secondary ones that are merely caused by propagation [24]. Furthermore, global events are introduced in [20], while in [31] spontaneous events (e.g. collisions on a network) are taken into account. Finally, limits of the logical clock and the vector clock are illustrated in [36].

3. Controlled Logical Clock.

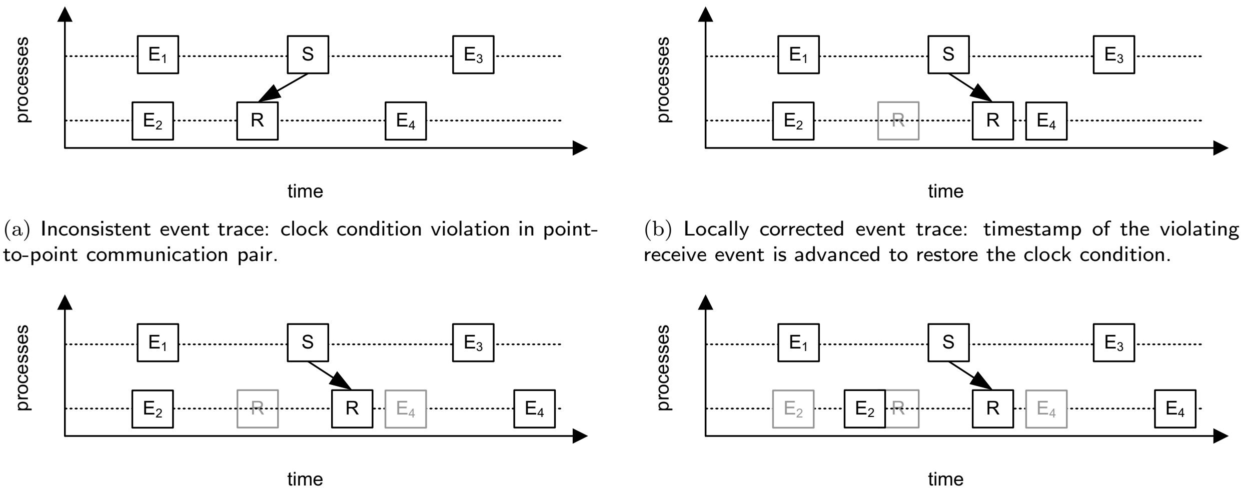
The *controlled logical clock* (CLC) algorithm by Rabenseifner [32, 33] retroactively corrects clock condition violations in event traces of message-passing applications by shifting message events in time while trying to preserve the length of intervals between local events. The algorithm restores the clock condition using happened-before relations derived from message semantics. The clock condition, given in Equation (3.1), requires that a receive event occurs at the earliest l_{min} after the matching send event, with l_{min} being the minimum message latency.

$$t_{recv} \geq t_{send} + l_{min} \tag{3.1}$$

If the condition is violated for a send-receive event pair, the receive event is moved forward in time. To preserve the length of intervals between local events, events following or immediately preceding the corrected event are moved forward as well. These adjustments are called forward and backward amortization, respectively. Note that the accuracy of the adjustment depends on the accuracy of the original timestamps. Therefore, the algorithm benefits from weak pre-synchronization such as the aforementioned linear offset interpolation.

Figure 3.1 illustrates the different steps of the CLC algorithm using a simple example consisting of two processes exchanging a single message. The subfigures show the time lines of the two processes along with their send (S) and receive (R) events, each of them enclosed by two other events (E_i). Figure 3.1(a) shows the initial event trace based on timestamps measured with insufficiently synchronized local clocks. The trace exhibits a violation of the clock condition by having the receive event appear earlier than the matching send event. To restore the clock condition, R is moved forward in time to be l_{min} ahead of S (Figure 3.1(b)). Since now the distance between R and E_4 becomes too short, E_4 is adjusted during the forward amortization to preserve the length of the interval between the two events (Figure 3.1(c)). However, the jump discontinuity introduced by adjusting R affects not only events later than R but also events earlier than R. This is corrected by the backward amortization which shifts E_2 closer to the new position of R, see Figure 3.1(d).

While the forward amortization is at least initially applied to all events following R, the backward amortization applies a linearly increasing correction to a limited amortization interval before R. However, in order to avoid new violations of the clock condition, the correction must not advance any send event located in this interval farther than the matching receive event (minus the minimum message latency). In such a case, we apply the linear correction piecewise, advancing the send events as far as possible and calculating a different slope for each subinterval before, after, or between those sends [3, 33].



(a) Inconsistent event trace: clock condition violation in point-to-point communication pair.

(b) Locally corrected event trace: timestamp of the violating receive event is advanced to restore the clock condition.

(c) Forward amortized event trace: event E_4 following the receive event is adjusted to preserve the length of the interval between both events.

(d) Backward amortized event trace: event E_2 preceding the receive event is advanced to smooth the jump.

FIG. 3.1. *Backward and forward amortization in the controlled logical clock algorithm.*

Note that the algorithm only moves events forward in time. To prevent an increase of the overall time represented by the trace that may occur as a result of a domino-style propagation of forward amortizations, the algorithm applies scaling factors (i. e. control variables) to ensure that the overall error remains within predefined boundaries. The CLC algorithm always tries to advance all processor clocks to the fastest clock when correcting the non-linearity of the clocks. Given that the original timestamps may be logically wrong, this correction leads to logically correct timestamps with marginal local inaccuracies. As a result, timestamp differences between events on different processes normally become more accurate than the original ones because the clocks are advanced to the fastest one. In comparison, the aforementioned algorithms of Duda, Hofmann, and colleagues align the timestamps with the average of the local clocks. However, for monitoring purpose this difference is not significant because it is in the range of the drift rates among local clocks (i.e, in the range of about $10^{-6} - 10^{-4}$). Combined with linear offset interpolation between program start and end, the expected differences are in the range of 10^{-8}.

Since the original (CLC) algorithm takes only point-to-point messages into account, it has been extended in our earlier work [3] to make it applicable to realistic MPI applications that perform not only point-to-point but also collective communication. In our event model, a collective operation instance consists of multiple pairs of enter and exit events (i. e., one pair for each participating process). The basic idea behind extending the CLC algorithm to collective communication is to map collective communication onto point-to-point communication. For this purpose, we consider a single collective operation as a composition of multiple point-to-point operations, taking the semantics of the different flavors of MPI collective operations into account (e.g. *1-to-N*, *N-to-1*, etc.). For instance, in an *N-to-1* operation one root process receives data from N other processes. Given that the root process is not allowed to exit the operation before it has received data from the last process to enter the operation, the clock condition must be observed between the last enter event and the exit event of the root process. Depending on the flavor of the collective operation, different enter and exit events are mapped onto send and receive events, respectively. In reference to the fact that our method is based on logical clocks, we call the send and receive event type assigned during this mapping the *logical event type* as opposed to the actual event type (e.g., enter or collective exit) specified in the event trace.

Until recently, only a serial implementation of the original (CLC) algorithm existed. In the next section, we describe how the extended version of the algorithm has been parallelized and how the parallel version has been integrated into the Scalasca trace-analysis framework.

4. Parallel Timestamp Synchronization. Scalasca, which has been specifically designed for large-scale systems, scans event traces of parallel applications for wait states that occur when processes fail to reach synchronization points in a timely manner, for example, as a result of an unevenly distributed workload. Such wait states can present severe challenges to achieving good performance, especially when trying to scale communication-intensive applications to large processor counts. As a first step towards reducing their impact, Scalasca provides a diagnostic method that allows their localization, classification, and quantification, in particular at larger scales. Scalability is achieved by analyzing the process-local traces in parallel, making Scalasca a parallel program in its own right.

Similar to the wait-state analysis [19] performed by Scalasca, the CLC algorithm requires comparing events involved in the same communication operation, which makes it a suitable candidate for the same parallelization strategy. Instead of sequentially processing a single global trace file, Scalasca processes separate process-local trace files in parallel by *replaying* the original communication on as many CPUs as have been used to execute the target application itself. Since trace processing capabilities (i. e., processors and memory) grow proportionally with the number of application processes, we can achieve good scalability at very large scales. During the replay, sending and receiving processes exchange relevant information needed to analyze the communication operation being replayed. The parallel CLC algorithm is divided into two replay phases: a forward phase for the forward amortization and a backward phase for the backward amortization. The backward phase is only needed if clock condition violations appear during the forward phase.

4.1. Integration with Scalasca. Almost all the postmortem trace-analysis functionality of Scalasca including the parallel CLC algorithm is implemented on top of PEARL [17], a parallel library that offers higher-level abstractions to read and analyze large volumes of trace data. A typical PEARL application is a parallel program having as many processes as the target application had that generated the trace data, resulting in a one-to-one mapping of target application and analysis processes. All analysis processes read the trace data of “their” application process into main memory and traverse the traces in parallel while exchanging information at synchronization points.

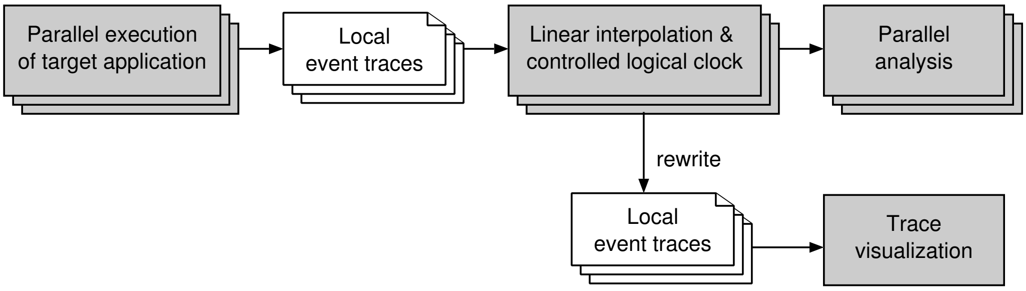

Parallel execution of target application
Local event traces
Linear interpolation & controlled logical clock
Parallel analysis
rewrite
Local event traces
Trace visualization

Fig. 4.1. *Parallel trace-analysis process. Grey rectangles denote programs and white rectangles with the upper right corner turned down denote files. Stacked symbols denote multiple instances of programs or files running or being processed in parallel.*

In Scalasca, the parallel CLC algorithm is applied after the traces have been loaded and before the wait-state analysis takes place. To increase the fidelity of the CLC outcome, the timestamps first undergo a pre-synchronization step. This step performs linear offset interpolation based on offset measurements taken during initialization and finalization of the target application. Once the offset values are known to each analysis process, the interpolation operation is performed locally and does not require any further communication. As an alternative to the native Scalasca wait state analysis, the traces can also be rewritten with modified timestamps, converted, and visualized using Vampir. The full analysis process is illustrated in Figure 4.1.

4.2. Forward Amortization. During the forward phase, the communication replay proceeds in the same direction as it did in the target application. For every pair of logical send and receive events, the sending process sends the timestamp of the logical send event to the receiving process, which compares it to the timestamp of the matching logical receive event (minus the minimum message latency) and, if necessary, applies the forward-amortization equation described in [3]. Recall that, in addition to actual send and receive events, events pertaining to entering or leaving collective communication operations may be classified as logical send or receive events. In this case, the logical event type is derived from the name of the collective operation and the role (e.g., root) a particular process plays in the operation.

In its treatment of events the algorithm distinguishes between (logical) send or receive events and *internal* events that neither send nor receive any kind of message. A different action is performed for each of the three types. Since the correction of an internal event does not require any extra communication, the timestamp adjustment is immediately applied. A send event is adjusted locally and the new timestamp is sent via forward-replay to the receiving process. On the receiver side, the order of these two steps is reversed. The adjusted send timestamp must be obtained from the sender, before the correction can be performed. Finally, the receiver saves detected clock condition violations temporarily along with the associated error so that this information can be reused during the backward amortization phase.

While the direction of inter-process exchange of timestamps is determined by the (logical) type of an event (i. e., send or receive), the actual communication operation invoked to accomplish the transfer depends on the operation originally used by the target application. For this purpose, communication operations are classified according to the number of peers involved on either side: point-to-point, 1-to-N, N-to-1, N-to-N, and two special classes for scan and exscan operations. In brief, point-to-point operations are replayed using point-to-point communication, while collective operations are replayed using different flavors of collective communication.

For the sake of simplicity, our current implementation uses two different values for the minimum message latency l_{min} (see Equation (3.1)): the minimum inter-node and the minimum intra-node latency. Following a conservative approach aimed at avoiding overcorrection, we refrained from considering an extra collective latency, as the duration of collective operations may depend on many factors that are hard to identify, some of them even hidden inside the underlying MPI implementation. Thus, the algorithm requires exchanging the timestamps and the node identifiers to know which of the two latency values must be used.

As mentioned earlier, the CLC algorithm uses so-called control variables. Control variables are scaling factors that are applied to interval expressions when calculating new event timestamps with the purpose of preserving the length of local intervals and avoiding an avalanche-like propagation of corrections [3]. Usually, the control variables are kept less than 1 minus the maximal drift of the clocks. To determine their exact value, however, a global view of the trace data is needed, which is too expensive to establish in our parallel scheme as global communication would be required for every single event. Instead, we approximate a (single) suitable value for all control variables by performing multiple passes of forward replay through the trace data until the maximum error is below a predefined threshold. In practice, more than one pass is seldom needed.

4.3. Backward Amortization. The purpose of the backward amortization phase is to smooth jump discontinuities introduced during the forward amortization by slowly building up the ascension to the jump. This is achieved by applying a process-local linear correction to the interval immediately preceding the jump. However, to preserve the clock condition, the algorithm must not advance the timestamp of any send event located in this interval farther than that of the matching receive event (minus the minimum message latency), leading to the piecewise linear interpolation mentioned earlier. In addition to what has already been stated in our initial design [3], determining these upper limits requires a backward replay, starting at the end of the trace with communication proceeding in backward direction to avoid the danger of deadlocks. While replaying the communication backward, we store with each logical send event the timestamp of the matching receive event after forward amortization. With this information available, an appropriate piecewise linear interpolation function can be calculated for the amortization interval behind every receive event shifted during the forward replay. Note that during the backward amortization the roles of sender and receiver are reversed: the timestamp of a receive event must be available at the process of the matching send event.

4.4. Remark. Given that most MPI implementations use binomial tree algorithms to perform their collective operations, our replay-based approach reduces the communication complexity of replaying collectives automatically to $\mathcal{O}(\log N)$. Moreover, the stepwise parallel replay during the backward amortization phase can, in theory, be replaced by a single collective operation per communicator for the entire trace, but would impose impractical memory requirements. For the actual operations used during both replay phases and the timestamps being exchanged, please refer to [3].

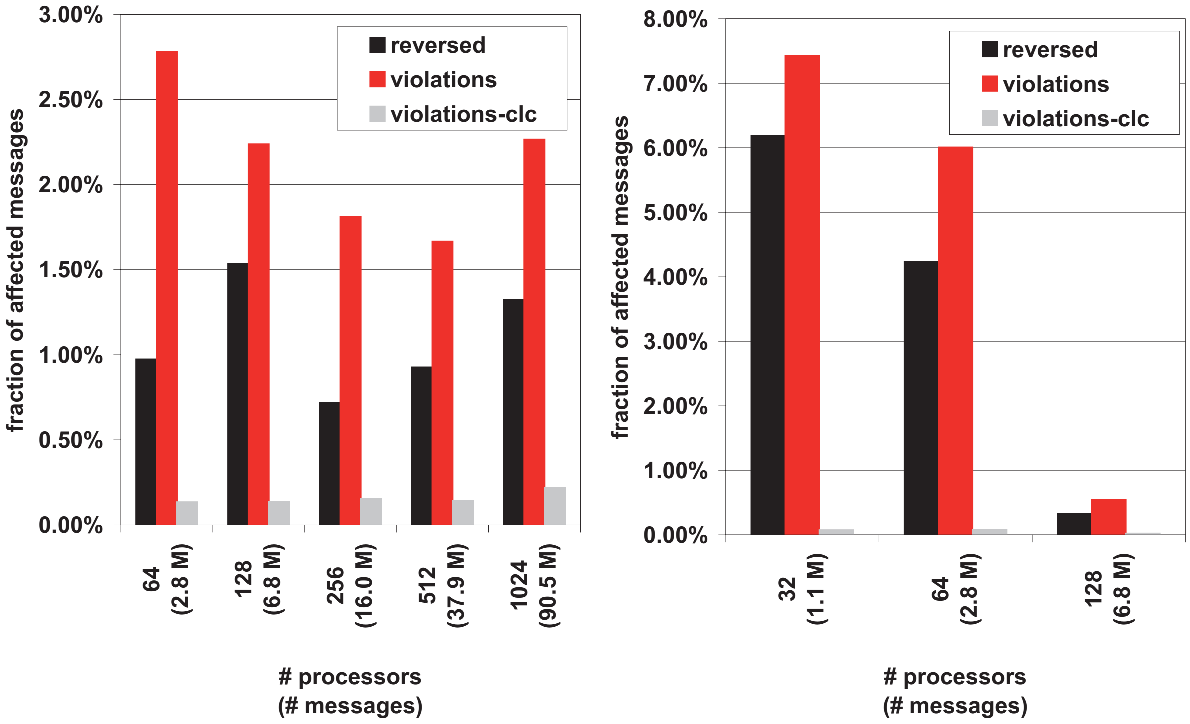
5. Experimental Evaluation. Here we evaluate the scalability and accuracy of the parallel controlled logical clock algorithm and also give evidence of the frequency and the extent of clock condition violations in event traces of a realistic MPI application. We ran experiments on the following two platforms:

MareNostrum consists of 2560 JS21 blade computing nodes, each with 2 dual-core IBM 64-bit PowerPC 970MP processors running at 2.3 GHz. The computing nodes of MareNostrum communicate primarily through a Myrinet network with Myrinet adapters integrated on each server blade. The measured MPI inter-node latency was $7.7\mu s$, the measured MPI intra-node latency was $1.3\mu s$.

CACAU consists of 200 compute nodes, each with 1 dual-core Intel Xeon EM64T CPU running at 3.2GHz. The nodes are linked with a Voltaire Infiniband Network and a Gigabit Ethernet. The measured MPI inter-node latency was $4.7\mu s$, the measured MPI intra-node latency was $1.0\mu s$.

As a test application, we used the MPI version of the ASC SMG2000 benchmark, a parallel semi-coarsening multigrid solver that uses a complex communication pattern and performs a large number of non-nearest-neighbor point-to-point communication operations. Applying a weak scaling strategy, a fixed 16 × 16 × 8 problem size per process with five solver iterations was configured.

While linear interpolation can remove most of the clock condition violations in traces of short runs, it is usually insufficient for longer runs. We therefore emulated a longer run by inserting sleep statements immediately before and after the main computational phase so that it was carried out ten minutes after initialization and ten minutes before finalization. This corresponds to a scenario, in which only distinct intervals of a longer run are traced with tracing being switched off in between. Since full traces of long running applications may consume a prohibitive amount of storage space, the "partial" tracing emulated here mimics the recommended practice of tracing only pivotal points that warrant a more detailed analysis. For our purposes, the artificial chronological distance to the offset measurements on either end of the run adjusted the interpolation interval to roughly twenty minutes execution time. However, with many realistic codes running for hours, this can still be regarded as an optimistic assumption. Compared to true partial tracing of a longer SMG2000 run, our method had the advantage that the total runtime including the actual computational activity and therefore the distance between the two offset measurements was roughly the same for all configurations.



(a) MareNostrum. (b) Cacau.

FIG. 5.1. *Percentage of messages with the order of send and receive events being reversed, of messages with clock condition violations, and of clock condition violations explicitly corrected by the* CLC *algorithm during forward amortization.*

Table 5.1
Average and maximum errors of message events in reversed order.

Platform	Avg. error [μs]	Max. error [μs]
MareNostrum	2.6	323
Cacau	4.3	186

Figure 5.1 shows the frequency of clock condition violations on MareNostrum and Cacau for a range of scales. Since the number of violations varies between runs, the numbers represent averages across three measurements for each configuration. The numbers show the percentage of messages with the order of send and receive events being reversed in the original trace, of messages with clock condition violations ($t_{recv} < t_{send} + l_{min}$) in the original trace, and of clock condition violations explicitly corrected by the CLC algorithm during forward amortization. We also counted logical messages that can be derived by mapping collective communication onto point-to-point semantics. When visualized, messages with the order of send and receive events being reversed seem to flow backward in time. The number of violations explicitly corrected by the CLC algorithm is usually smaller than the initial number of violations because some of them are already implicitly removed during forward amortization before a correction can be applied. On MareNostrum, around 1% of the messages flow backward in time, while on Cacau the percentage ranges between 1 and 6%. Higher latencies on MareNostrum offer a potential explanation for the smaller number of violations detected on this system because higher latencies naturally insert a larger temporal distance between send and receive events of the same message. Although the number of inconsistent messages on Cacau seem to decrease with growing numbers of processes, the results on MareNostrum do not confirm a clear correlation between the two indicators. Table 5.1 lists the average and maximum displacement errors (i. e. the time the receive event appears earlier than the send event) of message events in backward order, as seen in the original trace.

5.1. Scalability. Because it is the larger system, we evaluated our algorithm's scalability on MareNostrum. According to Figure 5.2, the parallel timestamp synchronization, the Scalasca wait-state analysis, and the execution time of SMG2000 itself exhibit roughly equivalent scaling behavior - a result of the replay-based nature of the two analysis mechanisms and the communication-bound performance characteristics of SMG2000. The fact that the total time needed by the integrated Scalasca analysis (synchronization and wait-state analysis) including loading the traces grows more steeply suggests that I/O will increasingly dominate the overall behavior beyond 1024 processes, rendering the additional cost of the synchronization negligible.

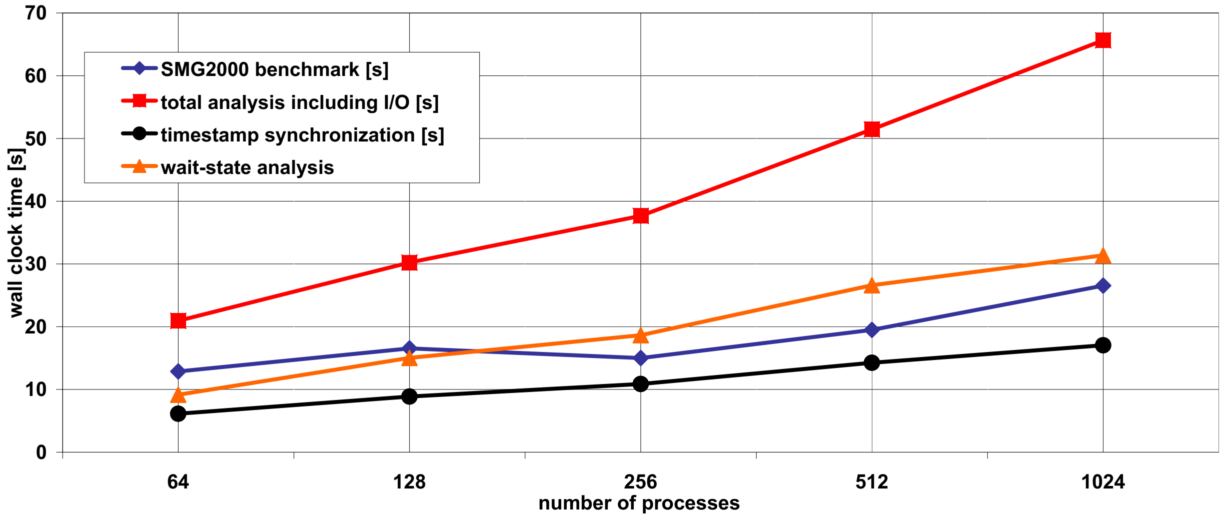


Fig. 5.2. *Scalability of parallel timestamp synchronization on MareNostrum.*

5.2. Accuracy. The transformation performed by the CLC algorithm raises the question of how accurate the modified traces actually are. To answer this question, it must first be acknowledged that traces with clock-condition violations are inaccurate because they are inconsistent. The behavior they reflect violates causation and is therefore impossible. The CLC algorithm eliminates these inconsistencies, improving the accuracy of inter-process timings. A very simple metric quantifying this improvement is the fraction of clock-condition violations found in the original trace (second row in Figure 5.1), which are removed in the modified trace.

However, the necessary corrections applied in the course of the algorithm also modify—as a collateral effect—relative process-local event timings, which may lead to differences in the lengths of local intervals when comparing the original with the modified trace. Again, since the original timestamps have been taken with clocks that exhibit unstable drifts, there is, of course, no guarantee that the measured interval lengths are correct, although the deviation can be expected to be small in comparison to the total interval length. In addition, the original timings are all we "know" about the target application's local execution behavior. For this reason, we decided to evaluate how this knowledge has been preserved in the modified trace.

To assess the fidelity of process-local timings after applying the CLC algorithm, we determined the relative deviation of local interval lengths, considering two different types of intervals: (i) intervals between an event and the first event of the same process, which roughly corresponds to the timestamp of an event relative to the begin of the trace and (ii) intervals between adjacent process-local events (i. e., intervals between an event and its immediate successor). We refer to the first interval type as the *event position* and to the second interval type as the *event distance.*

The maximum relative deviation of the event position across all our measurements was below 0.0001 %; the maximum absolute deviation was 425.18 μs, roughly corresponding to the maximum displacement error observed (Table 5.1). Table 5.2 shows the relative deviation of the event distance across different combinations of platform and number of processors. The numbers in individual columns are percentages and are rounded to two digits after the decimal point. They represent the maximum across three measurements. To account for the relatively long "correct" stretches artificially introduced by the sleep statements before and after the main computation, only the middle section of the trace between the sleep statements was considered. Furthermore, since deviations in larger intervals are more relevant to performance analysis than those in smaller intervals, the average was calculated using $\sum|\Delta t|/\sum|t|$ to assign appropriate weight to larger intervals, with Δt being the absolute deviation and t being the original interval length.

Table 5.2
Relative deviation of event distance. All numbers are given in percent and rounded to two digits after the decimal point.

	MareNostrum					**Cacau**		
# CPUs	64	128	256	512	1024	32	64	128
Average	0.00	0.01	0.01	0.00	0.01	0.00	0.01	0.00
Maximum	27.17	461.74	411.52	311.64	974.44	82.78	69.31	16.77
Percentage of intervals with deviation above threshold								
> 0%	80.80	96.21	97.34	98.23	99.07	92.43	93.67	24.27
> 0.01%	0.15	0.63	0.39	0.48	0.81	0.47	0.27	0.04
> 0.1%	0.15	0.61	0.38	0.46	0.79	0.45	0.26	0.04
> 1%	0.01	0.10	0.06	0.07	0.18	0.09	0.04	0.00
> 10%	0.00	0.01	0.00	0.00	0.01	0.01	0.00	0.00
> 100%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Percentage of execution time consumed by intervals with deviation above threshold								
> 0%	80.51	96.22	96.86	92.20	95.17	50.37	92.50	23.00
> 0.01%	0.41	0.95	0.58	0.71	0.72	0.24	0.68	0.05
> 0.1%	0.20	0.23	0.28	0.21	0.24	0.19	0.55	0.04
> 1%	0.01	0.02	0.03	0.02	0.02	0.04	0.11	0.00
> 10%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
> 100%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

It can be seen that in spite of very small averages, deviations of occasionally more than 100% are still possible. Although the backward amortization is designed to smooth sudden jumps introduced by the forward amortization, it can happen that a send event cannot be advanced far enough without causing a new clock condition violation when passing the corresponding receive event. To evaluate frequency and extent of such situations, we calculated (i) the percentage of intervals whose deviation exceeds a certain threshold and (ii) the percentage of execution time (accumulated across all processes) consumed by intervals whose deviation exceeds the threshold. The results given in Table 5.2 indicate that larger deviations are rare and that their influence on performance-analysis results will usually be negligible.

6. Conclusion. Event traces of parallel applications may suffer from inaccurate timestamps in the absence of synchronized hardware clocks. As a consequence, the analysis of such traces may yield wrong quantitative and even qualitative results or confuse the user of time-line visualizations with messages flowing backward in time. Because linear offset interpolation based on offset measurements can account for such deficiencies only for very short runs, we have designed and implemented a parallel algorithm for the retroactive correction of timestamps based on logical clocks that eliminates inconsistent inter-process timings while only marginally changing the length of local intervals. Our replay-based implementation scales easily to more than thousand application processes and shows potential for even larger configurations. The algorithm has been incorporated into the Scalasca framework to facilitate trace analyses of longer runs on larger cluster systems. In the future, we want to extend our algorithm to hybrid applications employing a mix of message passing and shared-memory parallelism, which will require paying attention to happen-before relationships, for example, imposed by OpenMP barrier or lock event semantics.

Acknowledgment. This work has been funded by the Helmholtz Association under Grant No. VH-NG-118. The authors also thankfully acknowledge the computer resources, technical expertise and assistance provided by the Barcelona Supercomputing Center. In particular, we would like to express our gratitude to Judit Gimenez, Jesus Labarta, and David Vicente for their generous support.

REFERENCES

[1] *Scalasca*. `http://www.scalasca.org/`.

[2] O. Babaoğlu and R. Drummond, *(Almost) no cost clock synchronization*, in Proceedings of 7th International Symposium on Fault-Tolerant Computing, IEEE Computer Society Press, July 1987, pp. 42–47.

[3] D. Becker, R. Rabenseifner, and F. Wolf, *Timestamp synchronization for event traces of large-scale message-passing applications*, in Proc. of the 14th European PVM/MPI Users' Group Meeting, Paris, France, vol. 4757 of Lecture Notes in Computer Science, Springer, September-October 2007, pp. 315–325.

[4] M. Biberstein, Y. Harel, and A. Heilper, *Clock synchronization in Cell BE traces*, in Proc. of the 14th Euro-Par Conference, Las Palmas de Gran Canaria, Spain, vol. 5168 of Lecture Notes in Computer Science, Springer, August - September 2008, pp. 3–12.

[5] H. Brunst and W. E. Nagel, *Scalable performance analysis of parallel systems: Concepts and experiences*, in Proceedings of the Parallel Computing Conference (ParCo), Dresden, Germany, 2003.

[6] A. Chan, W. Gropp, and E. Lusk, *Scalable log files for parallel program trace data (draft)*, tech. report, Argonne National Laboratory, 2000.

[7] F. Cristian, *Probabilistic clock synchronization*, Distributed Computing, 3 (1989), pp. 146–158.

[8] J. E. Cuny, A. A. Hough, and J. Kundu, *Logical time in visualizations produced by parallel programs*, in Proceedings of Visualization '92, Boston, MA, USA, IEEE Computer Society Press, October 1992, pp. 186–193.

[9] G. v. Dijk and A. v. d. Wal, *Partial ordering of sychronization events for distributed debugging in tightly-coupled multiprocessor systems*, in 2nd European Conference on Distributed Memory Computing (EDMCC2), Munich, Germany, vol. 487 of Lecture Notes in Computer Science, Springer, April 1991, pp. 100–109.

[10] J. Doleschal, A. Knüpfer, M. S. Müller, and W. E. Nagel, *Internal timer synchronization for parallel event tracing*, in Proceedings 15th European PVM/MPI Users' Group Meeting, Dublin, Ireland, Lecture Notes in Computer Science, Dublin, Ireland, September 2008, Springer.

[11] R. Drummond and O. Babaoğlu, *Low-cost clock synchronization*, Distributed Computing, 6 (1993), pp. 193–203.

[12] A. Duda, G. Harrus, Y. Haddad, and G. Bernard, *Estimating global time in distributed systems*, in Proceedings of the 7th International Conference on Distributed Computing Systems, Berlin, Germany, IEEE Computer Society Press, September 1987, pp. 299–306.

[13] T. H. Dunigan, *Hypercube clock synchronization*, Technical Report ORNL TM-11744, Oak Ridge National Laboratory, TN, February 1991.

[14] D. Edwards and P. Kearns, *DTVS: A distribute trace visualization system*, in Proceedings of the 6th IEEE Symposium on Parallel and Distributed Processing, Dallas, TX, IEEE Computer Society Press, October 1994, pp. 281–288.

[15] C. J. Fidge, *Timestamps in message-passing systems that preserve partial ordering*, in Proceedings of 11th Australian Computer Science Conference, February 1988, pp. 56–66.

[16] ———, *Partial orders for parallel debugging*, ACM SIGPLAN Notices, 24 (1989), pp. 183–194.

[17] M. Geimer, F. Wolf, A. Knüpfer, B. Mohr, and B. J. N. Wylie, *A parallel trace-data interface for scalable performance analysis*, in Proceedings of the Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA), Umeå, Sweden, vol. 4699 of Lecture Notes in Computer Science, Springer, June 2006, pp. 398–408.

[18] M. Geimer, F. Wolf, B. J. N. Wylie, E. Abraham, D. Becker, and B. Mohr, *The Scalasca performance toolset architecture*, in Proceedings of the Int'l Workshop on Scalable Tools for High-End Computing (STHEC), Kos, Greece, June 2008.

[19] M. Geimer, F. Wolf, B. J. N. Wylie, and B. Mohr, *Scalable parallel trace-based performance analysis*, in Proc. 13th European PVM/MPI Users' Group Meeting, Bonn, Germany, vol. 4192 of Lecture Notes in Computer Science, Springer, September 2006, pp. 303–312.

[20] D. Haban and W. Weigel, *Global events and global breakpoints in distributed systems*, in Proceedings of the 21st Hawaii International Conference on System Sciences, 1988, pp. 166–175, vol. II.

[21] R. Hofmann, *Gemeinsame Zeitskala für lokale Ereignisspuren*, in Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen, 7. GI/ITG-Fachtagung, Aachen, Germany, Springer, September 1993.

[22] ———, *Gesicherte Zeitbezüge für die Leistungsanalyse in parallelen und verteilten Systemen*, PhD thesis, Universität Erlangen-Nürnberg, Technische Fakultät, 1993.

[23] R. Hofmann and U. Hilgers, *Theory and tool for estimating global time in parallel and distributed systems*, in Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Processing, Madrid, Spain, January 1998, pp. 173–179.

[24] H. Jafri, *Measuring causal propagation of overhead of inefficiencies in parallel applications*, in Proceedings of the 19th IASTED International Conference on Parallel and Distributed Computing and Systems, Cambridge, MA, November 2007, pp. 237–243.

[25] J.-M. Jézéquel, *Building a global time on parallel machines*, in Proceedings of the 3rd International Workshop on Distributed Algorithms, J.-C. Bermond and M. Raynal, eds., vol. 392 of Lecture Notes in Computer Science, Springer, 1989, pp. 136–147.

[26] L. Lamport, *Time, clocks, and the ordering of events in a distributed system*, Communications of the ACM, 21 (1978), pp. 558–565.

[27] E. Maillet and C. Tron, *On efficiently implementing global time for performance evaluation on multiprocessor systems*, Journal of Parallel and Distributed Computing, 28 (1995), pp. 84–93.

[28] F. Mattern, *Virtual time and global states of distributed systems*, in Proceedings of the International Workshop on Parallel and Distributed Algorithms, Chateau de Bonas, France, M. Cosnard and P. Quinton, eds., Elsevier, October 1989, pp. 215–226.

[29] D. L. Mills, *Network Time Protocol (Version 3).* The Internet Engineering Task Force - Network Working Group, March 1992. RFC 1305.

[30] W. E. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach, *VAMPIR: Visualization and analysis of MPI resources*, Supercomputer 63, XII (1996), pp. 69–80.

[31] R. L. Probert, H. Yu, and K. Saleh, *Relative-clock-based specification and test result analysis of distributed systems*, in Eleventh Annual International Phoenix Conference on Computers and Communications, Scottsdale, AZ, USA, IEEE, New York, April 1992, pp. 687–694.

[32] R. Rabenseifner, *The controlled logical clock - a global time for trace based software monitoring of parallel applications in workstation clusters*, in Proceedings of the 5th EUROMICRO Workshop on Parallel and Distributed (PDP'97), London, UK, January 1997, pp. 477–484.

[33] ———, *Die geregelte logische Uhr, eine globale Uhr für die tracebasierte Überwachung paralleler Anwendungen (The controlled logical clock, a global clock for trace-based monitoring of parallel applications)*, PhD thesis, University of Stuttgart, Faculty of Computer Science, March 2000. `http://elib.uni-stuttgart.de/opus/volltexte/2000/600/`.

[34] M. Raynal, *A distributed algorithm to prevent mutual drift between n logical clocks*, Information Processing Letters, 24 (1987), pp. 199–202.

[35] M. v. Riek, B. Tourancheau, and X.-F. Vigouroux, *Monitoring of distributed memory multicomputer programs (A general approach to the monitoring of distributed memory MIMD multicomputers)*, Technical Report CS-93-204, University of Tennessee, 1993. `http://www.netlib.org/tennessee/ut-cs-93-204.ps`.

[36] R. Schwarz and F. Mattern, *Detecting causal relationships in distributed computations: in search of the holy grail*, Distributed Computing, 7 (1994), pp. 149–174.

[37] Z. Yang and T. A. Marsland, *Annotated bibliography on global states and times in distributed systems*, Operating Systems Review, 27 (1993), pp. 55–74.

Edited by: Fatos Xhafa, Leonard Barolli
Received: September 30, 2008
Accepted: December 15, 2008

Scalable Computing: Practice and Experience
Volume 10, Number 1, pp. 61–74. http://www.scpe.org

ISSN 1895-1767
© 2009 SCPE

PERFORMANCE ANALYSIS AND OPTIMIZATION OF PARALLEL SCIENTIFIC APPLICATIONS ON CMP CLUSTERS

XINGFU WU, VALERIE TAYLOR, CHARLES LIVELY AND SAMEH SHARKAWI*

Abstract. Chip multiprocessors (CMP) are widely used for high performance computing. Further, these CMPs are being configured in a hierarchical manner to compose a node in a cluster system. A major challenge to be addressed is efficient use of such cluster systems for large-scale scientific applications. In this paper, we quantify the performance gap resulting from using different number of processors per node; this information is used to provide a baseline for the amount of optimization needed when using all processors per node on CMP clusters. We conduct detailed performance analysis to identify how applications can be modified to efficiently utilize all processors per node using three scientific applications: a 3D particle-in-cell, magnetic fusion application Gyrokinetic Toroidal Code (GTC), a Lattice Boltzmann Method for simulating fluid dynamics (LBM), and an advanced Eulerian gyrokinetic-Maxwell equation solver for simulating microturbulent transport in plasma (GYRO). In terms of refinements, we use conventional techniques such as loop blocking, loop unrolling and loop fusion, and develop hybrid methods for optimizing MPI_Allreduce and MPI_Reduce. Using these optimizations, the application performance for utilizing all processors per node was improved by up to 18.97% for GTC, 15.77% for LBM and 12.29% for GYRO on up to 2048 total processors on the CMP clusters.

Key words: performance analysis, performance optimization, chip multiprocessors (CMP), clusters, parallel scientific applications

1. Introduction. The current trend in high performance computing systems has been shifting towards cluster systems with CMPs (chip multiprocessors). Further, the CMPs are usually configured hierarchically (e.g., multiple CMPs compose a multi-chip module and multiple multi-chip modules compose a node) to compose a node of a CMP cluster. For example, each node of DataStar P690 at the San Diego Supercomputing Center (SDSC) consists of four multi-chip modules for which each module consists of four CMPs, and each CMP consists of two cores [13]. SDSC DataStar p655 has one multi-chip module per node. When using these clusters to execute a given application, one issue to be addressed is how to efficiently utilize all processors per node given the significant sharing of node resources (e.g., caches, networks) among the processors within the node. In this paper, we quantify the performance gap resulting from using different number of processors per node for application executions. Further, we use the detailed performance results to identify performance optimizations that can be made to these applications for efficient execution using all processors per node.

Other work in this area has focused on using all processors per node. Phillips et al. [10] presented the performance results for 4 processors per node and 3 processors per node on Lemieux Alpha cluster at Pittsburgh Supercomputing Center that has a maximum of 4 processors per node, and noted that leaving idle one processor per node reduces performance variability. Petrini et al. [9] found that application execution times may vary significantly between 3 processors per node and 4 processors per node on a large scale supercomputer, ASCI Q. They concluded that system noise within the nodes was the source of the performance variability, and used a discrete-event simulator to evaluate the contribution of each component of the noise to the overall application behavior. In our previous work [20], we quantified the performance gap resulting from using different number of processors per node for the NAS parallel benchmarks on SMP clusters. In this paper, however, we identify optimizations that result in efficient use of all processors per node on the CMP clusters for large-scale scientific applications.

The three large-scale, scientific applications used for our experimental analysis are as follows: a 3D particle-in-cell application Gyrokinetic Toroidal Code (**GTC**) in magnetic fusion [2], a Lattice-Boltzmann Method for simulating fluid dynamics (**LBM**) [19], and an advanced Eulerian gyrokinetic-Maxwell equation solver for simulating microturbulent transport in core plasma (**GYRO**) [3]. The Prophesy system [17, 16] is used to collect all application performance data. The programming environments and problem sizes used in the three applications are shown in Table 1.1. The test case studied for GTC is 100 particles per cell and 100 time steps. The problem size for LBM is a 3D mesh computational domain. B1-std and B2-cy are two benchmark datasets for GYRO.

The experiments conducted in this work utilize systems with different number of processors per node. DataStar P655 has 8 processors per node and P690 has 32 processors per node. Bassi has 8 processors per node and Seaborg has 16 processors per node at the DOE National Energy Research Scientific Computing Center

*Department of Computer Science, Texas A&M University, College Station, TX 77843, USA ({wuxf, taylor}@cs.tamu.edu)

Table 1.1
Programming environments and problem sizes of GTC, LBM and GYRO

Application	Discipline	Problem Sizes	Programming Environments
GTC	Magnetic Fusion	100 particles per cell	Fortran90, MPI, OpenMP
LBM	Fluid Dynamics	128x128x128 512x512x512	C, MPI
GYRO	Plasma Physics	B1-std: 6x140x4x4x8x6 B2-cy: 6x128x4x4x8x6	Fortran90, MPI

(NERSC) [8]. BlueGene/L at the Renaissance Computing Institute has 2 processors per node [11]. Further, each system has a different node memory hierarchy. The experimental results indicate that large performance gaps can exist. For example, the performance gap for LBM using 32 processors corresponds to an increase of 23.53% when using 32 processors per node versus using 8 processors per node (resulting from resource conflicts) for a problem size of 128x128x128 on the P690.

Much of the computation involved in parallel scientific applications occurs within nested loops. Therefore, loop optimization is fundamentally important for these applications. In this paper, we use loop blocking, loop unrolling and loop fusion to optimize three scientific applications. We also develop hybrid methods to optimize MPI_Allreduce and MPI_Reduce, which are very common communication subroutines. For a given optimization of GTC, the application performance was improved by up to 18.97% on up to 2048 processors. For a given optimization of LBM, the application performance was improved by up to 15.77% when utilizing all processors per node. For a given optimization of GYRO, the application performance was improved by up to 12.29% when utilizing all processors per node. This is important because with CMP clusters, the only way to access a large number of processors is to use all processors per node.

The remainder of this paper is organized as follows. Section 2 discusses the architecture and memory hierarchy of the five supercomputers used in our experiments, and presents their MPI performance when utilizing different configurations of the node hierarchy. Section 3 discusses application optimization methods used to refine the applications. Section 4 investigates performance characteristics of GTC, and presents our optimization results. Section 5 discusses performance characteristics and optimization results of LBM. Section 6 explores performance characteristics and optimization results of GYRO. Section 7 concludes this paper. In the remainder of this paper, we describe a processor partitioning scheme as **MxN whereby M denotes the number nodes with N processors per node (PPN)**. The job scheduler for each supercomputer always dispatches one process to one processor. All experiments were executed multiple times to insure consistency of the performance data.

2. Execution Platforms and Corresponding MPI Performance. Details about the five supercomputers used for our experiments are given in Table 2.1. These systems differ in the following main features: number of processors per node, configurations of node memory hierarchy, CPU speed, multi-core processors, and communication networks.

DataStar P655 [13] has 176 (8-way) compute nodes with 1.5GHz POWER4 and 16GB memory, and 96 (8-way) compute nodes with 1.7GHz POWER4 and 32GB memory. Each node of P655 has one MCM (Multiple-Chip Module) with 4 chips per MCM. DataStar P690 has 7 (32-way) compute nodes. Each node of P690 has four MCMs with 4 chips per MCM. The use of 8-way nodes for P655 is exclusive. The use of 32-way nodes on P690 is shared among users, and the access to P690 is limited to at most five nodes.

UNC RENCI BlueGene/L [11] is a system-on-chip supercomputer, and is designed to achieve high performance for low cost and with low power consumption. The BlueGene/L has 1024 dual 700 MHz PowerPC 440 nodes with 1GB memory per node. Note that L2 cache is a prefetch buffer that holds 16 128-byte lines.

Seabog [8] is a distributed memory computer with 6,080 processors available to run scientific applications. The processors are distributed among 380 compute nodes with 16 PPN and a shared memory pool of size 16-64GB (312 nodes have 16GB; 64 nodes have 32GB; 4 nodes have 64GB) per node. The use of 16-way nodes is exclusive. Bassi [8] is a distributed shared memory computer with 888 processors available to run scientific applications. The processors are distributed among 111 compute nodes with 8 PPN and a shared memory pool of 32GB per node. The use of 8-way single-core POWER5 nodes is also exclusive. Only Bassi is configured to use 20GB large page on each node. The large page uses hardware prefetch mechanisms to eliminate costly TLB misses at the expense of an increase in process start-up time.

Table 2.1
Specifications of five supercomputer architectures

Configurations	P655	BlueGene/L	P690	Seaborg	Bassi
Number of Nodes	272	1024	7	380	111
MCMs/Node	1	NA	4	NA	NA
chips/MCM	4	NA	4	NA	NA
Cores/chip	2	2	2	1	1
CPUs/Node	8	2	32	16	8
CPU type	1.5,1.7GHz POWER4	700 MHz PowerPC	1.7GHz POWER4	375MHz POWER3	1.9GHz POWER5
Memory/Node	16,32GB	1GB	128GB	16-64GB	32GB
L1 Cache/CPU	64/32 KB	32KB	64/32 KB	32/64 KB	64/32 KB
L2 Cache/chip	1.41MB	128-byte lines	1.41MB	8MB	1.92MB
L3 Cache/chip	32MB	4MB	32MB	NA	36MB
Network	Federation	3D Torus	Federation	Colony	Federation

2.1. Multi-Chip Module and Processor Affinity. In this section, we address processor affinity (i. e., how processes are dispatched to processors) and its policy in the following CMP clusters: P655, P690, and BlueGene/L. Although Bassi is POWER5 system, it is configured with only one active core per POWER5 chip. Seaborg is a POWER3 system with one processor per chip.

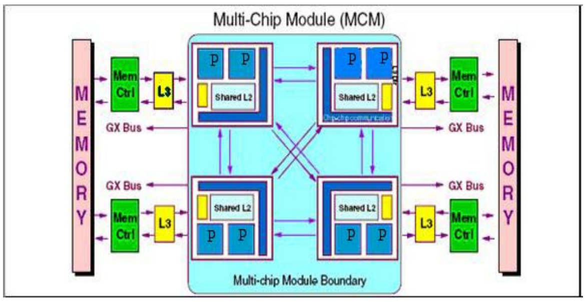


Fig. 2.1. *A logical view of a MCM [1]*

Fig. 2.1 shows the logical view of a POWER4 MCM. The MCM is used as an eight-way basic building block. Each MCM has four POWER4 chips; the two processors on the same chip share L2 and L3 caches. The logical interconnection of four POWER4 chips is point-to-point with uni-directional buses connecting each pair of chips to form an 8-way SMP with an all-to-all interconnection topology. P655 has one MCM per node. With the single MCM configuration, each chip always sends requests, commands and data on its own bus, but snoops all buses for requests or commands from other MCMs. Multiple MCMs can be interconnected to form larger SMPs, such as P690 with four MCMs, by extending each bus from each module to its neighboring module in one direction.

The processor affinity policy for POWER4 and POWER5 is discussed in [4]. When a virtual processor is dispatched, it is first dispatched onto the same physical processor that it last ran on. Otherwise, it will be dispatched onto the first available processor in the following order: on the same chip, then to another chip on the same MCM, then to a chip on the same node.

Fig. 2.2 presents the bi-directional MPI bandwidth comparison on P690 using processor binding (e.g., we use IBM AIX command *bindprocessor* to implement processor binding and use Intel's IMB benchmark Sendrecv to measure bi-directional bandwidth). We use the term **PPM** to denote processors per MCM and **PPC** to denote processors per chip. We measure the MPI bandwidth at the following levels: within a chip (**Same**), across two chips on one MCM (**Chips**), across two MCMs (**MCMs**) or across two nodes (**Nodes**) on P690. The results indicate that using two processors within a chip results in much better bandwidth than using one processor per chip (on the same MCM) or using one processor per MCM or per node for small or medium message sizes in the range of 1 byte to 256KB messages. It is interesting to note that the bandwidth for the message size of 256KB or larger on P690 is very similar for using two processors in the same chip or using two processors across different chips (but on the same MCM) or using two processors on the different MCMs of the same node; using two nodes still resulted in poor bandwidth for large message sizes.

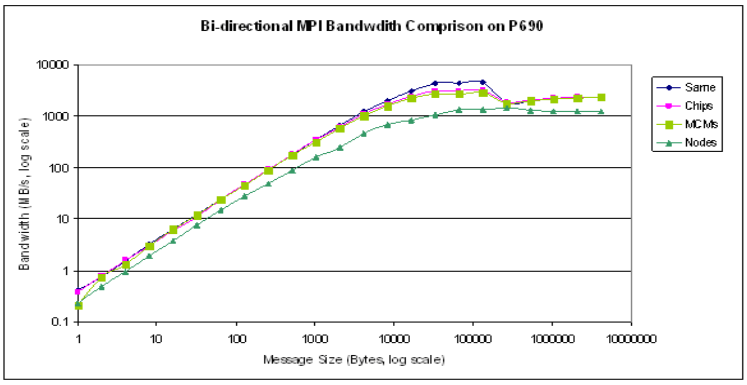


FIG. 2.2. *Bi-directional bandwidth comparison on P690 using processor binding*

3. Application Optimization Methods.

3.1. Loop Optimizations. Much of the computation involved in scientific applications such as LBM, GTC and GYRO occurs within nested loops. Therefore, loop optimization is fundamentally important for such applications. In this section, we discuss loop blocking, loop unrolling and loop fusion to optimize the scientific applications.

Loop blocking is a well-known loop optimization technique to aid in taking advantage of memory hierarchy; its main purpose is to eliminate as many cache misses as possible. This technique transforms the memory domain of an application into smaller chunks, such that computations are executed on the chucks that easily fit into cache to maximize data reuse. The optimal loop block size varies with different applications on different systems. In this paper, we apply the following loop block sizes: 2x2, 4x4, 8x8 and 16x16 to the three scientific applications to measure which loop block size is optimal. Our experimental results indicate that the optimal loop block size is 2x2 on BlueGene/L and 4x4 on P655 for GTC, and 16x16 on BlueGene/L and 4x4 on P655 for LBM and GYRO.

Loop unrolling is a well-known code transformation technique that replicates the original loop body multiple times, adjusts the loop termination code and eliminates redundant branch instructions. Outer loop unrolling can increase computational intensity and minimize load/stores, while inner loop unrolling can reduce data dependency and eliminate intermediate loads and stores. We combine inner and outer loop unrolling to optimize the scientific applications. For examples, we unroll the inner loops four times for four major double nested loops in GTC code so that we reconfigure the double nested loops into the single loops, then use compiler directives for further loop unrolling.

The performance for the hand-tuned scientific codes using loop blocking and unrolling can be further improved by using compiler directives, which are hardware-specific. The IBM XL Fortran compiler provides some hardware-specific directives for performance optimization, such as −qdirective=UNROLL_AND_FUSE [4]. The UNROLL_AND_FUSE directive instructs the compiler to allow loop unrolling and fusion where applicable. Loop fusion is also a code transformation technique. It minimizes the required number of loop iterations, and improves data locality by increasing data reuse in registers and cache. For each execution environment used in this study, we utilize the appropriate compiler directives to further optimize the code.

3.2. Hybrid Methods for MPI_Allreduce and MPI_Reduce. In this section, we present our hybrid methods to optimize MPI_Allreduce and MPI_Re-duce, which are common in GTC and GYRO. In scientific applications, especially GTC and GYRO, MPI_Allreduce dominates the most communication time. In order to optimize MPI_Allreduce, we incorporate the following hybrid method that we implemented:

- Use Intel's MPI benchmarks to measure the performance of the original MPI_Allreduce with different message sizes on each cluster,
- Measure the performance of Rabenseifner's algorithm for allreduce in [14] with different message sizes on the same cluster,
- Compare both performance to decide message size ranges for the best performance,
- Implement a hybrid method for MPI_Allreduce based on the message size ranges (basically using the original algorithm at small message sizes and the Rabenseifner's algorithm otherwise).

Table 4.1
Datasets with scaling the number of processors for GTC

Processors	2	4	8	16	32	64	128	256	512	1024	2048
micell	100	100	100	100	100	100	200	400	800	1600	3200
mecell	100	100	100	100	100	100	200	400	800	1600	3200
mzetamax	2	4	8	16	32	64	64	64	64	64	64
npartdom	1	1	1	1	1	1	2	4	8	16	32

Rabenseifner's algorithm performs a reduce-scatter followed by an allgather for MPI_Allreduce, and the algorithm also performs a reduce-scatter followed by a gather for MPI_Reduce. The hybrid method takes different systems and message sizes into account in order to optimize MPI_Allreduce. We also use the similar hybrid method to optimize MPI_Reduce. A MPI program is required to be recompiled with our own MPI library for the two subroutines Allreduce and Reduce.

4. Gyrokinetic Toroidal Code (GTC). The Gyrokinetic Toroidal code (GTC) [2] is a 3D particle-in-cell application developed at the Princeton Plasma Physics Laboratory to study turbulent transport in magnetic fusion. GTC is currently the flagship SciDAC fusion microturbulence code written in Frotran90, MPI and OpenMP. Fig. 4.1 shows a visualization of potential contours of microturbulence for a magnetically confined plasma using GTC. The finger-like perturbations (streamers) stretch along the weak field side of the poloidal plane as they follow the magnetic field lines around the torus [12]. Fig. 4.2 presents the basic steps in the GTC code.

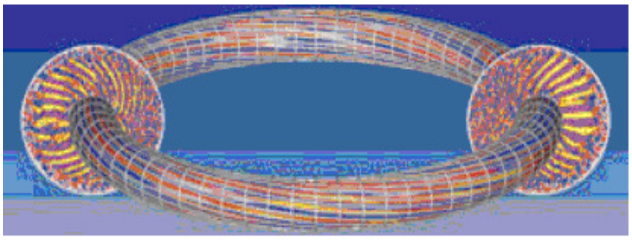
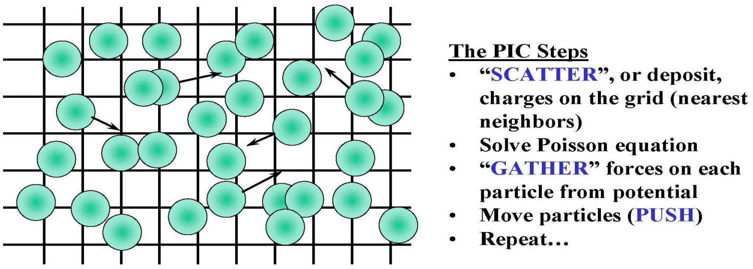


Fig. 4.1. *Potential contours of microturbulence for a magnetically confined plasma [12]*



The PIC Steps
- **"SCATTER", or deposit, charges on the grid (nearest neighbors)**
- **Solve Poisson equation**
- **"GATHER" forces on each particle from potential**
- **Move particles (PUSH)**
- **Repeat...**

Fig. 4.2. *Particles in cell (PIC) steps [2]*

The test case for GTC studied in this paper is 100 particles per cell and 100 time steps. The problem sizes for the GTC code are listed in Table 4.1, where micell is the number of ions per grid cell, mecell is the number of electrons per grid cell, mzetamax is the total number of toroidal grid points, and npartdom is the number of particle domain partitions per toroidal domain.

4.1. Application Performance Analysis. In this section, we investigate how different number of PPN impacts the application performance. Runtime (unit: seconds) indicates the total application execution time, which also includes I/O time. Table 4.2 provides the performance results from P655 for dispatching two processes to two processors on the same chip, two processors on different chips on the same MCM, and two processors across two nodes. The percentage given next to runtime corresponds to the increase in runtime comparing

Table 4.2
Performance comparisons of GTC on 2 processors on P655

	Across 2 nodes	Across 2 chips	Within a chip
Metrics	2x1/1PPC	1x2/1PPC	1x2/2PPC
Runtime (% difference)	984.26 (baseline)	990.02 (0.59%)	1114.32 (13.21%)
L1 hit rate	92.356%	92.375%	92.386%
TLB miss rate	0.005%	0.005%	0.005%
L2 bandwidth/processor	4618.906MB/s	4582.185MB/s	4071.585MB/s
% accesses from L2	2.228%	2.214%	2.235%

Table 4.3
Performance comparisons of GTC on 2 processors on P690

	Across 2 nodes	Across 2 MCMs	Across 2 chips	Within a chip
Metrics	2x1/1PPM-1PPC	1x2/1PPM-1PPC	1x2/2PPM-1PPC	1x2/2PPM-2PPC
Runtime (% difference)	980.23 (baseline)	994.58 (1.46%)	1001.49 (2.17%)	1009.66 (3.00%)
L1 hit rate	92.515%	92.48%	92.457%	92.475%
TLB miss rate	0.005%	0.005%	0.005%	0.005%
L2 bandwidth /processor	4578.505 MB/s	4499.834 MB/s	4491.888 MB/s	4457.119 MB/s
% accesses from L2	2.211%	2.181%	2.177%	2.16%

to the baseline configuration corresponding to using one processor per node (or the least amount of sharing of node resources). There is 13.21% increase in execution time when using two processors within the same chip (1x2/2PPC) due to contention resulting from the sharing of resources, such as L2 and L3 caches. For the case of using one processor per chip with two chips in the same MCM (1x2/1PPC), there is a very small increase in execution time (only 0.59%). Recall that each node on P655 has one MCM, which consists of four dual-core POWER4 chips. Table 4.2 also presents the hardware counters' performance using hpmcount [5]. The hardware counters indicate large difference (11.85%) in L2 bandwidth per processor for the two cases of using one processor per node versus using two processors within the same chip. Hence, there is significant contention for L2 when using two processors within the same chip.

Table 4.3 shows the performance results for dispatching two processes to two processors on the same chip, on two different chips on the same MCM, or two different chips on two different MCMs on P690 for two processors, respectively. Each node on P690 has four MCMs with four dual-core POWER4 chips per MCM shown in Table 2.1. There is only a 3.00% increase in execution time when using two processors within the same chip (1x2/2PPC) due to contention resulting from the sharing of resources, such as L2 and L3 caches. Comparing Table 4.3 with Table 4.2, it is clear that L2 bandwidth per processor accounts for the performance difference when using two processors within one chip (1x2/2PPC) versus using two processors across nodes (2x1/1PPM-1PPC).

Table 4.4 indicates the performance gaps resulting from using different PPN on a total of 64 processors on five supercomputers. As expected, the smallest times corresponds to the minimum amount of sharing of node resources. With increasing number of PPN, the application execution time increases. The performance data in Table 4.4 utilized default processor affinity described in Section 2.1. For example, with the 16x4 configuration on P655 (16 nodes with 4 PPN), the 4 processors per node were dispatched such that the only two chips were used, whereby both cores per chip were used. The time difference between the schemes 64x1 and 8x8 is 8.51% on P655. The time difference between the schemes 64x1 and 4x16 is 6.44% on Seaborg. The time difference between the schemes 32x2 and 8x8 is 2.23% on Bassi. The time difference between the schemes 4x16 and 2x32 is 7.18% on P690. Because the GTC code could not be executed on UNC RENCI BlueGene/L with 2 PPN (in VN mode) except 2048 processors, we could not collect any data for using 2 PPN on the machine. We find that the subroutines *pushi* and *chargei* in GTC take more than 90% of the application execution time on 64 processors, and are sensitive to different memory access patterns and communication patterns for different PPN.

Table 4.5 illustrates that impact of using processor binding to reduce contention of chip resources. Using 1 processor per chip (1PPC) results in approximately 3% decrease in execution time versus using 2 processors per chip (2PPC) for both 32x2 and 16x4 configurations. While the difference is small, it is the case that processor binding can aid task schedulers in maximizing the application performance in dedicated usage of CMP nodes.

Table 4.4
Execution times (seconds) of GTC for different PPN on 64 processors

System	64x1	32x2	16x4	8x8	4x16	2x32
Seaborg	3545.08	3549.41	3574.92	3617.47	3773.43	NA
P655	1203.32	1253.83	1266.49	1305.74	NA	NA
BlueGene/L	3937.46	–	NA	NA	NA	NA
Bassi	–	875.11	886.54	894.64	NA	NA
P690	NA	NA	NA	NA	1210.26	1297.12

Table 4.5
Performance comparisons of GTC on P655 using processor binding

32x2			16x4		
2PPC	1PPC	% difference	2PPC	1PPC	% difference
1253.83	1218.31	2.92	1266.49	1230.47	2.93

4.2. Application Performance Optimization. Much of the computation involved in GTC, especially the subroutines *pushi* and *chargei*, occurs within nested loops. Therefore, we use loop blocking, loop unrolling and loop fusion to optimize the application. We also use the hybrid method for MPI_Allreduce to optimize MPI_Allreduce. Further, MPI_Allreduce dominates the most communication time of the code on large number of processors. Therefore, we use the hybrid method described in Section 3 to optimize the MPI_Allreduce. The range of message sizes are defined on different systems as follows: for P655, the hybrid method uses the original algorithm for the message sizes of less than 1KB and the Rabenseifner's algorithm otherwise. For BlueGene/L, because GTC could only be executed on the number of processors with 1PPN, the hybrid method uses the Rabenseifner's algorithm for the message sizes of between 256 bytes and 2KB and the original algorithm otherwise.

Our experimental results indicate that the optimal loop block size for GTC is 2x2. This block size is used to optimize outer loops of the triple nested loops in GTC code. We also unroll the inner loops four times for four major double nested loops in the subroutines *pushi* and *chargei* such that the double nested loops are reconfigured into single loops. Further, we used the compiler options consistent with that given in [2].

The results of applying the aforementioned optimizations are given in Tables 4.6 and 4.7 for BlueGene/L and P655, respectively. We performed the optimization in series so that we could quantify the results of each optimization. These results are not given in this paper due to space. The results, however, indicated that majority of the optimization for GTC resulted from hand-tuned loop unrolling and blocking. We achieve up to 18.97% performance improvement on BlueGene/L. It should be noted that the performance gap for P655 using 64x1 versus 8x8 was 8.51% in Table 4.4. Using the aforementioned optimizations we were able to reduce the execution time for using all PPN by 4.99%.

5. LBM Application. The Lattice Boltzmann Method (LBM) [19] is widely used for simulating fluid dynamics; this method has the ability to deal efficiently with complex geometries and topologies. The LBM application is computation intensive. In our simulations, we use the D3Q19 lattice model (19 velocities in 3D) with the collision and streaming operations. The LBM application code is divided into six kernels, which are given below in Fig. 5.1.

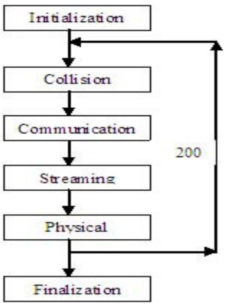


Fig. 5.1. *Program control flow with kernels for the LBM code*

Table 4.6
Execution time (seconds) comparison of GTC between the original and the optimized on BlueGene/L

Processors (MxN)	original	optimized	% improvement
8 (8x1)	3804.03	3082.54	18.97%
16 (16x1)	3834.98	3124.80	18.52%
32 (32x1)	3869.60	3166.56	18.17%
64 (64x1)	3937.46	3221.31	18.19%
128 (128x1)	3919.06	3202.81	18.28%
256 (256x1)	3913.07	3208.95	17.99%
512 (512x1)	3820.28	3120.65	18.31%
1024 (1024x1)	3788.49	3096.36	18.27%
2048 (1024x2)	3808.03	3108.40	18.37%

Table 4.7
Execution time (seconds) comparison of GTC between the original and the optimized on P655

Processors (MxN)	original	optimized	% improvement
8 (1x8)	1207.07	1144.73	5.16%
16 (2x8)	1242.56	1174.80	5.45%
32 (4x8)	1273.75	1203.32	5.53%
64 (8x8)	1305.73	1240.52	4.99%
128 (16x8)	1263.71	1201.99	4.88%
256 (32x8)	1237.22	1177.27	4.85%
512 (64x8)	1228.28	1172.25	4.56%

The description of each kernel is given below:

- *Initialization:* reads input files and sets up the initial parameter values.
- *Collision:* computes the effect of the collisions, which occur during the particle movement.
- *Communication:* is to communicate the needed data among neighboring blocks.
- *Streaming:* moves particles in motion to new locations along their respective velocities.
- *Physical:* calculates macroscopic variables such as fluid density, which are used in the collision and streaming steps.
- *Finalization:* cleans up the program after everything is done, and outputs the results.

The kernels *Collision, Communication, Streaming*, and *Physical* stay in a loop with the number of iterations of 200. The kernels *Initialization* and *Finalization* are executed once. The problem size for the LBM application is a 3D mesh computational domain. In this paper, we use two problem sizes of 128x128x128 and 512x512x512 for the LBM application in our experiments.

5.1. Application Performance Analysis. Table 5.1 provides the execution times of the LBM application with problem sizes of 128x128x128 and 512x512x512 for different PPN on a total of 32 processors across the five supercomputers. The performance data was collected using the default processor affinity described in Section 2.1. With increasing number of PPN, the application execution time increases significantly on Seaborg, P655, and P690; there is very little difference in execution times on Bassi. For example, the time difference between 1 PPN and 16 PPN on Seaborg is 21.3% for the problem size of 512x512x512, and 15.83% for the problem size of 128x128x128. The time difference between the scheme 1 PPN and 8 PPN on P655 is 20.73% for the problem size of 512x512x512, and 10.03% for the problem size of 128x128x128. The time difference between the scheme 8 PPN and 32 PPN on P690 is 17.79% for the problem size of 512x512x512, and 23.53% for the problem size of 128x128x128.

Table 5.2 indicates the results from using processor such that only one core per chip is used thereby minimizing resource sharing. The results indicate a 4.71% decrease in execution time for the 16x2 configuration and a 6.75% reduction in execution time for the 8x4 configuration. This indicates that processor binding can aid task schedulers in maximizing the application performance in dedicated usage of CMP nodes.

Table 5.3 provides the following details about the performance characteristics for LBM with problem size of 512x512x512: the L1 hit rate, L2 bandwidth (per processor), and percentage accesses from L2. Because BlueGene/L does not support hpmcount, we did not collect its hardware level performance. With increasing PPN, the L1 hit rate varies very little across all systems because of equal workload per processor. On P655, the L2 bandwidth and percentage accesses from L2 decrease with an increase of PPN; this is especially the case

Table 5.1
Execution times (seconds) of LBM for different PPN on 32 processors

System Name	Problem size	32x1	16x2	8x4	4x8	2x16	1x32
Seaborg	128x128x128	94.21	94.42	95.29	98.72	109.12	NA
	512x512x512	6247.87	6307.41	6418.77	6722.31	7578.84	NA
P655	128x128x128	33.09	33.38	34.57	36.41	NA	NA
	512x512x512	2154.94	2224.76	2361.67	2601.66	NA	NA
P690	128x128x128	NA	NA	NA	29.15	30.35	36.01
	512x512x512	NA	NA	NA	2198.90	2248.62	2590.18
Bassi	128x128x128	21.43	21.92	21.14	21.78	NA	NA
	512x512x512	1379.63	1382.30	1380.32	1384.84	NA	NA
BlueGene/L	128x128x128	33.40	33.42	NA	NA	NA	NA
	512x512x512	2252.99	2253.27	NA	NA	NA	NA

Table 5.2
Performance comparisons of LBM on P655 using processor binding

16x2			8x4		
2PPC	1PPC	% difference	2PPC	1PPC	% difference
2224.76	2124.66	4.71	2361.67	2212.28	6.75

for the L2 bandwidth. For example, the L2 bandwidth for the 32x1 scheme is more than 11% larger than that for the 4x8 scheme. On Bassi, there is very small difference in percentage accesses from L2 across different PPN due to 20GB large page size, which uses hardware prefetch mechanisms to eliminate costly TLB misses. Further, Bassi has the highest memory bandwidth and MPI bandwidth, which result in very little change in communication rate for the different configurations. Hence, the execution time is flat for the different PPN on Bassi. It is interesting that Bassi has the lowest L1 hit rate and correspondingly, the highest percentage accesses from L2.

5.2. Application Performance Characteristics and Optimization. In this section, we use the LBM application with the problem sizes of 128x128x128 and 512x512x512 to utilize the information learned from the previous section to determine how to optimize the application.

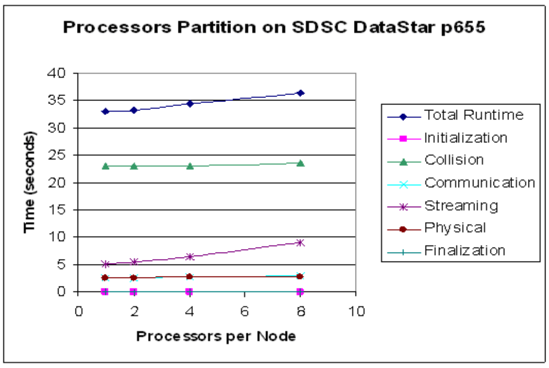


Fig. 5.2. *Execution times of LBM for different PPN on 32 processors on P655*

Figs. 5.2, 5.3 and 5.4 indicate the similar performance trend for LBM with the problem size of 128x128x128 using different PPN on 32 processors on P655, Seaborg, and P690, that is, the total application execution time increases as the number of PPN increases, and the performance of the kernel *Streaming* follows the pattern of the total application execution time on P655, Seaborg, and P690. It means that the kernel *Streaming* is the main cause of the increase in the total execution time. One major reason for this increase is the memory bottleneck, which results in that large number of processors on each node have to compete for a shared memory while the application requires enough memory for the dataset. It is interesting to see that the performance for the kernel *Communication* does not change much with increasing the number of PPN, and the performance for the dominated kernel *Collision* remains flat. The similar performance trend also occurs for the problem size of 512x512x512.

Table 5.3
Memory performance for LBM with the size of 512x512x512

System	Metrics	32x1	16x2	8x4	4x8	2x16
P655	L1 hit rate	98.13%	98.15%	97.99%	97.69%	NA
	L2 bandwidth (MB/s)	1127.01	1098.07	1082.61	1015.14	
	% accesses from L2	0.618%	0.602%	0.593%	0.541%	
Bassi	L1 hit rate	94.93%	92.82%	95.33%	95.85%	NA
	L2 bandwidth (MB/s)	3.035	4.788	2.523	2.145	
	% accesses from L2	5.069%	7.177%	4.669%	4.147%	
Seaborg	L1 hit rate	99.14%	99.00%	98.93%	98.88%	99.14%
	L2 bandwidth (MB/s)	0.069	0.066	0.064	0.060	0.050
	% accesses from L2	0.332%	0.380%	0.409%	0.420%	0.340%

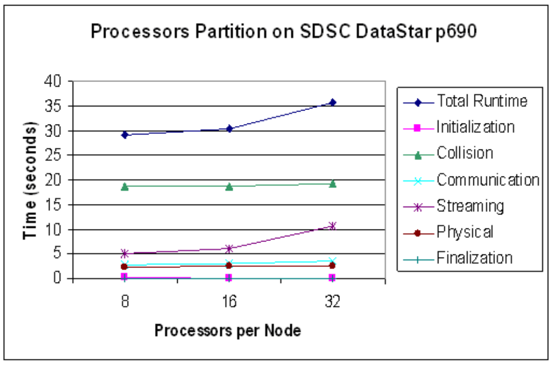


Fig. 5.3. *Execution times of LBM for different PPN on 32 processors on P690*

Figs. 5.2, 5.3 and 5.4 also indicate how the LBM code and its kernels are sensitive to different memory access patterns and communication patterns on the CMP clusters. This kind of performance characteristics of the LBM code could not be found in most application performance analysis just based on a single run on a given number of processors. The kernel *collision* dominates most of the application execution time, but the performance trend for the kernel *Streaming* is similar to that for the total runtime on P655. Optimizing the kernel *collision* does not eliminate the performance impact using different number of PPN. Hence, we focus on optimizing *Streaming*, which entails moving particles in motion to new locations along their respective 19 velocities [19]. This kernel requires a significant number of memory copy operations, which causes memory congestion when large numbers of PPN are used. In particular, *Streaming* consists of five triple-nested loops. We focus on using loop blocking to optimize the outer two loops of each triple-nested loop. For the sake of simplicity, we just present results for the large problem size of 512x512x512.

Table 5.4
Execution time (seconds) comparison of LBM between the original and the optimized on P655

Processors (MxN)	original	optimized	% improvement	Optimal block size
32 (4x8)	2601.66	2191.28	15.77%	4x4
64 (8x8)	1306.55	1110.71	14.99%	4x4
128 (16x8)	653.31	561.56	14.04%	4x4
256 (32x8)	319.71	287.71	10.01%	4x4
512 (64x8)	163.44	157.53	3.62%	16x16
1024 (128x8)	83.9	79.37	5.40%	16x16
2048 (256x8)	48.56	45.3	6.71%	16x16

Table 5.4 provides the performance comparison between the original code and the optimized for the problem size of 512x512x512 on P655 using all PPN. The optimized code with the block size of 4x4 achieved the best performance on 32, 64, 128 and 256 processors, but the optimized code with the block size of 16x16 obtained the best performance on 512, 1024, and 2048 processors. For only optimizing the kernel *Streaming* in the original application code, the percentage of performance improvement is up to 15.77%. Compare to the results for different PPN on P655 in Table 5.1, where there is 20.73% time difference between the scheme 32x1 and the

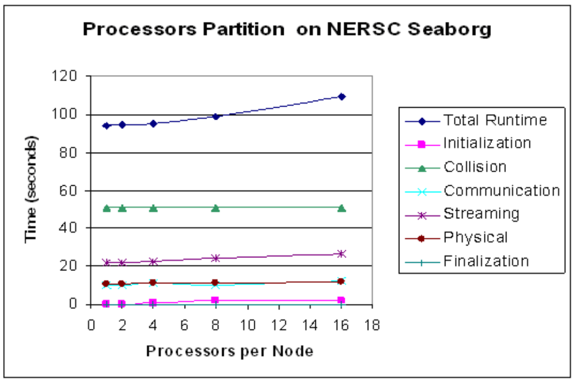


FIG. 5.4. *Execution times of LBM for different PPN on 32 processors on Seaborg*

4x8, our optimization reduces execution time by 15.77%. This is a big improvement. Note that the performance improvement percentage trend decreases with increasing the number of total processors for the optimal block size of 4x4 because of the decrease of workload per processor and the increase of the communication percentage with increasing the number of processors. For the problem size of 512x512x512, the LBM application can only be executed on 32 processors or more because of the large sufficient memory requirement.

TABLE 5.5
Execution time (seconds) comparison of LBM between the original and the optimized on BlueGene/L

Processors(MxN)	original	optimized	% improvement	Optimal Block size
32 (16x2)	2253.27	1954.95	13.24%	16x16
64 (32x2)	1280.72	1092.11	14.73%	16x16
128 (64x2)	699.71	606.83	13.27%	16x16
256 (128x2)	356.09	311.35	12.56%	16x16
512 (256x2)	153.95	150.73	2.09%	2x2

Table 5.5 provides the performance comparison between the original code and the optimized for the problem size of 512x512x512 on RENCI BlueGene/L using all PPN. The optimized code with the block size of 16x16 achieved the best performance on 32, 64, 128 and 256 processors, but the optimized code with the block size of 2x2 obtained the best performance on 512 processors. For only optimizing the kernel *Streaming* in the original application code, the percentage of performance improvement is up to 14.73%. Compare to the results for different PPN on BlueGene/L in Table 5.1, this is a big improvement.

Hence, for the same problem size and same total number of processors, the loop blocking optimization resulted in much larger performance improvement of the LBM application when using all the processors per node versus one processor per node.

6. GYRO Application.

GYRO [3] is an advanced Eulerian gyrokinetic-Maxwell equation solver that is capable of facilitating a better understanding of plasma microinstabilities and turbulence flow in the tokamak geometry. GYRO is used widely in simulating microturbulent transport in core plasma. In this section, we use GYRO with two benchmark datasets: the Waltz Standard benchmark, B1-std, consisting of grids of size 6 x 140 x 4 x 4 x 8 x 6, and the Cyclone Base Benchmark, B2-cy, consisting of grids of size 6 x 128 x 4x 4 x 8 x 6.

6.1. Application Performance Analysis.

Table 6.1 provides the execution times of the GYRO application with B1-std and B2-cy for different PPN on a total of 32 processors. The performance data was collected using the default processor affinity described in Section 2.1. With increasing number of PPN, the application execution time increases significantly on Seaborg, P655, P690, and Bassi. For example, for the problem size of B1-std, there is a 55.79% increase in the execution time with increasing the PPN from 8 to 32 on P690, 42% increase in the execution time with increasing the PPN from 1 to 8 on P655, 9.15% increase with increasing the PPN from 1 to 8 on Bassi, and 4.86% increase with increasing the PPN from 1 to 16 on Seaborg.

In GYRO, the subroutine get_RHS, which calculates the periodic and non-periodic boundary conditions, requires extensive global communication through the use of MPI_Alltoall and MPI_Allreduce operations. The

Table 6.1
Execution times (seconds) of GYRO for schemes on 32 processors

System Name	Problem size	32x1	16x2	8x4	4x8	2x16	1x32
Seaborg	B1-std	9177.38	9208.42	9280.395	9322.49	9623.39	NA
	B2-cy	21108.59	21132.95	21245.59	21441.89	21849.849	NA
P655	B1-std	390.14	449.75	484.13	554.24	NA	NA
	B2-cy	910.09	972.17	1046.47	1158.01	NA	NA
BlueGene/L	B1-std	1378.41	1379.20	NA	NA	NA	NA
	B2-cy	2983.37	2988.42	NA	NA	NA	NA
Bassi	B1-std	310.62	318.04	325.67	341.917	NA	NA
	B2-cy	697.79	714.17	724.49	765.25	NA	NA
P690	B1-std	NA	NA	NA	443.64	501.01	691.15
	B2-cy	NA	NA	NA	987.61	1146.11	1489.69

MPI_Alltoall and MPI_Allreduce subroutines account for more than 95% of the total communication across all platforms. It is noted that the performance of these routines are highly inefficient for using all PPN on P655, and the communication rate increases from 8.52% to 11.33% with increasing the PPN. We find that *kernel 3* which computes the RHS of the electron and ion GKEs for both periodic and nonperiodic boundary conditions, and *kernel 5* which computes the newly needed RHS in GYRO, dominate the application execution time, and are sensitive to memory access patterns and communication patterns for different processors per node.

Table 6.2
Performance comparisons of GYRO on P655 using processor binding

16x2			8x4		
2PPC	1PPC	% difference	2PPC	1PPC	% difference
449.75	423.36	5.89	484.13	449.48	7.16

Table 6.2 indicates that using processor binding can achieve the better results for the 16x2 and 8x4 schemes. Using one processor per chips results in a 5.89% decrease in execution time for the 16x2 configuration and a 7.16% decrease in execution for the 8x4 configuration. This indicates that processor binding can aid task schedulers in maximizing the application performance in dedicated usage of CMP nodes.

6.2. Application Performance Optimization. In this section, we use the GYRO application with the problem size of B1-std to utilize the information learned from the previous performance analysis to determine how to optimize the application. It is noted that similar results occur for the problem size of B2-std.

Given the results of the application characterization and the kernel performance, we focus on optimizing the MPI_Allreduce implementation to take advantage of the CMP clusters. In particular, we consider the communication architectures and message sizes to develop an adaptive communication pattern to optimize the MPI_Allreduce. Such improvements impact *kernels 3* and 5. The results shown in Table 6.3 illustrate the improvements that have been achieved utilizing a hybrid implementation of the MPI_Allreduce routine for different message size ranges. The message size ranges for MPI_Allreduce are defined on P655 as follows: the hybrid method uses the original algorithm for the message sizes of less than 1KB and the Rabenseifner's algorithm otherwise. This optimization also consists of loop blocking the outer nested loops of the *get_RHS* subroutine used in *kernel 3* and *kernel 5* with the optimal block size of 4x4. The compiler optimization utilizing the UNROLL_AND_FUSE option is applied to the entire code.

It is noted that the performance improvements in Table 6.3 are representative of an optimization on a small segment of the GYRO code, with respect to the get_RHS subroutine. The results indicate performance improvement up to 9.33% on P655 and up to 12.29% on BlueGene/L. It is noted that, however, from Table 6.1 a performance difference of 42% on B1-std when increasing the PPN from 1 to 8 on P655 using a total of 32 processors. In Table 6.3, however, the performance improvement is 7.81%. The performance improvement is very good when we consider that only part of the GYRO code, *kernels 3 and 5*, were optimized. Further, only about one third of each of these two kernels was optimized. The improvement illustrates proof of concept. To achieve close to the 42% performance gap, we need to optimize the entire GYRO code.

Performance improvements on BlueGene/L are a result of using a hybrid MPI_All-reduce scheme, loop blocking (with the optimal block size of 16x16) to *kernel 3* and *kernel 5*, and a compiler optimization using the UNROLL_AND_FUSE option shown in Table 6.4. Note that the MPI_Alltoall routine did not significantly impact the performance on the BlueGene/L system. The message size ranges for MPI_Allreduce are defined on BlueGene/L: the hybrid method uses the original algorithm for the message sizes of less than 256 bytes and the Rabenseifner's algorithm otherwise.

Note that the performance improvement percentage trend decreases with increasing the number of processors in Tables 6.3 and 6.4 because of the decrease of workload per processor and the increase of the communication percentage with increasing the number of processors and the fixed problem size.

Table 6.3
Execution time (seconds) comparison of GYRO between the original and the optimized on P655

Processors (MxN)	original	optimized	% improvement
16 (2x8)	1130.2923	1024.7543	9.33%
32 (4x8)	554.24	510.93	7.81%
64 (8x8)	276.78	260.58	6.21%
128 (16x8)	146.91	136.79	6.88%
256 (32x8)	84.81	81.05	4.43%
512 (64x8)	61.46	58.939	4.10%

Table 6.4
Execution time (seconds) comparison of GYRO between the original and the optimized on BlueGene/L

Processors (MxN)	original	optimized	% improvement
16 (8x2)	2746.85	2409.21	12.29%
32 (16x2)	1379.20	1210.66	12.22%
64 (32x2)	761.45	697.09	8.45%
128 (64x2)	410.70	372.69	9.25%
256 (128x2)	235.09	214.00	8.97%
512 (256x2)	161.02	150.01	6.83%
1024 (512x2)	121.38	114.37	5.77%

7. Conclusions. This paper used three large-scale, scientific applications, GTC, LBM and GYRO , to analyze the performance impact of the sharing of node resources on five supercomputers, P655, P690, BlueGene/L, Seaborg and Bassi. The experimental results indicated that there can be a significant difference in execution time when using different numbers of processors per node and using different numbers of cores per chip, chips per MCM, and different numbers of MCMs. Memory bandwidth contention, especially L2 cache, is the primary source of performance degradation. Using the loop optimization techniques which aids in taking advantage of advanced memory hierarchy and the hybrid communication optimization methods, the performance can be improved by up to 18.97% for GTC, up to 15.77% for LBM, and up to 12.29% for GYRO on up to 2048 processors when utilizing all processors per node. Hence, understanding the application and system characteristics can lead to optimization that aids in the efficient use of all processors per node. Future work is focused on further optimizing the entire GYRO code by considering how its kernels interact with each other using kernel coupling techniques [15, 18].

Acknowledgements. The authors would like to acknowledge the SDSC for the use of DataStar p655 and p690, the DOE NERSC for the use of the Seaborg and Bassi, and Renaissance Computing Institute for the use of BlueGene/L. We would also like to thank Stephane Ethier from Princeton Plasma Physics Laboratory and Shirley Moore from University of Tennessee for providing the GTC code and datasets, Patrick Worley from Oak Ridge National Laboratory for providing the GYRO code, and Dazhi Yu and Jacques Richard from Department of Aerospace Engineering at Texas A&M University for providing the LBM code.

REFERENCES

[1] S. Behling, R. Bell, et al., *The POWER4 Processor Introduction and Tuning Guide*, IBM Redbooks, Nov. 2001.

[2] S. Ethier, First Experience on BlueGene/L, *BlueGene Applications Workshop*, ANL, April 27-28, 2005. http://www.bgl.mcs.anl.gov/Papers/GTC{_}BGL{_}20050520.pdf

[3] M. Fahey, and J. Candy, GYRO: A 5-d gyrokinetic-maxwell solver, *SC04*, 2004.

[4] B. Gibbs, B. Atyam, et al., *Advanced POWER Virtualization on IBM@server p5 Servers: Architecture and Performance Considerations*, IBM Redbooks, Nov. 2005.

[5] hpmcount, `http://www.nersc.gov/nusers/resources/software/ibm/hpmcount/`

[6] Intel MPI Benchmarks, Users Guide and Methodolgy Description (Version 2.3), `http://www.intel.com/cd/software/products/asmo-na/eng/cluster/mpi/219848.htm`

[7] ipm, `http://www.nersc.gov/nusers/resources/software/tools/ipm.php`

[8] NERSC Seaborg and Bassi, `http://www.nersc.gov/nusers/resources/`

[9] F. Petrini, D. J. Kerbyson, and S. Pakin, The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q, *SC03*, 2003.

[10] J. Phillips, G. Zheng, S. Kumar, and L. Kale, NAMD: Biomolecular Simulation on Thousands of Processors, *SC02*, 2002.

[11] UNC RENCI BlueGene/L, `http://www.renci.org/about/computing.php`

[12] Scientific Discovery, A progress report on the US DOE SciDAC program, 2006.

[13] SDSC DataStar, `http://www.sdsc.edu/user_services/datastar/`

[14] R. Thakur, R. Rabenseifner, and W. Gropp, Optimization of Collective Communication Operations in MPICH, *The International Journal of High Performance Computing Applications*, Vol. 19, No. 1 (2005).

[15] Valerie Taylor, Xingfu Wu, Jonathan Geisler, and Rick Stevens, Using Kernel Couplings to Predict Parallel Application Performance, *the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC2002)*, July, 2002.

[16] Valerie Taylor, Xingfu Wu, and Rick Stevens, Prophesy: An Infrastructure for Performance Analysis and Modeling System of Parallel and Grid Applications, *ACM SIGMETRICS Performance Evaluation Review*, Volume 30, Issue 4, March 2003.

[17] Xingfu Wu, Valerie Taylor and Rick Stevens, Design and Implementation of Prophesy Automatic Instrumentation and Data Entry System, *the 13th International Conference on Parallel and Distributed Computing and Systems (PDCS2001)*, August, 2001.

[18] Xingfu Wu, Valerie Taylor, Jonathan Geisler, and Rick Stevens, Isocoupling: Reusing Coupling Values to Predict Parallel Application Performance, *the 18th International Parallel and Distributed Processing Symposium (IPDPS2004)*, April, 2004.

[19] Xingfu Wu, Valerie Taylor, Shane Garrick, Dazhi Yu, and Jacques Richard, Performance Analysis, Modeling and Prediction of a Parallel Multiblock Lattice Boltzmann Application Using Prophesy System, *IEEE International Conference on Cluster Computing*, Sep. 2006.

[20] Xingfu Wu and Valerie Taylor, Processor Partitioning: An Experimental Performance Analysis of Parallel Applications on SMP Cluster Systems, *the 19th International Conference on Parallel and Distributed Computing and Systems (PDCS2007)*, Nov. 2007.

Edited by: Fatos Xhafa, Leonard Barolli
Received: September 30, 2008
Accepted: December 15, 2008

SCALABLE COMPUTING: PRACTICE AND EXPERIENCE
Volume 10, Number 1, pp. 75–86. http://www.scpe.org

ISSN 1895-1767
© 2009 SCPE

MINIMIZATION OF DOWNLOAD TIME VARIANCE IN A DISTRIBUTED VOD SYSTEM*

ANNE-ELISABETH BAERT†, VINCENT BOUDET†, ALAIN JEAN-MARIE†, AND XAVIER ROCHE†

Abstract. In this paper, we examine the problem of minimizing the variance of the download time in a particular Video on Demand System. This VOD system is based on a Grid Delivery Network which is a hybrid architecture based on P2P and Grid Computing concepts. In this system, videos are divided into blocks and replicated on hosts to decrease the average response time. The purpose of the paper is to study the impact of the block allocation scheme on the variance of the download time. We formulate this as an optimization problem, and show that this problem can be reduced to finding a Steiner System. We analyze different heuristics to solve it in practice, and validate through simulation that a random allocation is quasi-optimal.

Key words: performance evaluation, video on demand, approximation algorithms, constraint optimization problem, simulation.

1. Introduction and Problems.

This paper is about the replication of data in a particular Grid Delivery Network (GDN). A GDN is a distributed data delivery system that is able to provide video services, among which Video On Demand (VOD). The idea of VOD is to allow users to request video documents at any time, without a preestablished time schedule. One of the main challenge of GDN system resides therefore in ensuring that users can download video at any time with guaranteed, pre-established download time.

The recent literature has pointed out the interest of using distributed systems for massive video distribution [9, 12, 13]. Documents replication and repartition can also solve some performance problems for video distribution, and is an effective way to improve the performance of video download time. Those problems have been studied extensively in the literature [2, 13, 10]. Ibaraki *et al.* employed a replication optimization technique borrowed from the theory of resource allocation problems [5]. Chou *et al.* gave a comparison of scalability and reliability characteristics of servers by the use of stripping and replication [2].

Venkatasubramanian *et. al.* proposed a family of heuristic algorithms for dynamic load balancing in a distributed VOD server [10]. Zhou *et al.* formulate the problem of video replication and placement on a distributed storage cluster as a combinatorial optimization problem [13].

For a popular document, increasing its replication may decrease its download time. However, as the total storage of the system is usually fixed, increasing its replication implies to decrease the replication of other documents. Video replication and placement algorithms based on popularity were therefore developed in [4, 9, 12]. In these algorithms, the most popular videos are all duplicated on the servers, the problem is then to find how much the less popular documents should be replicated. The response waiting time problem is then restricted to certain videos. In [8], the authors analyzed the impact of the division of the documents into blocks on availability and average download time in a general, BitTorrent-like P2P file systems.

As this brief review of the literature shows, there already exist many replication and placement algorithms aimed at optimizing the average download time for videos. However: on the one hand, none of these models applies directly to the GDN architecture, and on the other hand, no attention has been given yet to optimizing the *variance* (or the standard deviation) of download times. Yet, the system can foresee statistically and *guarantee* the download time only if this variance is small. This is the question we address in the present paper. We have shown in a previous work [1] that with an appropriate replication factor, the average download response time is independent of the allocation methods. On the other hand, we showed that the allocation methods have a direct impact on the variance of response waiting time.

The remainder of this paper is structured as follows. In Section 2, we briefly describe the architecture of the system, we also summarize some of the most important differences between GDN and traditional P2P approaches for VOD. In Section 3, we formulate the problem of minimizing the variance of the waiting time as a combinatorial optimization problem called MINVAR. We study this objective function and show the difficulty of this problem. We compute a lower bound on the optimal value for this objective function. In Section 4 we analyze some heuristics to solve the MINVAR problem. We evaluate them in Section 5 our algorithms using extensive simulations representing different situations. Section 6 concludes the paper.

*This research is supported the French National Agency for the Research, Audiovisual and Multimedia program, project VOODDO (contract 2007 AM 012-02).

†LIRMM, CNRS, Université de Montpellier 2, 161 rue ADA, 34392 MONTPELLIER Cedex 5, FRANCE. A. Jean-Marie is with INRIA Méditerannée. ({baert, boudet, ajm, roche}@lirmm.fr)

2. Grid Delivery Network description. The system under study is a hybrid architecture based on ideas from P2P Networking and "Grid Computing" called the *Grid Delivery Network* (GDN). This concept uses the principle of Grid Computing to keep the control of this centralized network. However, to optimize the data transmission and to propose a document delivery network within guaranteed time, this network uses the transport capacity of Peer to Peer networks, through its capacity of multi-source data delivery.

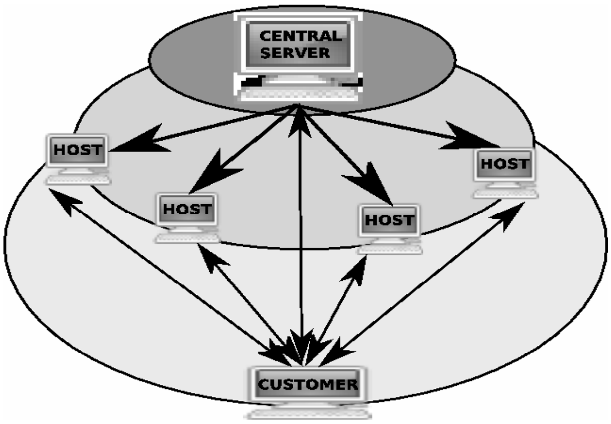


FIG. 2.1. *Grid Delivery network's architecture*

The particular GDN system analyzed in this paper is an implementation described in [11] (see also [1]) and represented in Figure 2.1. It is composed of a central server which manages the whole network and hosts which are "partners" of the GDN and deliver the documents to customers. In this GDN, each host is assumed to be a personal computer of the Grid Computing network, which is bound by contract to provide some resources to the system. Nevertheless, hosts are not available all the time to the network; the proportion of time each host is online is denoted with δ.

Documents are divided into pieces called "blocks", which are distributed *and* replicated over the partners, in order to improve reliability and decrease response time see Figure 2.2.

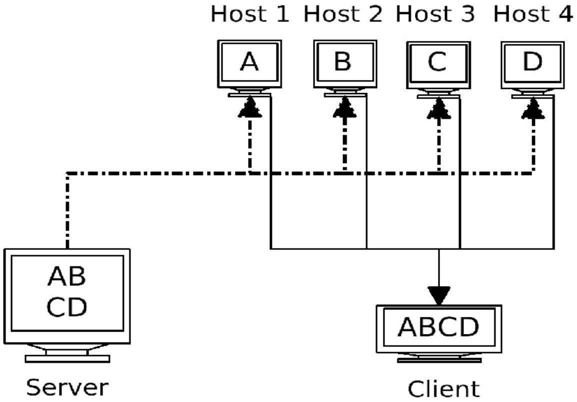


FIG. 2.2. *Document Delivery in the GDN*

The operation of the system is as follows. When some client wants to download a document on the GDN, it first contacts the central server. The server sends a *download plan* which specifies, for each block of the document, the identity of some host on which the block is stored. The client then executes this plan, by downloading the blocks in parallel from the partners which are online. Then, all missing blocks are downloaded directly from the central server (or some backup data storage).

In [1], we have proposed a probabilistic model of this GDN, and we have shown that, under appropriate assumptions, the download time for a whole document is given by

$$R = a - bN , \tag{2.1}$$

where N is the number of blocks of the document that happen to be available on the network, and a, b are positive constants.

By replicating a block on several hosts, the GDN increases the probability that some available host owns this block, and therefore that the download time will be smaller. However, as the total hosts capacity of the GDN is limited it is not possible to replicate every document on every host. We have studied in [1] the problem of minimizing the *average* response time, by picking the replication factor of each document appropriately, in function of their popularity, and under memory constraints. We have proved that the *average* download time is independent of the particular way the replicated blocks are assigned to distinct partners. On the other hand, it turns out that this assignment has a very strong impact of the *variance* of the download time. Since the Quality of Service objective of the GDN is to guarantee the download time to customers, it is important to find assignments which minimize this variance.

3. Response waiting time optimization.

3.1. Variance of the response waiting time. We begin with computing the variance of the download time of a particular document. According to Eq. (2.1), the variance of R is proportional to the variance of N, on which we concentrate in the remainder.

Let us consider a particular document D, segmented into T blocks. Let $B_1, \cdots, B_T$ be these blocks. Assume that the optimal replication factor L has been computed so as to optimize the average download time [1]. Define $U_j = 1$ if the block B_j is available on some online partner, 0 otherwise. Assuming that the hosts' on-time is uniformly distributed over time, and that hosts are independent, we have:

$$\begin{aligned}\mathbb{P}(U_j = 1) &= 1 - \prod_{i=1}^{L} (1 - \delta) \\ &= 1 - (1 - \delta)^L \ .\end{aligned}$$

Let N be the number of available blocks of the document D. We have $N = \sum_{j=1}^{T} U_j$. The variance of N is given by:

$$\mathrm{Var}(N) = \sum_{j=1}^{T} \mathrm{Var}(U_j) + \sum_{k \neq j} \mathrm{Cov}(U_k, U_j)\,. \tag{3.1}$$

We denote by $L(j)$ the list of hosts which own the block B_j. It is known that, using the notation $\bar{\delta} = 1 - \delta$,

$$\begin{aligned}\mathrm{Cov}(U_k, U_j) &= \mathbb{E}(U_k U_j) - \mathbb{E}(U_k)\mathbb{E}(U_j) \\ &= \bar{\delta}^{|L(k) \cup L(j)|} - \bar{\delta}^{|L(k)|+|L(j)|}\,.\end{aligned} \tag{3.2}$$

The variance of U_j is given by :

$$\begin{aligned}\mathrm{Var}(U_j) &= \mathbb{E}({U_j}^2) - \mathbb{E}(U_j)^2 \\ &= \bar{\delta}^{|L(j)|} - \bar{\delta}^{2|L(j)|}\,.\end{aligned} \tag{3.3}$$

We have :

$$|L(k) \cup L(j)| = |L(j)| + |L(k)| - |L(k) \cap L(j)|\,.$$

We use the assumption that the replication factor is the same for each block i. e. $|L(j)| = L\,, \forall j$, we obtain for Eq. (3.1):

$$\mathrm{Var}(N) = T\gamma^{-L}(1 - \gamma^{-L}) + \gamma^{-2L} \sum_{k \neq j} \left(\gamma^{|L(k) \cap L(j)|} - 1 \right)$$

where $\gamma = (1 - \delta)^{-1}$.

Accordingly, minimizing Eq. (3.1) is equivalent to minimizing the following objective function

$$\sum_{k \neq j} \gamma^{|L(k) \cap L(j)|}\,.$$

3.2. The MINVAR Problem. In this part, we define the MINVAR problem in the language of combinatorial optimization.

DEFINITION 3.1. *Given a bipartite graph $\mathcal{G} = (\mathcal{T}, \mathcal{P}, \mathcal{E})$, where $\mathcal{T}$, $\mathcal{P}$ are a set of vertices and $\mathcal{E}$ the set of edges, we define the interference function as*

$$J(\mathcal{G}, \gamma) = \sum_{b \neq b' \in \mathcal{T}} \gamma^{|L(b) \cap L(b')|} , \tag{3.4}$$

where $L(b)$ denotes the neighborhood of b. Obviously, the problem does not depend on the nature of the sets $\mathcal{T}$ and $\mathcal{P}$ but only on their size.

Our problem is the following:

DEFINITION 3.2. *The $MINVAR(T, P, L, \gamma)$ optimization problem consists in finding one bipartite graph $\mathcal{G} = (\mathcal{T}, \mathcal{P}, \mathcal{E})$ with $|\mathcal{T}| = T$ and $|\mathcal{P}| = P$, which minimizes the interference function $J(\mathcal{G}, \gamma)$ under constraints*

$$\forall b \in \mathcal{T} \ , |L(b)| = L \ . \tag{3.5}$$

3.3. Complexity of MINVAR. Now, we turn on the complexity of the MINVAR problem.

DEFINITION 3.3 ([3, 7]). *A Steiner system $S(B, L, P)$ is a collection of L-subsets (also called blocks) of a P-set such that each B-tuple of elements of this P-set is contained in a unique block.*

We have the following:

THEOREM 3.4. *There is a Steiner system $S(B, L, P)$ iff there is a solution $\mathcal{G}$ of the $MINVAR(T, P, L, \gamma)$ problem with*

$$T = \frac{P!(L-B)!}{L!(P-B)!} \text{ and } \gamma = T(T-1) + 1 \tag{3.6}$$

that satisfies $J(\mathcal{G}, \gamma) < \gamma^B$.

Proof.

If there is a collection C solution of the Steiner system $S(B, L, P)$, then we can assign to each vertex $b \in T$ the neighborhood that matches one of the subsets of C. There is exactly

$$\frac{P!(L-B)!}{L!(P-B)!}$$

subsets in C. So each pair $(b, b') \in T^2$, $b \neq b'$ shares at most $(B-1)$ neighbors. We have created a solution $\mathcal{G}$ such as

$$\begin{aligned} J(\mathcal{G}, \gamma) &\leq T(T-1)\gamma^{B-1} \\ &< \gamma^B . \end{aligned}$$

On the contrary, if there is a solution $\mathcal{G}$ of the MINVAR problem, i. e.,

$$T = \frac{P!(L-B)!}{L!(P-B)!}, \text{ and } \gamma = T(T-1) + 1 \tag{3.7}$$

with $J(\mathcal{G}) < \gamma^b$ then we define a solution of $S(B, L, P)$ such as the collection $C = \{L(b) | b \in T\}$. As $J(\mathcal{G}) < \gamma^B$, each subset of P of size L appears at most once in C. Furthermore, we have

$$\frac{P!(L-B)!}{L!(P-B)!}\binom{L}{B} = \binom{P}{B}$$

then each subset of P of size L appears; we have a Steiner system $S(B, L, P)$. □

REMARK 1. *In fact, finding the optimal solution of $MINVAR(T, P, L, \gamma)$ with*

$$T = \frac{P!(L-B)!}{L!(P-B)!} \text{ and } \gamma > T(T-1)$$

is equivalent to know if there exists a Steiner system $S(B, L, P)$.

It is well-known that Steiner systems are very difficult to find. Therefore Problem $MINVAR(T, P, L, \gamma)$ is a complex problem in the general case with no obvious efficient algorithmic solution.

We now provide a lower bound for the interference function $J(\mathcal{G}, \gamma)$, which will be useful later to validate solution heuristics.

THEOREM 3.5. *For every bipartite graph $\mathcal{G} = (\mathcal{T}, \mathcal{P}, \mathcal{E})$ with $|\mathcal{T}| = T$ and $|\mathcal{P}| = P$ and such that $|L(b)| = L$ for all $b \in \mathcal{T}$, we have:*

$$J(\mathcal{G}, \gamma) \;\geq\; T(T-1)\, \gamma^{\frac{T^2L^2/P - TL + q(P-q)/P}{T(T-1)}} \;, \tag{3.8}$$

where $q = (TL) \mod P$.

Proof. We have, using the convexity of the power function $f(x) = \gamma^x$:

$$\begin{aligned}
\frac{J(\mathcal{G}, \gamma)}{T(T-1)} &= \sum_{b \neq b'} \frac{1}{T(T-1)} \gamma^{(|L(b) \cap L(b')|)} \\
&\geq \gamma^{\left(\sum_{b \neq b'} \frac{1}{T(T-1)} |L(b) \cap L(b')|\right)} \;.
\end{aligned}$$

Using the notation $\mathbf{1}_{b,p} = 1$ if $p \in L(b)$ and 0 otherwise,

$$\begin{aligned}
\sum_{b \neq b' \in \mathcal{T}} |L(b) \cap L(b')| &= \sum_{b \neq b' \in \mathcal{T}} \sum_{p \in \mathcal{P}} \mathbf{1}_{b,p} \mathbf{1}_{b',p} \\
&= \sum_{p \in \mathcal{P}} \sum_{b \neq b' \in \mathcal{T}} \mathbf{1}_{b,p} \mathbf{1}_{b',p} \\
&= \sum_{p \in \mathcal{P}} \delta(p) \times (\delta(p) - 1) \\
&= \sum_{p \in \mathcal{P}} \delta(p)^2 - T \times L \;.
\end{aligned}$$

The square function is convex and the minimum of

$$\sum_{p \in \mathcal{P}} \delta(p)^2 \,, \text{ under constraint } \sum_{p \in \mathcal{P}} \delta(p) = T \times L \,,$$

is obtained for values of $\delta(p) = \lfloor TL/P \rfloor$ or $\delta(p) = \lceil TL/P \rceil$.

We decompose $TL = Pr + q$ with $1 \leq q < P$, and any vector $(\delta(p))_p$ which components are q times the value $r + 1$ and $P - q$ times the value r minimizes the function while obeying the constraint. Evaluating the function for such vectors gives the lower bound. □

4. Heuristics. In this part, we analyse heuristics to solve $MINVAR(T, P, L, \gamma)$ problem. We propose differents algorithms: a greedy algorithm, two algorithms based on the round-robin paradigm, and two based on random choices.

```
Algorithm 1: Greedy
Data: $T$, $P$, $L$ and $\gamma$
Result: A bipartite graph $\mathcal{G}$
begin
    $\mathcal{G} = \emptyset$
    for $k \in \{1 \cdots L\}$ do
        for $i \in \{0 \cdots T-1\}$ do
            $j = \arg\min_j \{J(\mathcal{G} \cup (i,j), \gamma) + J((\mathcal{G} \cup (i,j))^c), \gamma)\}$
            $\mathcal{G} = \mathcal{G} \cup (i,j)$
end
```

4.1. Greedy Algorithm. This algorithm begins with an empty graph and, at each iteration, finds the edge which minimizes $J(\mathcal{G}, \gamma) + J(\mathcal{G}^c, \gamma)$, where $\mathcal{G}^c$ is the complementary bipartite graph of $\mathcal{G}$. It turns out that this algorithm gives better results than a simple greedy algorithm that would minimize $J(\mathcal{G}, \gamma)$ at each step. This is illustrated by the example (where $L = 2$ and $\gamma = 3$) given by Figure 4.1 .

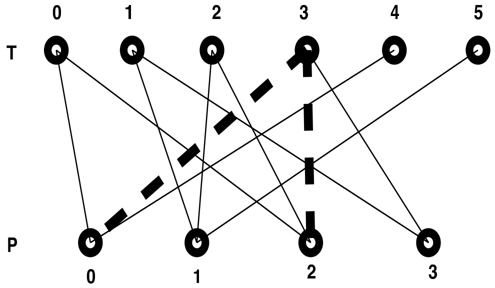


FIG. 4.1. *A greedy example for $L = 2$ and $\gamma = 3$*

Starting from the graph represented with continuous edges, choosing any of the dotted edges gives the same result for $J(\mathcal{G}, \gamma)$, but only the edge $(3, 2)$ minimizes $J(\mathcal{G}, \gamma) + J(\mathcal{G}^c, \gamma)$. It turns out that this is the only choice towards the optimal solution. Minimizing $J(\mathcal{G}, \gamma) + J(\mathcal{G}^c, \gamma)$ therefore guides the choice of edges towards better solutions.

4.2. Round-Robin Algorithm. Round-robin algorithms are generally based on a cyclical scan of vertices or edges. We propose two variants.

Algorithm 2: Blocks Round-Robin (BRR)

```
Data: $T$, $P$ and $L$
Result: A bipartite graph $\mathcal{G}$
begin
    $\mathcal{G} = \emptyset$ ; $j = 0$
    for $i \in \{0 \cdots T-1\}$ do
        for $k \in \{1 \cdots L\}$ do
            $\mathcal{G} = \mathcal{G} \cup (i, j)$ ;$j = j + 1 \bmod P$
end
```

In this Blocks Round-Robin (BRR) algorithm, we give as neighborhood to each vertex of $\mathcal{T}$, the L “next” vertices of $\mathcal{P}$.

Algorithm 3: Mixed Round-Robin (MRR)

```
Data: $T$, $P$ and $L$
Result: A bipartite graph $\mathcal{G}$
begin
    $\mathcal{G} = \emptyset$
    $j = 0$
    for $k \in \{1 \cdots L\}$ do
        for $i \in \{0 \cdots T-1\}$ do
            while $(i, j) \in \mathcal{G}$ do
                $j = j + 1 \bmod P$
            $\mathcal{G} = \mathcal{G} \cup (i, j)$
            $j = j + 1 \bmod P$
end
```

In the Mixed Round-Robin (MRR) algorithm, we scan simultaneously the vertices of $\mathcal{T}$ and $\mathcal{P}$, and select the corresponding edge unless it is already in the graph; in that case the corresponding partner is skipped. After L turns, each vertex of $\mathcal{T}$ has a complete neighborhood.

4.3. Random Algorithms. The RandomSubset algorithm computes for a vertex b its neighborhood of size L among the P possible vertices. It draws uniformly at random one neighborhood among the $\binom{P}{L}$ possibilities.

```
Algorithm 4: RandomSubset
  Data: $b$, $P$ and $L$
  Result: $L$ vertices from a vertex $b$
  begin
      $S = \emptyset$
      while $|S| < L$ do
          $j = random(1 \cdots P)$
          $S = S \cup (b, j)$
  end
```

The Random Algorithm simply performs such a choice, independently, for each $b \in \mathcal{T}$.

```
Algorithm 5: Random
  Data: $T$, $P$ and $L$
  Result: A bipartite graph $\mathcal{G}$
  begin
      $\mathcal{G} = \emptyset$
      for $b \in \{0 \cdots T-1\}$ do
          $S$=RandomSubset($b$, $P$, $L$)
          $\mathcal{G} = \mathcal{G} \cup S$
  end
```

We show now that it is possible to compute the average value of the objective function for such randomly generated solutions. The key result for this is the distribution of the number of common partners in two neighborhoods.

Theorem 4.1. *Let $X_{bb'} = |L(b) \cap L(b')|$ be the number partners common to both neighborhoods. For every $b \neq b'$, the distribution of $X_{bb'}$ is given by:*

$$\mathbb{P}(X_{bb'} = k) = \frac{\binom{L}{k}\binom{P-L}{L-k}}{\binom{P}{L}} \qquad \textit{and}$$
$$\mathbb{E}(X_{bb'}) = \frac{L^2}{P}\,.$$

Proof. If k elements of $\mathcal{P}$ are common between $L(b)$ and $L(b')$, the set $L(b')$ is formed by adding $L-k$ elements chosen among the $L-P$ elements that are not in $L(b')$ (see the interpretation of the Chu-Vandermonde convolution in e.g. [6, p. 56]). Hence the formula for $\mathbb{P}(X_{bb'} = k)$. As a consequence,

$$\begin{aligned}\mathbb{E}(X_{bb'}) &= \sum_{k=0}^{L} k\mathbb{P}(X_{bb'} = k) = \sum_{k=0}^{L}(L-k)\frac{\binom{L}{k}\binom{P-L}{k}}{\binom{P}{L}} \\ &= L\sum_{k=0}^{L}\frac{\binom{L}{k}\binom{P-L}{k}}{\binom{P}{L}} - \sum_{k=0}^{L}k\frac{\binom{L}{k}\binom{P-L}{k}}{\binom{P}{L}} \\ &= L - (P-L)\,\frac{\binom{P-1}{L-1}}{\binom{P}{L}} = \frac{L^2}{P}\,. \qquad \square\end{aligned}$$

Remark 2. *In practical cases, Theorem 4.1 gives us the average number of the blocks in common for a given number of partners in the GDN. For example, if the number of partners is* 100 *and if* $L < 10$*, we can conclude that this number of blocks in common is less than 1 in the GDN.*

Corollary 4.2 gives the average value of the objective function for this algorithm.

Corollary 4.2. *If* $\mathcal{G}$ *is the graph generated by Algorithm 5, then the average interference function is*

$$\mathbb{E}(J(\mathcal{G},\gamma)) = \frac{T(T-1)}{\binom{P}{L}} \sum_{k=0}^{L} \gamma^k \binom{L}{k}\binom{P-L}{L-k} .$$

If we are convinced that it is possible to find a solution where $|L(b) \cap L(b')| \leq K$ for each $b, b' \in \mathcal{T}$, then we may improve the random algorithm : we refuse to accept a neighborhood that has more than K vertices in common with the previous neighborhoods. After Nb_Max unsuccessful tries, we increase K to accept neighborhoods more easily.

For example, if we consider 100 partners and the replication $L = 30$, we obtain by theorem 4.1 an average intersection of 9. We may want to consider the following constraint: forbid an intersection number greater than $K = 5$. We come back to the effectiveness of this idea in Section 5.

Thus we obtain the variant fully described in the following Algorithm named Random2.

Algorithm 6: Random2

```
Data: $T$, $P$, $L$, $Nb\_Max$ and $K$
Result: A bipartite graph $\mathcal{G}$
begin
    $\mathcal{G} = \emptyset$
    for $b \in \{0 \cdots T-1\}$ do
        OK=false
        nb=0
        while $!OK$ do
            $S$=RandomSubset($b$, $P$, $L$)
            OK=true
            for $b' \in \{0 \cdots b-1\}$ do
                if $|L(b') \cap S| > K$ then
                    OK=false
                    nb=nb+1
                if $nb > Nb\_max$ then
                    nb=0
                    K=K+1
        $\mathcal{G} = \mathcal{G} \cup S$
end
```

5. Experimental Evaluation. We proceed now to determine how well these algorithms perform. Since the optimal solution to problem MINVAR is not known, we compare the outcome of the algorithms with an ideal situation given by the theoretical lower bound given in Theorem 3.5. In this section, we describe the experimental settings and then discuss the results.

5.1. General settings. We have built a specific simulator for the heuristics of Section 4. The program implements the three deterministic algorithms, and for the random algorithms, it allows to generate any specified number of independently drawn samples. We have carried out extensive experiments to evaluate the respective performance of the algorithms. In all simulations, we assume $\gamma = 2$, which corresponds to an availability of $\delta = 50\%$ for hosts. The size of documents ranges from $T = 100$ to 1000 blocks. The number P of hosts is 10 or 100, see the figure's captions.

5.2. Comparison. Figures 5.1, 5.2 and 5.3 display the ratio of the objective function to the lower bound of Proposition 3.5, for various sets of parameters. We observe on those figures that the two round robin algorithms have a relatively bad performance, which becomes very bad for large values of L.

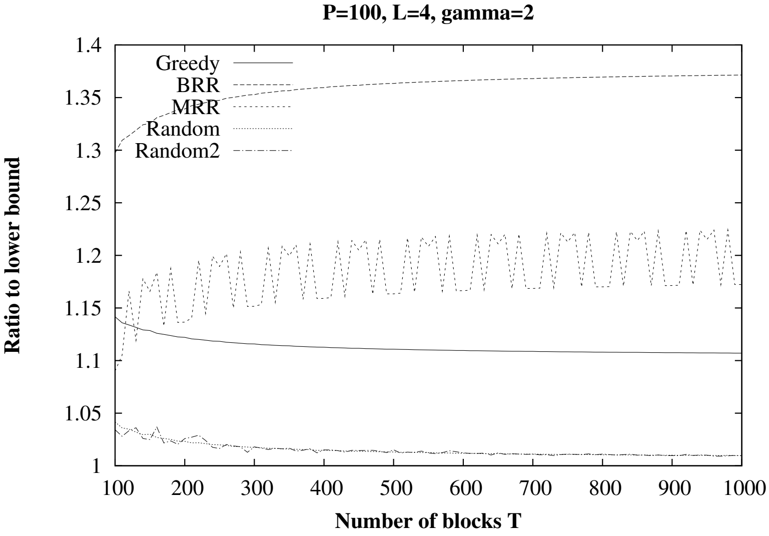


FIG. 5.1. *Algorithms Performance for P=*100*, L=*4

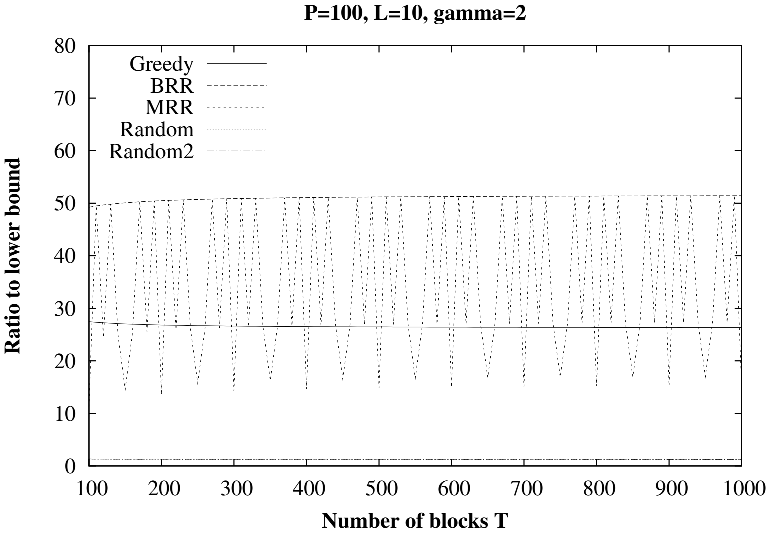


FIG. 5.2. *Algorithms Performance for P=*100*, L=*10

In real data distribution schemes, these two algorithms are often used because of their simplicity. It is interesting to see that for the present optimization problem, their performance is very far from the lower bound. In addition, the result of MRR appears to suffer sometimes from large fluctuations of arithmetic nature.

Another information is that the performance of the greedy algorithm is very dependent on the data. For some values of T, its results are better but for some other worst, than the random algorithms. Generally, the algorithm seems to perform well for small L and P parameters. When P gets large, the algorithm performs relatively well if L is small (ex. $L = 4$), but degrades if L is larger (ex. $L = 10$).

The figures also show that the greedy algorithm and the two random algorithms are very close to the lower bound when L is not too large.

We investigate this issue more closely using Figure 5.4, which is a zoom on Figure 5.3. This figure, as well as Figure 5.1, top, suggest that all three algorithms have an asymptotically robust performance when T is large, in the sense that their performance ratio remains bounded. However, the greedy algorithm turns out to be numerically unstable. Evaluating the J function is very hard for large values of T, L and P. Another

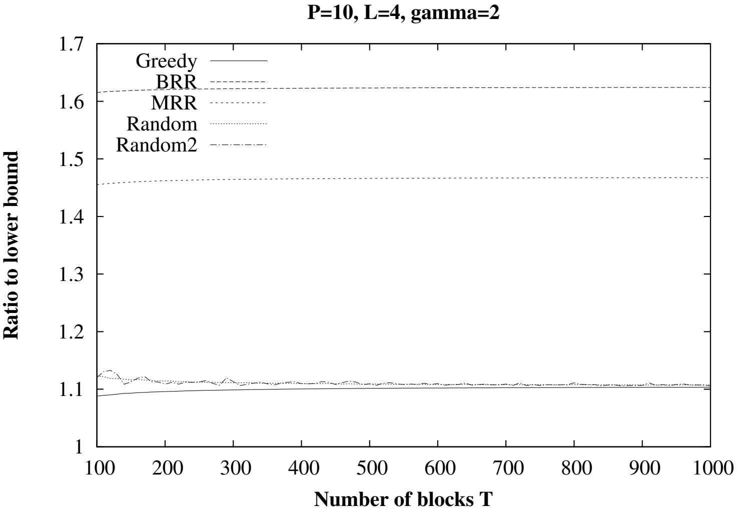


FIG. 5.3. *Algorithms Performance for P=10, L=4*

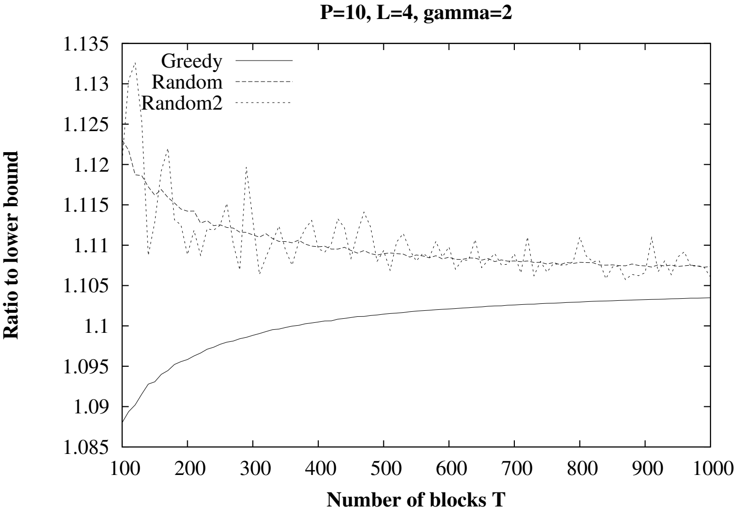


FIG. 5.4. *Best algorithms comparison*

disadvantage for the greedy algorithm is its algorithmic complexity: it is $O(PLT^3)$ whereas the complexity of the random algorithms is $O(PLT)$.

It appears therefore that the random algorithms are the best choice to determine a good block allocation.

5.3. Analysis of the random algorithm. Focusing now on the random algorithms, we compare in Figure 5.5 the average of the 1000 experiments with the *best* allocation found among them. For the Random2 algorithm, we take $Nb\_Max = 1000$.

We observe that the ratio is very close to 1; the standard deviation is actually very small. We conclude that in practice, the first allocation obtained by a random algorithm is probably already close to the optimal solution.

Finally, in Figure 5.6, we compare the Random and Random2 algorithms, through the ratio of their average and best performances.

In general, these two algorithms give similar results, but in some particular cases, there is a real improvement for Random2. In this figure, we have $P = 100$, $L = 30$ and $\gamma = 3$: the average interference number for each couple of blocks is 9 (see corollary 4.2). Still, rejecting neighborhoods with more than 5 vertices in common with

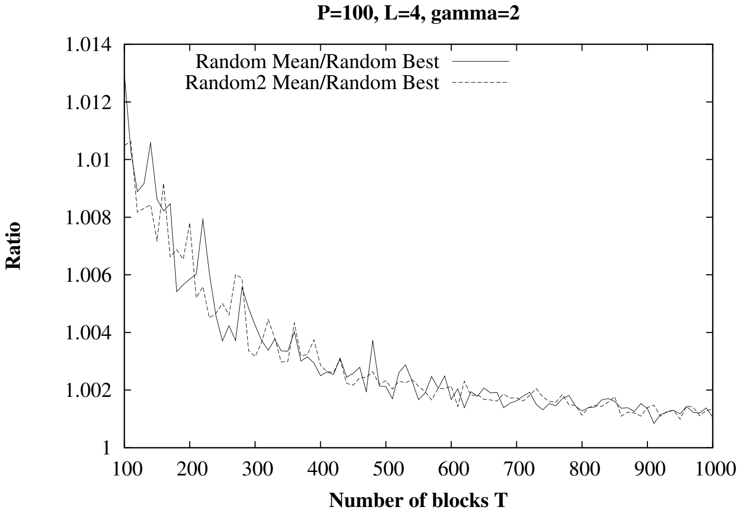


FIG. 5.5. *Ratio average to best*

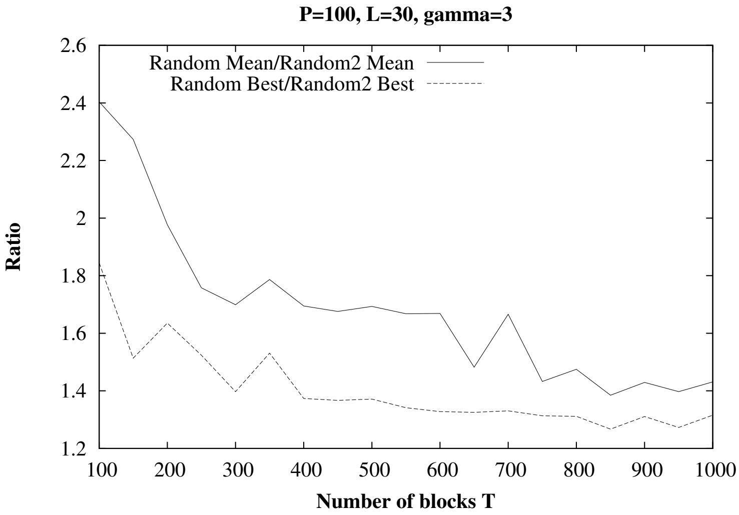


FIG. 5.6. *Random vs Random2 algorithm*

the previous neighborhoods turns out to be possible and this improves the block allocation found by $20 - 40\%$ for a large range of document sizes. For a smaller range of documents size (between 100 and 400), we can say that we obtain an average improvement of 80%.

6. Conclusion and future work. In this paper, we have proposed, and analyzed by simulation, various algorithms for optimizing the variance of download times in a particular distributed architecture, by placing appropriately replications on distributed data servers. In particular, we have shown that conventional Round Robin algorithms do not perform well in general, and that random algorithms are the best practical choice. Our methods have been applied to a real GDN deployment, and we are currently investigating their effectiveness in practice.

REFERENCES

[1] A.-E. BAERT, V. BOUDET, AND A. JEAN-MARIE, *Performance analysis of data replication in grid delivery networks*, in Int. Conf. on Complex, Intelligent and Software Intensive Systems, 2008, pp. 369–374.

[2] C.-F. Chou, L. Golubchik, and J. C. S. Lui, *Striping doesn't scale: How to achieve scalability for continuous media servers with replication*, in Int. Conf. on Distributed Computing Systems, 2000, pp. 64–71.

[3] C. Colbourn and J. Dinitz, *The CRC handbook of combinatorial designs, 2nd edition*, 2006.

[4] S. Gonzalez, S. Navarro, A. Lopez, and E. L. J. Zapata, *A case study of load sharing based on popularity in distributed VoD systems*, in IEEE transactions on Multimedia, vol. 8, 2006, pp. 1299–1304.

[5] T. Ibaraki and N. Katoh, *Resource allocation problems: algorithmic approaches*, MIT Press, Cambridge, MA, USA, 1988.

[6] D. Knuth, *The art of computer programming Vol. 1*, Addison-Wesley, 1973.

[7] F. MacWilliams and N. Sloane, *The theory of error-correcting codes*, Horth Holland, 1977.

[8] R. Susitaival and S. Aalto, *Analyzing the file availability and download time in a P2P file sharing system*, in Int. Conf on Next Generation Internet Networks, 2007, pp. 88–95.

[9] K. Tanaka, H. Sakamoto, H. Suzuki, and K. Nishimura, *Performance improvements of large-scale video servers by video segment allocation*, Syst. Comput. Japan, 35 (2004), pp. 27–35.

[10] N. Venkatasubramanian and S. Ramanathan, *Load management in distributed video servers*, in Int. Conf. on Distributed Computing Systems, 1997.

[11] VodDNET Company, `http://www.voddnet.com`.

[12] J. Z. Wang and R. K. Guha, *Data allocation algorithms for distributed video servers*, in ACM int. conf. on Multimedia, New York, NY, USA, 2000, ACM Press, pp. 456–458.

[13] X. Zhou and C.-Z. Xu, *Efficient algorithms of video replication and placement on a cluster of streaming servers*, J. Netw. Comput. Appl., 30 (2007), pp. 515–540.

Edited by: Fatos Xhafa, Leonard Barolli
Received: September 30, 2008
Accepted: December 15, 2008

Scalable Computing: Practice and Experience
Volume 10, Number 1, pp. 87–98. http://www.scpe.org

ISSN 1895-1767
© 2009 SCPE

VIRTUAL LARGE-SCALE DISK BASE ON PC GRID[*]

ERIANTO CHAI, KATSUYOSHI MATSUMOTO, MINORU UEHARA AND HIDEKI MORI[†]

Abstract. With the recent flood of data, one of the major issues is the storage thereof. Although commodity HDDs are now very cheap, appliance storage systems are still relatively expensive. As a result, we developed the VLSD (Virtual Large-Scale Disk) toolkit to assist in the construction of large-scale storage using only cheap commodity hardware and software. As an experiment in using the VLSD toolkit, storage was created for an educational environment. This paper presents an evaluation of the performance of the storage by performing a GCC build on the trial production storage system both during trouble-free operation and failure.

Key words: virtual large-scale disk; grid; open source; raid; fault tolerance

1. Introduction. Large-scale storage has become a necessity as we are flooded with information. Large-scale storage is needed for research in areas involving large amounts of data, such as the Human Genome Project or calculation engineering. In addition, a large amount of data is used privately for video and music storage, etc. Virtual machines such as VMware and Microsoft Virtual PC 2004 allow students to learn how to maintain Linux servers safely and are very useful in higher education, but they have large disk space requirements. VMware recommends 8GB as the HDD capacity for running Linux and 16GB for Windows. In our experiments, we found that running Windows actually requires 24GB HDD capacity. This means that 40-120TB HDD capacity is needed to support 5000 students (Figure 1.1).

To service this requirement, we considered an appliance file server, with 70TB capacity. The quote for such a file server was 250 million yen (about 2 million dollars). However, we already have 500 PCs. If we added 180GB HDDs to each PC at a cost of 150 dollars per HDD, the 70TB distributed file server could be realized at a cost of just 75 thousand dollars. This represents a cost ratio of 1:33, making the cost saving more important than any other factor such as performance, dependability or maintainability. Thus we proposed a distributed storage system and have developed a toolkit that can assist in constructing such a system. We call this the VLSD (Virtual Large-Scale Disk) toolkit. [3], [4].

When we set about constructing a large-scale storage system, we tried mounting several isolated disks using a distributed file system such as SMB/CIFS or NFS. There are, however, two fundamental problems with this file system based approach. The first problem is not being able to utilize the full storage capacity. For example, when we construct 70TB of storage using 180GB HDDs, the maximum size of any file must be less than 180GB. The total size of all files in a directory must also be less than 180GB. In addition, for some OSs it is impossible to create a file greater than 4GB in such a physical file system. The second problem is linked to the Linux OS platform. Almost all distributed file systems are based on the NFS protocol, which is not suited to Windows. Windows clients have a larger capacity for HDDs than a Linux server. Thus a file system based on a distributed storage system such as NFS can neither stripe physical storage nor concatenate disks. We therefore used a disk based approach, in which a virtual disk was created in a distributed physical file system.

VLSD can concatenate multiple virtual disks, thus increasing the storage capacity. VLSD also supports typical RAID classes such as RAID0, 1, 4, 5, and 6. It can construct multiple hierarchies of RAID combining any RAID class with any other RAID class.

VLSD is independent of platform because it is written in 100% pure Java. Using a VLSD, we constructed a trial 70TB storage system consisting of 484 PCs, with each PC providing 180GB free disk space. This system realizes an acceptable MTTF using RAID66, which is 2 layered RAID6.

In an educational environment storage system, a great number of small files tend to be generated. This corresponds to the Small Read and Small Write on the VLSD system. The performance of the storage built using a VLSD was evaluated both during normal operation and at times of failure. During disk failure, performance drops for the Small Read as illustrated by the graph in Fig. 3.9. This paper therefore, evaluates performance by performing a GCC build on the trial storage system during times of both trouble-free service and failure.

This paper is organized as follows. Section 2 summarizes related works. Section 3 describes the VLSD, while Section 4 explains the build and evaluation of GCC. Finally, we present our conclusions.

[*] This research was assisted by department laboratory grant: "Research(195000066) of high trust by the PC grid and highly effectivedecentralized virtual storage".

[†] Department of Information and Computer Sciences, Toyo University, 2100 Kujirai, Saitama, 3508585, Japan

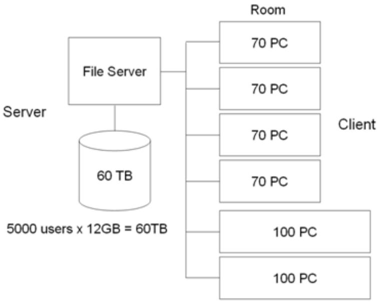


Fig. 1.1. University PC classroom

2. Related Work.

2.1. RAID. RAID (Redundant Arrays of Inexpensive Disks) [1] is used to raise the reliability of large-scale storage. It is a technique that increases the performance and fault-tolerance of data storage. These goals can be combined and are achieved by distributing data and creating data redundancy for recovering data on two or more hard disk drives. RAID levels are designated by integers from 0 to 6 each denoting a different structure of data redundancy and distribution.

RAID0 is striping without redundancy and thus capacity is increased with the addition of drives. RAID1 implements mirroring. Capacity therefore does not increase with the addition of a second drive. In a RAID2 array, a disk is made to distribute each bit and data errors are corrected by ECC. Since there is little improvement compared with a parity based system, RAID2 is hardly used. RAID3 uses parity to correct errors and carries out striping per bit or per byte. Usually, parity is saved on a separate disk. RAID3 and RAID4 differ only in the point at which striping is performed. Both record parity on a separate disk, which can become a bottleneck. RAID5 distributes where parity is saved. Since a bottleneck does not occur, performance is high. RAID5 is however, susceptible to mistakes where an operator removes the drive under drive, causing two drives to fail simultaneously. RAID6 is an extension of RAID5 and can sustain a double failure by copying parity data twice in two separate ways. Under RAID6 it is possible to recover without losing data even if two disks fail at the same time.

RAID arrays are classified into software (SW) and hardware (HW) systems. HW RAID is predominant, as SW RAID is considered to have lower performance than HW RAID and adds load to the CPU. However, the latest CPUs have higher performance than a HW RAID controller. CPU control of SW RAID is effective in machines such as file servers that can concentrate on the RAID operation. Moreover, SW RAID is the only option for implementing RAID across a network.

2.2. RAID6. There are two types of parity in RAID6, namely P parity and Q parity. [5] P parity is generated by XORing the block data of all the disks, while the Q parity used by RAID6 is more complicated to calculate. Generation of Q parity is by a Galois Field (GF) operation. Although a GF is called a limited object in algebra, it is also called a Galois object or region in computer-related fields. A GF is a set of values containing a finite number of elements. The GF of a 2^8 piece element is denoted as GF (2^8), and contains an integer element from 0 to $2^8 - 1$.

There are two types of parity in RAID6 (P parity and Q parity) [5]. P parity is generated from XORing the block data of all the disks. Q parity used by RAID6 is more complicated to calculate than P parity. Generation of Q parity is by Galois Field (GF) operation. Although a GF is called a limited object in algebra, it is also called a Galois object or region in computer-related fields. A GF is a set of values containing a finite number of elements. The GF of a 2^8 piece element is denoted as GF (2^8), and there is an element from an integer 0 to 2^{8-1}.

Four cases can be considered when two disks fail in RAID6: P and data drive failure, Q and data drive failure, P and Q drive failure, and failure of two data drives.

2.3. NBD. NBD [6] is a virtual disk system that can be compiled into the Linux kernel. The block device that this remote server offers can be handled as a local drive by clients. An arbitrary file system can be constructed in a high-order layer because it inputs and outputs at the block level. ISCSI etc. offers similar block level I/O. Because NBD has been adopted by Linux as standard it is far simpler to install compared with the alternatives. Moreover, mounting the protocol is simple and easy. The proposed system, however, does not depend on NBD. It should be noted that NBD is merely one of the available choices.

2.4. Samba. Samba [7] is used to access the file system on Linux, UNIX and other similar operating systems from Windows clients. Samba is free software that can be used to mount network shares created using Microsoft Corporations NetBIOS protocol. File and print services can also be provided to Windows clients by UNIX-like operating systems such as Linux, Solaris, and FreeBSD using Samba. The first version was developed by Andrew Tridgell, and it was opened to the public under the GPL in 1992. In this research, Samba is used to transmit requests from Windows clients to NBD.

2.5. NFS. NFS [8] enables directories and files to be shared with other machines via a network. Users and programs can access files on remote systems using NFS in the same way as they would a local file. NFS is a de facto standard distributed file system in UNIX. In Linux NFS is supported as standard. Moreover, Windows can also employ NFS using the Service for UNIX (SFU). However, operations that use SFU are more difficult than operations that use Samba. Therefore, NFS is not used by the client OS to connect to the network in this research. However, it is used to transmit the requests from Linux clients to NBD.

2.6. XFS. XFS [9], widely recognized as a highly efficient file system, offers quick restoration from system crashes and support for very large disks. XFS was the first journaling file system for Linux, and has built a strong track record in production environments because the second half of 1994. XFS makes it possible to control a huge file system using a 64-bit file system or allocation group. Handling large files presents problems both in securing the writing/reading area, and in the time taken to retrieve an empty area on the disk. In XFS, this is handled by a delay allocation and B+-Tree. When the resources on an XFS disk run short they can be extended; the logical volume can be expanded and the file system can easily be resized, without unmounting. XFS is also efficient, in that it does not generate unnecessary disk activity.

3. VLSD. VLSD is a toolkit for constructing large-scale storage systems. It is written in pure Java and can therefore be used on any platform on which Java runs, such as Windows, Linux, etc. VLSD can even be used if there is no native NBD server in the OS, as the toolkit includes software for RAID and NBD. VLSD contains typical RAID classes and NBD client/server software, thus allowing the combination of any RAID class and NBD devices with one another.

3.1. Class of VLSD. From a software perspective, VLSD consists of the following classes.

- **NBDServer**: The NBD server is called by the client, and offers empty capacity to the storage system as a virtual disk file. The OS of the client can be either Windows or Linux. Since the NBD server is mounted using Java, it is platform independent. Moreover, two or more disks may be added to the array by a single client. FAT32 may also be used despite its 4GB maximum file size restriction. Because a virtual disk created using 120GB drives cannot be used as a single file, a virtual disk is created by bundling two or more files and combining them with RAID0 or JBOD as described later.
- **DiskServer**: Is an interface to the disk server.
- **DiskServerImpl**: Is an implementation of the disk server, and a remote disk using RMI.
- **Disk**: Is an interface to a virtual disk.
- **AbstractDisk**: Is a class of an abstract virtual disk that defines the constants and methods that are used in low order classes.
- **DiskArray**: RAID1 is mounted by the base class of the disk wrapper that consists of two or more disks.
- **RAID**: The base class for RAID. Mounting of RAID1 is inherited from DiskArray.
- **SingleDisk**: Is the base class of the disk wrapper which consists of a single disk.
- **PagedDisk**:Is a wrapper for an arbitrary disk accessed via disk paging. For the wrapped disk, read/write is per page only. The fraction on each page is disregarded.
- **VariableDisk**: Is a disk with changeable capacity that does not direct resources beforehand, but can have resources secured dynamically if necessary.
- **RemoteDisk**: Is a remote disk, where the disk server is accessed.
- **RAID0**: RAID0 is used to increase capacity. It differs from JBOD, described below, in that striping is performed. Though the performance is good, adding additional disks only adds capacity equal to that of the smallest drive in the array. For example, striping drives of 100GB, 120GB, and 160GB results in 300GB only 100GB x 3. It is better to use JBOD when aiming purely to increase capacity, although

RAID0 can be expected to improve performance. In some file systems, the super block that manages i-nodes concentrates disk activity in a specific area. In such a case, striping has the effect of distributing this load. RAID0 performs per byte striping.

- **RAID5**: Parity is distributed and stored on each disk. The disk where parity is stored is different for each block.
- **RAID6**:The RAID6 class allows the generation of the GF table, block unit striping, and distributed parity to be performed.
- **JBOD**: This class does not provide redundancy like RAID0 and is used to increase capacity only. Because striping is not performed, capacity is simply summed. For example, the total capacity is 380GB if 100GB, 120GB and 160GB drives are connected. JBOD, though, lacks the load-balancing effect of RAID0 as described above. Since caching effectively creates a certain amount of scale, RAID0 performance may be superior.

Table 3.1 gives the required time for each class to do small read and small write operations, while Table 3.2 gives the required time to do large read and large write operations. The experiment is conducted on one machine. The block size of one disk stripe is 8KB and we executed 105 loops for small read, 103 loops for small write, 104 loops for large read and 102 loops for large write using random positions on the disk. As shown in Table 3.1, JBOD is faster than RAID0 on one machine. However, RAID0 can be faster than JBOD when executing on two or more machines.

Table 3.1
Required Time of each class on Small Read and Small Write

Class	Small Read (s)	Small Write
FileDisk	0.97	0.86
VariableDisk	1.09	0.94
JBOD (2 disks)	1.44	1.10
PagedDisk	2.03	1.22
SingleDisk	1.00	0.91
RAID0 (2 disks)	2.08	1.03
RAID1 (2 disks)	1.00	1.42
RAID5 (3 disks)	2.13	1.42
RAID6 (4 disks)	2.14	1.67

Table 3.2
Required Time of each class on Small Read and Small Write

Class	Large Read (s)	Large Write
FileDisk	-	-
VariableDisk	-	-
JBOD (2 disks)	-	-
PagedDisk	-	-
SingleDisk	-	-
RAID0 (2 disks)	0.52	0.17
RAID1 (2 disks)	0.28	0.17
RAID5 (3 disks)	0.52	0.36
RAID6 (4 disks)	0.50	0.59

3.2. Trial Production of the Large-Scale Storage Using VLSD. In this section we discuss the design of a 70TB distributed storage system in the environment described in Section 1. This system comprises 484 client PCs each of which provides 180GB free disk space. The total free space is 85TB. However, as a client PC may be inadvertently shut down, this makes the distributed storage system no more reliable than a conventional file server. We overcame this problem by using hierarchical RAID, RAID66, which is two layered RAID6 (Fig. 3.1).

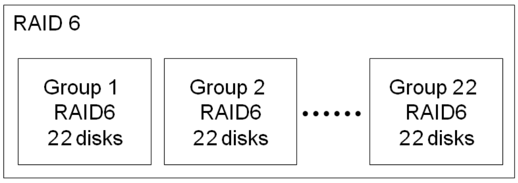


Fig. 3.1. *Composition of RAID66*

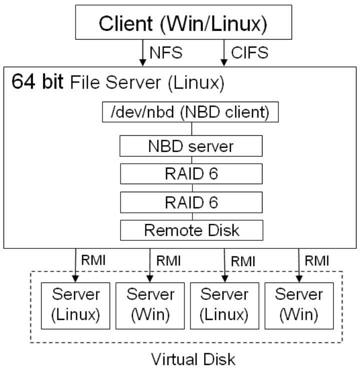


Fig. 3.2. *System Overview*

The trial system includes a 64-bit file server and the disk server as shown in Fig. 3.2. A virtual disk that supports OSs such as Linux and Windows for the disk servers provides read/write access to the disk via the Java RMI. The file server connects to the prepared disk and constructs RAID66. RAID66 is RAID6 on two levels. The NBD Server waits for access by an NBD Client. After the NBD Client is started, it is formatted with XFS. Windows clients access the file server through Samba, while NFS is used for Linux clients.

Since the NBD protocol lacks security, applying it on a network is dangerous. However, because NBD is used in our system for inter-process communication only, it can be applied without risk. Actual communication between client/server is realized by a protocol based on RMI with the relevant security considerations implemented.

Disk servers (of which there are 484) are started with the following command.

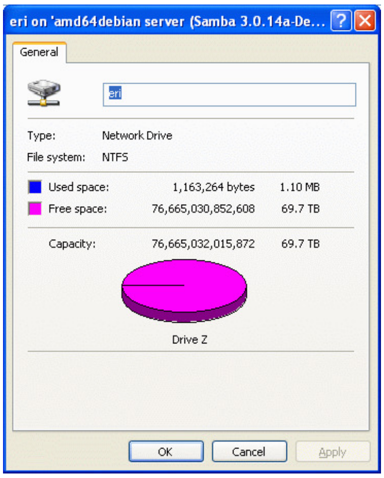
```
# java DiskServerImpl //localhost/pcn/DiskServerImpl imagen*
```

Here, imagen is a virtual disk of 180GB. It is executed on the file server side as follows.

```
# java vlsd.server.RAID66 9000
# nbd-client localhost 9000 /dev/nb0
# mkfs.xfs /dev/nbd0
# mount /dev/nbd0 /home/eri
# java vlsd.server.RAID66 9000
# nbd-client localhost 9000 /dev/nb0
```

Once execution has terminated successfully, storage can be used via a Windows mapped network drive. Figure 3.3 shows the disk drive properties of the shared disk and confirms the capacity of 70TB. Although the capacity of 484 x 180GB disks is 87TB because two per group parity is needed by RAID6, for a RAID6 array which consists of N nodes, only (N-2) / N disks can store data. Since RAID66 becomes two classes only 82% of the whole capacity can be used (=20/22*20/22). Therefore, the actual capacity is 70TB. In addition, a variable capacity virtual disk has been used in this trial. Data was not necessarily actually saved to all 70TB. Many client file systems converted only some of the available disk area during formatting.

3.3. Comparison of fixed and a variable length disks. As an experiment, a fixed-length disk and a variable-length disk were formatted by XFS, and the format times compared. The disks were virtual and saved as a file. Unlike in a fixed disk, only the part actually required in the variable-length disk was written to an image file.



eri on 'amd64debian server (Samba 3.0.14a-De...

eri

Type: Network Drive

File system: NTFS

🟦 Used space:	1,163,264 bytes	1.10 MB
🟪 Free space:	76,665,030,852,608	69.7 TB
Capacity:	76,665,032,015,872	69.7 TB

Fig. 3.3. *Disk drive properties of a RAID66 array*

Table 3.3
Format of virtual disk

Variable-Length Disk	
Capacity of Virtual Disk	Capacity nessary for format
1GB	10.45MB
10GB	10.66MB
100GB	50.78MB
1TB	129.41MB
10TB	129.39MB
100TB	132.07MB
1PB	165.45MB
10PB	498.58MB
100PB	3829.71MB

The time to format a variable-length disk is faster as depicted in Fig. 3.4. We postulate that the reason is because seek times are quicker with a variable-length disk. While formatting the variable-length disk using XFS, the capacity required for the format was recorded. As shown in Table 3.3 about 10MB is necessary when the disk is formatted to 1GB, and roughly a hundred MB is required for a 1TB disk. Actual physical disk space of 500MB is necessary for 10PB, and this increases to 3.8GB to format a 100PB disk.

3.4. Evaluation of RAID under Normal Operation. We compared the performance of our system with the theoretical performance presented in the paper by Chen et al. (1994). In this paper, typical RAID arrays are evaluated by their throughput per dollar relative to RAID0. Small read/write means read/write access to a block. The series RAIDNe gives the experimental values, while the values in the series RAIDNt are the theoretical values given in the paper by Chen et al. The throughput looks good, but this is simply because RAID0 is not so good and therefore makes comparisons seem highly effective. In our experiments, JBOD actually outperforms RAID0.

As shown in Fig. 3.5, the experimental values for RAID5 and RAID6 are almost equal to their respective theoretical values. This means that cost performance is also almost equal. Since for a Small read the data can be collected by access to a single disk in RAID5 and RAID6, performance is equal to RAID0.

In Fig. 3.6 the experimental values for RAID5 and RAID6 are about four times faster than the respective theoretical values. This is perhaps explained by the fact that RAID0 is slow. In RAID6, calculation of parity becomes complicated in comparison to RAID5 because there are two parity data records, and processing times become correspondingly larger. As a result, as shown in Fig. 3.6, RAID6 has lower throughput than RAID5. And unlike RAID0, in a Small Write RAID5 and RAID6 both require four accesses, namely read-out of data, read-out of parity, writing data, and writing parity.

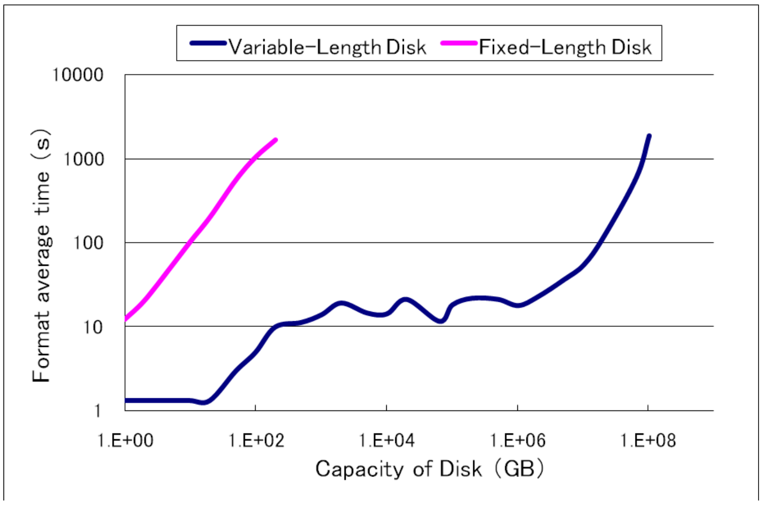


FIG. 3.4. *Comparison of a fixed and a wariable-length disk*

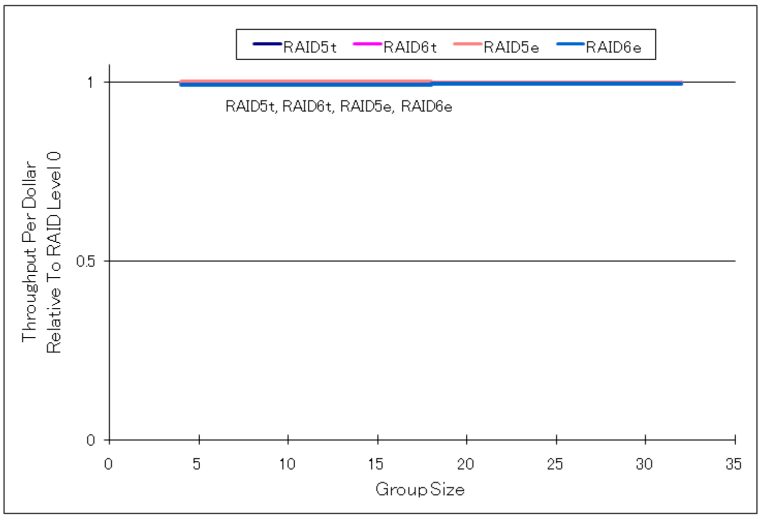


FIG. 3.5. *Small Read*

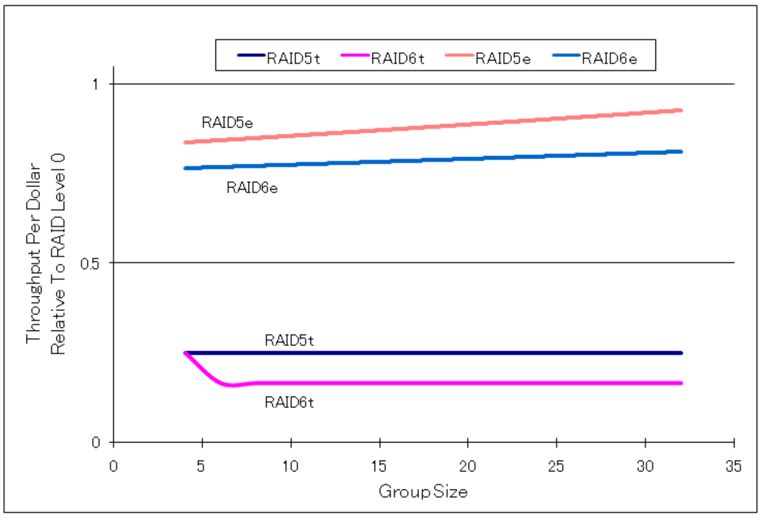


FIG. 3.6. *Small Write*

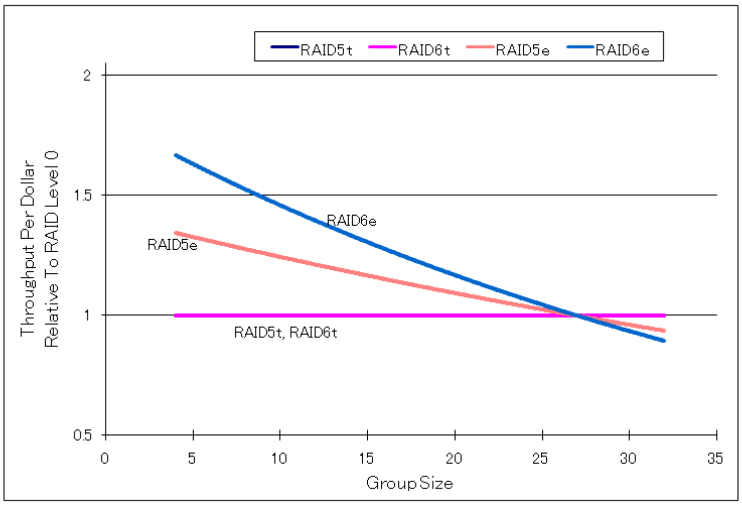


FIG. 3.7. *Large Read*

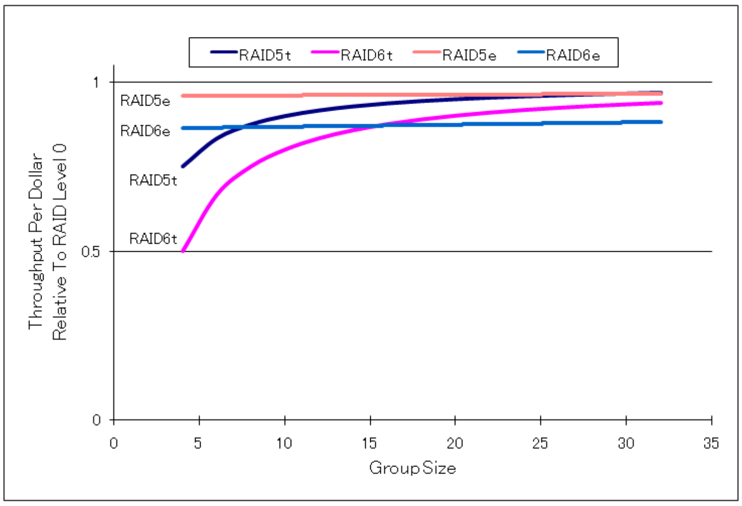


FIG. 3.8. *Large Write*

In the experimental values (Fig. 3.7) for Large Reads, although cost performance is very high for a small group size, it becomes lower for a large group. In a Large Read, because the reading domain of RAID6 is smaller than that of RAID5, cost performance is high. It is possible that the cost performance drops with a large group size because of an increase in some calculation processing.

The experimental values are almost constant in Fig. 3.8. For a large group size, the theoretical values exceed the experimental values for RAID6.

3.5. Evaluation of Small Read and Small Write of RAID6 under Drive Failure. Figures 3.9 to 3.12 depict graphs by throughput per dollar of typical RAID arrays using RAID6 in the case of a single disk failure (RAID6e1), and 2 disk failures (RAID6e2). The number of disk accesses for RAID6e1 and RAID6e2 is the same; the difference is that RAID6e2 has more parity processing operations.

In Small Read (Fig. 3.9), when one disk fails, cost performance falls to 25%, and when two disks fail, it falls to 10%. If the group becomes larger, the cost performance of RAID6e1 and RAID6e2 decreases. It is believed that in Small Read there is little disk access and almost all processing time is operation processing.

As shown in Fig. 3.10, in Small Write, the throughput of RAID6e1 is between 80% and 60%, while the throughput of RAID6e2 is about 70% to 40%. Because of the volume of disk access, the throughput of Small Write is higher than Small Read.

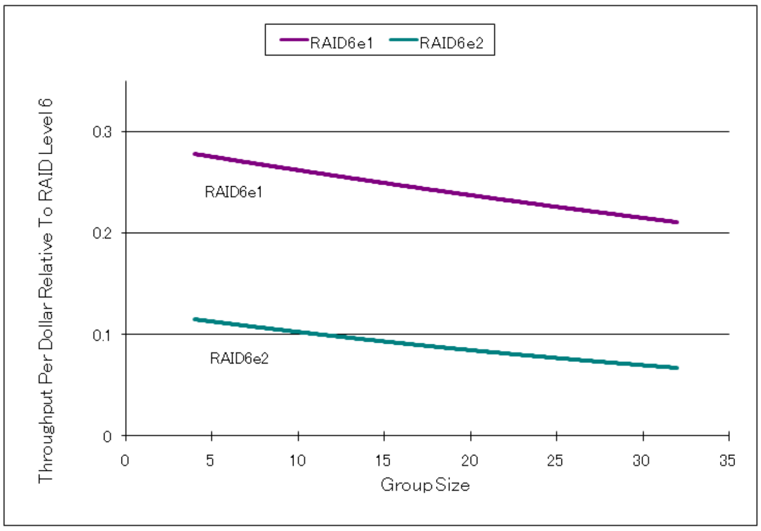


FIG. 3.9. *Small Read*

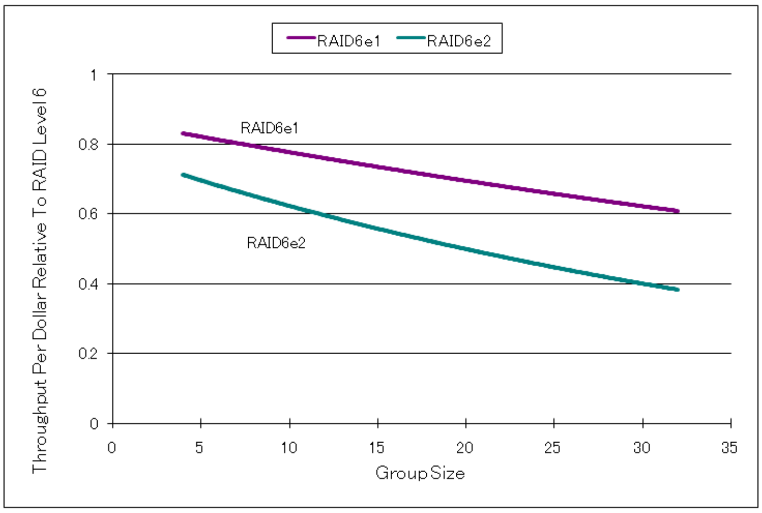


FIG. 3.10. *Large Write*

The difference between RAID6e1 and RAID6e2 is four fold in Large Read (Fig. 3.11). There is a throughput difference between RAID6e1 and RAID6e2 in both Large and Small Read. Meanwhile, because the disk writing time is far slower than the operation processing time in Small Write and Large Write (Fig. 3.12), there is no difference in throughput for Small Read and Large Read which perform disk reading only.

3.6. Reliability of hierarchy RAID on VLSD. The MTTF of the entire hierarchical RAID is calculated using the measured MTTF (mean time to failure) and MTTR (mean time to repair). Hierarchical RAID consists of a combination of RAID1, 5 and 6.

In Table 3.4, the total number of disks is 484. Layers 1 and 2 each contain 22 disks. RAID3 and 4 are omitted because they are the same as RAID5. Moreover, in RAID1 it was assumed that the contents were copied N times.

RAID1 should not be used, because the capacity efficiency decreases significantly. To achieve the largest capacity and highest MTTF, RAID66 is the best.

4. Building GCC. Many small files, such as text documents or source code of programs, are created in an educational environment. The size of most files is 1KB or less. Therefore, it is necessary to verify storage constructed with the VLSD in this environment.

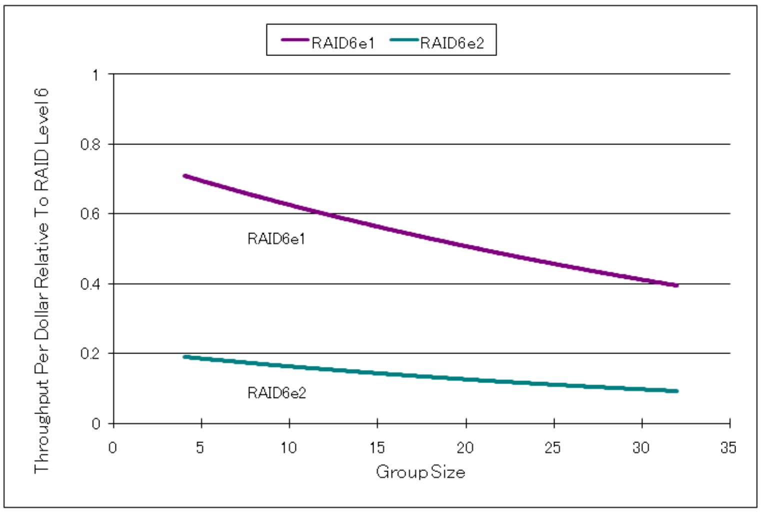


Fig. 3.11. *Large Read*

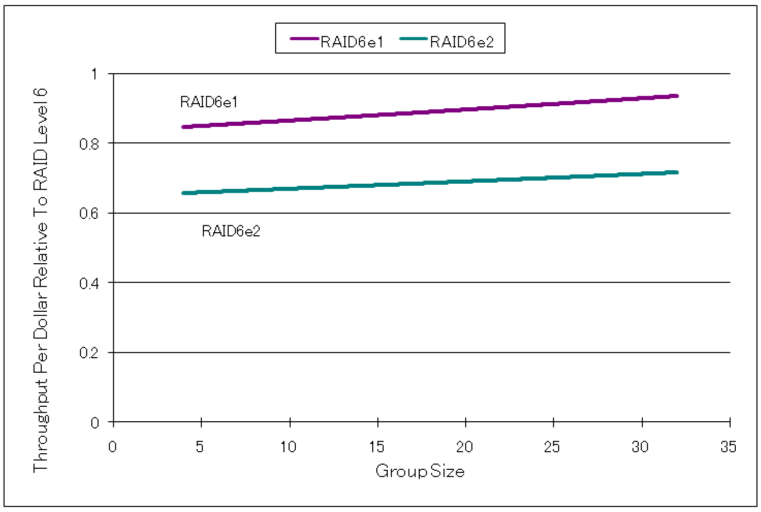


Fig. 3.12. *Large Write*

The performance of the storage built using VLSD was evaluated both during normal operation and in instances of failure. When a disk fails, performance drops for the Small Read as shown by the graph in Fig. 3.9.

A situation in which many small files are generated by the GCC build is actually created for this experiment, and is required to verify how the VLSD is affected. As shown in Fig. 4.1, before conducting the GCC build experiment, the experimental system must be constructed. Three Operating Systems, FreeBSD, Ubuntu Server, and Windows XP are installed on one machine for the experiment. For data communications, NFS is used between FreeBSD and Ubuntu, while NBD and VLSD are used between Ubuntu and Windows.

Below is the operating system specification.

- 1^{st} OS
  Windows XP Professional 64 bit
  CPU: AMD Athlone(tm) 64 X2 Dual Core 3800+
  RAM: 4GB
- 2^{nd} OS
  Ubuntu Server 7.10 64 bit using VMWare on 1st OS
  CPU: Same as 1st OS CPU
  RAM: 1GB

Table 3.4
MTTF and capacity of two level RAID

RAID level	MTTF[h]	Capacity Efficiency[%]
RAID11	4.25×10^{2463}	0.2
RAID15	1.7×10^{219}	4.3
RAID16	2.2×10^{328}	4.1
RAID51	4.19×10^{182}	4.3
RAID55	5×10^{15}	91.1
RAID56	1.1×10^{23}	86.8
RAID61	4.78×10^{272}	4.1
RAID65	7.8×10^{23}	86.8
RAID66	2.2×10^{35}	82.6

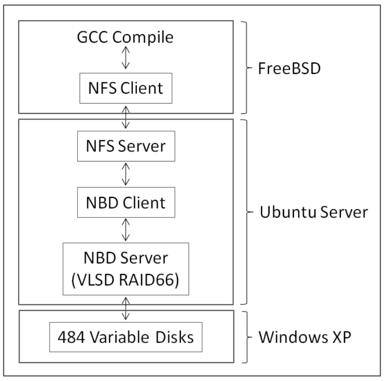


Fig. 4.1. *Experiment of GCC Build System*

- 3^{rd} OS
  FreeBSD 7.0 64 bit using VMWare on 1st OS
  CPU: Same as 1st OS CPU
  RAM: 1GB

To evaluate the performance of the VLSD, we setup three cases.

- In the first case (local disk), GCC is built on a local disk of FreeBSD.
- In the second case (NFS), GCC is built on a remote disk of the Ubuntu NFS server (local disk of Ubuntu).
- In the last case (VLSD), GCC is built on the remote disk of the Windows XP VLSD data grid (local disk of Windows).

As shown in Table 4.1, when performing the GCC build, the delay increases to 11% using NFS, while a 19% overhead was recorded when using VLSD. Consequently, the overhead for NFS is similar to the overhead for VLSD. It is thought that the network might be the cause of this delay.

When using RAID66, the storage system can still operate with up to nine failed disks depending on the fault position of the disk. This is the reason that up to eight disk faults were evaluated in this experiment. It is believed that the differences in the GCC build times with trouble-free operation, one disk failure, four disk failures and eight disk failures are not large because of the file caches in both the file system and operating system.

5. Conclusion. VLSD is a toolkit used to build mass storage using available PCs. Using VLSD a large-scale virtual disk of 70TB was constructed with RAID66 and its performance was evaluated. Although the results differed from the theoretical values, this variation can be considered to be no more than standard errors introduced by the empirical implementation. We also evaluated each disk class, namely FileDisk, VariableDisk, JBOD, PagedDisk, SingleDisk, RAID0, RAID1, RAID5 and RAID6.

To simulate a typical storage system in an educational environment that may contain many small files, we built GCC on various storage systems and evaluated the performance of each. The results show that the

Table 4.1
Required Time for GCC Build

Path	Compiled Time	Overhead
Local Disk	2:05:17	0%
NFS	2:18:43	11%
VLSD vith no disk fault	2:28:32	19%
VLSD vith 1 disk fault	2:29:02	19%
VLSD vith 4 disk fault	2:31:16	21%
VLSD vith 8 disk fault	2:32:16	22%

overhead for VLSD is the same as the overhead for NFS. This means that the storage system is usable even with many small files.

In this study, the estimation used only a single machine. However, the real system consists of around 500 machines. Performance still needs to be investigated when a network of two or more machines is used.

REFERENCES

[1] D. A. Patterson and R. L. Katz, *A Case for Redundant Arrays of Inexpensive Disks (RAID)*, ACM SIGMOD, 1988, pp. 109–116

[2] P. M. Chen, E. K. Lee, G. A. Gibson, H. Katz and D. A. Patterson, *RAID: High-Performance, Reliable Secondary Strage*, ACM Computing Surveys, Vol. 26, No.2, Jun. 1994, pp. 145–185

[3] E. Chai, M. Uehara, H. Mori and N. Sato, *Virtual Large-Scale Disk System for PC-Room*, LNCS 4658, Network-Based Information Systems, Sept. 2007, pp. 476–485

[4] E. Chai, M. Uehara and H. Mori, *Evaluating Performance and Fault Tolerance in a Virtual Large-Scale Disk*, In Proceedings of 22nd International Conference on Advaced Information Networking and Applications, Mar. 2008, pp. 926–933

[5] *Intelligent RAID6 Theory Overview and Information*, `http://download.intel.com/design/storage/papers/30812202.pdf`

[6] W. Verhelst,*Network Block Device*, `http://nbd.sourceforge.net`

[7] *Samba*, `http://www.samba.org`

[8] S. Shepler, et. al., *NFS version 4 Protocol,* `http://www.ietf.org/rfc/rfc3010.txt`

[9] *XFS*, `http://oss.sgi.com/projects/xfs/`

Edited by: Fatos Xhafa, Leonard Barolli
Received: September 30, 2008
Accepted: December 15, 2008

Scalable Computing: Practice and Experience
Volume 10, Number 1, pp. 99–109. http://www.scpe.org

ISSN 1895-1767
© 2009 SCPE

PERFORMANCE EVALUATION OF A WIRELESS SENSOR NETWORK FOR MOBILE AND STATIONARY EVENT CASES CONSIDERING ROUTING EFFICIENCY AND GOODPUT METRICS

TAO YANG[*], LEONARD BAROLLI[†], MAKOTO IKEDA[*], GIUSEPPE DE MARCO[‡], AND ARJAN DURRESI[§]

Abstract. Sensor networks are a sensing, computing and communication infrastructure that are able to observe and respond to phenomena in the natural environment and in our physical and cyber infrastructure. The sensors themselves can range from small passive microsensors to larger scale, controllable weather-sensing platforms. Presently, there are many research works for sensor networks. In our previous work, we built a simulation system for simulation the sensor networks. But, we considered that the event node is stationary in the observation field. However, in many applications the event node may move. For example, in an ecology environment the animals can move randomly. In this work, we investigate how the sensor network performs in the case when the event node moves. We carried out the simulations for lattice topology and TwoRayGround radio model considering AODV and DSR protocols. For the performance evaluation, we considered two metrics: routing efficiency and goodput and we compare the simulation results for two cases: when the event node is mobile and stationary. The simulation results have shown that the routing efficiency for the case of mobile event node is better than the stationary event node using AODV protocol. Also, the goodput for the mobile event node case does not change too much compared with the stationary event case using AODV, but the goodput is not good when the number of nodes is increased.

Key words: sensor network, mobile event, goodput, routing efficiency

1. Introduction. A Wireless Sensor Network (WSN) is a wireless network where the nodes are sensors, that is micro-devices with limited computation capacity and with on-board specific transducers. Recently, we witnessed a lot of research effort towards the optimization of standard communication paradigms for such networks. In fact, the traditional Wireless Network (WN) design has never paid attention to constraints such as the limited or scarce energy of nodes and their computational power. Another aspect which is different from traditional WN is the communication reliability and congestion control. In traditional wired nets, one reasonably supposes that communication paths are stable along the transmission instances. This fact permits to use the end-to-end approach to the design of reliable transport and application protocols. The TCP works well because of the stability of links. On the other hand, in WSN paths can change over time, because of time-varying characteristics of links and nodes reliability. These problems are important especially in a multi-hop scenario, where nodes accomplish also at the routing of other nodes' packets.

In this paper, as a case study, we study a particular application of WSNs for event-detection and tracking. The application is based on the assumption that WSNs present some degree of spatial redundancy. For instance, whenever an event happens, a certain number of sensor nodes, higher than that strictly required, will detect the event and transmit event data to the Monitoring Node (MN). Because of the spatial redundancy, we can tolerate some packet loss, as long as the required detection or event-reliability holds. This reliability can be formulated as the minimum number of packets required by the MN in order to re-construct the event field. An interesting consequence of this application is that we can use a simple connection-less transport of data. The rate of the transmitted data depends on many factors, as the bound on the signal distortion perceived at the MN. In the case of discrete event in the form of "event present" or "event not present", this scheme resembles the packet repetition scheme.

Recently, there are many research work for sensor networks [1, 2, 3, 4]. In our previous work [5], we implemented a simulation system for sensor networks. But, we considered that the event node is stationary in the observation field. However, in many applications the event node may move. For example, in an ecology environment the animals can move randomly. In this work, we investigate how the sensor network performs in the case when the event node moves. We carried out the simulations for lattice topology and TwoRayGround radio model considering Ad-hoc On-demand Distance Vector (AODV) and Dynamic Source Routing (DSR)

[*] Graduate School of Engineering Fukuoka Institute of Technology (FIT), 3-30-1 Wajiro-Higashi, Higashi-Ku, Fukuoka 811-0295, Japan (bd07003@bene.fit.ac.jp).

[†] Department of Information and Communication Engineering, Fukuoka Institute of Technology (FIT), 3-30-1 Wajiro-Higashi, Higashi-Ku, Fukuoka 811-0295, Japan (barolli@fit.ac.jp).

[‡] Department of Systems and Information Engineering, Toyota Technological Institute, 2-12-1 Tenpaku-Hisakata, Nagoya 468-8511, Japan (demarco@toyota-ti.ac.jp).

[§] Department of Computer and Information Science, Indiana University Purdue University at Indianapolis (IUPUI), 723 W. Michigan Street SL 280, Indianapolis, IN 46202, USA (durresi@cs.iupui.edu).

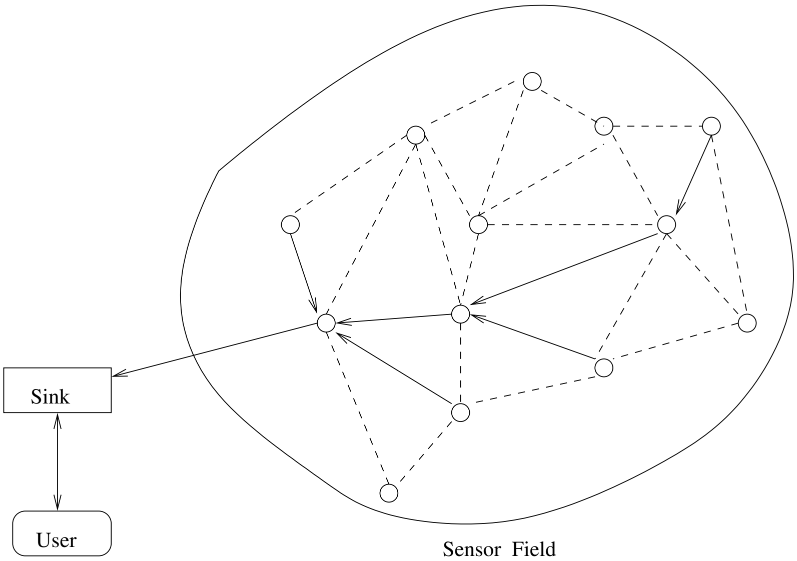


Fig. 2.1. *Physical architecture of WSN.*

protocols. For the performance evaluation, we considered two metrics: Routing Efficiency (RE) and goodput and we compare the simulation results for two cases: when the event node is mobile and stationary. The simulation results have shown that the routing efficiency for the case of mobile event node is better than the stationary event node using AODV protocol. Also, the goodput for the mobile event node case does not change too much compared with the stationary event case using AODV, but the goodput is not good when the number of nodes is increased.

The remainder of the paper is organized as follows. In Section 2, we explain the proposed network simulation model. In Section 3, we discuss the RE and goodput metrics. In Section 4, we show the simulation results. Conclusions of the paper are given in Section 5.

2. Proposed Network Simulation Model. In our WSNs, every node detects the physical phenomenon and sends back to the sink node data packets. In Fig. 2.1 is shown the physical architecture of WSN. We suppose that the sink node is more powerful than sensor nodes and it is always located a the borders of the service area. We analyze the performance of the network in a fixed time interval. This is the available time for the detection of the phenomenon and its value is application dependent. Proposed network simulation model is shown in Fig. 2.2.

In this paper, we consider that a mobile event is moving randomly in the WSNs field. In Fig. 2.3 is shown one pattern of movement event's path. We implemented a simulation system for WSNs considering moving event using ns-2. We evaluated the goodput and RE of AODV and DSR protocols using TwoRayGround propagation model and the lattice topology.

2.1. Topology. For the physical layout of the WSN, two types of deployment has been studied so far: the random and the lattice deployment. In the former, nodes are supposed to be uniformly distributed, while in the latter one nodes are vertexes of particular geometric shape, e.g. a square grid. as depicted in Fig. 2.4. In this work, we present results for the square grid topology only. In this case, in order to guarantee the connectedness of the network we should set the transmission range of every node to the step size, d, which is the minimum distance between two rows (or columns) of the grid. In fact, by this way the number of links that every node can establish (the node degree D) is 4. Nodes at the borders have $D = 2$.

2.2. Radio Model. There are three basic models for the propagation of the radio signals of sensor nodes: Free space model, TwoRayGround model and Shadowing model. In a simple deterministic model, the received power P_r at a certain distance d is the same along all directions in the plane[1]. For example, in the case of Line

[1] We are considering 2D networks, but similar results hold also in the more general case of 3D networks.

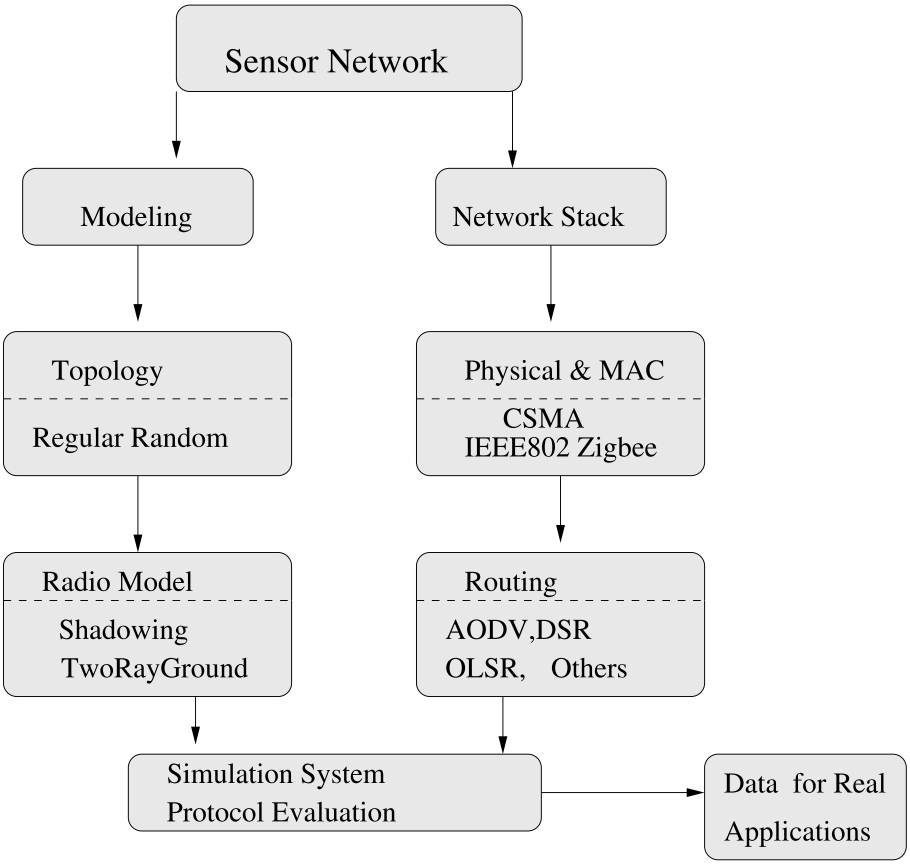


FIG. 2.2. *Network simulation model.*

Of Sight (LOS) propagation of the signal, the Friis formula predicts the received power as:

$$
\begin{aligned}
P_r(d) &= P_t - \beta \quad (\mathrm{dB}), \\
\beta &= 10 \log \left(\frac{(4\pi d)^2 L}{G_t G_r \lambda^2} \right)
\end{aligned}
\tag{2.1}
$$

where G_r and G_t are the antenna gains of the receiver and the transmitter, respectively. λ is the wavelength of the signal, L the insertion loss caused by feeding circuitry of the antenna, and β is the propagation pathloss. For omni-antennas, $G_R = G_t = 1$. The signal decay is then proportional to d^2.

In our simulation system, as a radio model we use TwoRayGround model. In this model the pathloss is:

$$
\beta = 10 \log \left(\frac{(4\pi d)^4 L}{G_t G_r h_t h_r \lambda^2} \right) \tag{2.2}
$$

where h_r and h_t are the receiver and transmitter antenna heights, respectively. The power decreases faster than Eq. (2.1) [6].

2.3. MAC protocol. We assume that the MAC protocol is the IEEE 802.11 standard. The receiver of every sensor node is supposed to receive correctly data bits if the received power exceeds the receiver threshold, γ. This threshold depends on the modulation scheme [2]. As reference, we select parameters values according to the features of a commercial device (MICA2 OEM). In particular, for this device, we found that for a central frequency of $f = 916$MHz and a data rate of 34KBaud, we have a threshold (or receiver sensitivity) $\gamma = -98$dBm [7]. The transmission range of sensor nodes should be at least d. However, in the case of a general

[2] Other MAC factors affect the reception process, for example the Carrier Sensing Threshold (CST) and Capture Threshold (CP) of IEEE.802.11 used in NS-2.

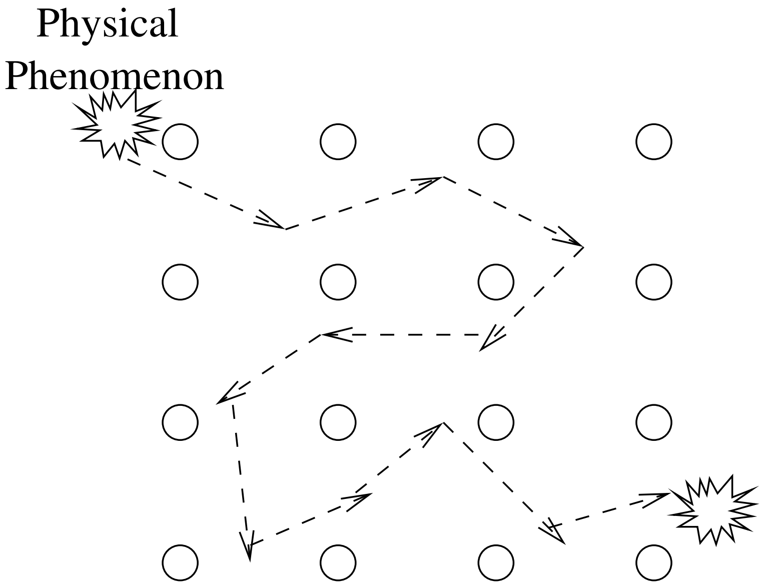


Fig. 2.3. *One pattern of movement event path.*

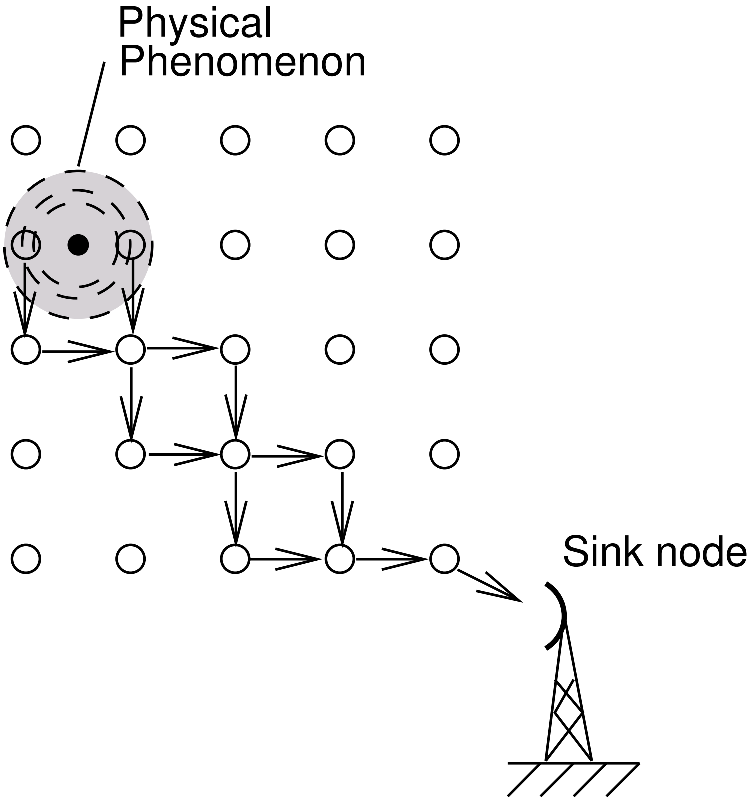


Fig. 2.4. *Lattice topology.*

radio model, we must take into account randomness of pathloss. The transmission power of each sensor is shown by the following equation:

$$P_t(d)|_{\mathrm{dB}} = \left[10\beta \log_{10}\frac{d}{d_0} + \gamma - \mathrm{erfc}^{-1}(2p)\sqrt{(2)}\sigma\right] + + 10\log_{10}\left(\frac{d_0}{K}\right)^2 , \tag{2.3}$$

where erfc^{-1}is the inverse of the standard error function.

2.4. Routing Protocols. We are aware of many proposals of routing protocols for ad-hoc networks. Here, we consider reactive protocols such as AODV and DSR [8].

The AODV is an improvement of DSDV to on-demand scheme. It minimize the broadcast packet by creating route only when needed. Every node in network should maintain route information table and participate in routing table exchange. When source node wants to send data to the destination node, it first initiates route discovery process. In this process, source node broadcasts Route Request (RREQ) packet to its neighbors. Neighbor nodes which receive RREQ forward the packet to its neighbor nodes. Neighbor nodes which receive RREQ forward the packet to its neighbors, and so on. This process continues until RREQ reach to the destination node. When the intermediate nodes receive RREQ, they record in their tables the address of neighbors, thereby establishing a reverse path. When the node which knows the path to destination or destination node itself receive RREQ, it sends back Route Reply (RREP) packet to source node. This RREP packet is transmitted by using reverse path. When the source node receives RREP packet, it can know the path to destination node and it stores the discovered path information in its route table. This is the end of route discovery process. Then, AODV performs route maintenance process. In route maintenance process, each node periodically transmits a Hello message to detect link breakage.

In DSR, the source node knows the entire path to the destination. And intermediate nodes don't need to maintain route information table for neighbor's address. This source routing information is stored in cache and updated when the node discover new route. DSR also consist of two mechanisms (Route Discovery and Route Maintenance processes). In route discovery process, DSR also uses RREQ and RREP as AODV. However in DSR, intermediate nodes don't need to maintain route table because hop-by-hop path information (reverse path information) is stored in RREQ header when RREQ is forwarded from intermediate nodes. When destination node which knows the path to destination receives this RREQ packet, it sends back RREP to source node using header stored in RREQ packet. When the source node receives RREP packet, it knows the entire path to destination node and store the discovered path in cache. Route maintenance process is accomplished through the use of Route Error (RRER) packets. RRER packets are generated when the link is broken due to mobility of nodes.

2.5. Event Detection and Transport. As event detection and transport, we use the data-centric model similar to [9], where the end-to-end reliability is transformed into a bounded signal distortion concept. In this model, after sensing an event, every sensor node sends sensed data towards the MN. The transport used is a UDP-like transport, i. e. there is not any guarantee on the delivery of the data. While this approach reduces the complexity of the transport protocol and well fit the energy and computational constraints of sensor nodes, the event-reliability can be guaranteed to some extent because of the spatial redundancy. The sensor node transmits data packets reporting the details of the detected event at a certain transmission rate [3]. The setting of this parameter, T_r, depends on several factors, as the quantization step of sensors, the type of phenomenon, and the desired level of distortion perceived at the MN. In paper [9], the authors used this T_r as a control parameter of the overall system. For example, if we refer to event-reliability as the minimum number of packets required at MN in order to reliably detect the event, then whenever the MN receives a number of packets less than the event-reliability, it can instruct sensor nodes to use a higher T_r. This instruction is piggy-backed in dedicated packets from the MN. This system can be considered as a control system, as shown in Fig. 2.5, with the target event-reliability as input variable and the actual event-reliability as output parameter. The target event-reliability is transformed into an initial T_r^0. The control loop has the output event-reliability as input, and on the basis of a particular non-linear function $f(\cdot)$, T_r is accordingly changed. We do not implement the entire control system, but only a simplified version of it. For instance, we vary T_r and observe the behavior of the system in terms of the mean number of received packets. In other words, we open the control loop and analyze the forward chain only.

[3] Note that in the case of discrete event, this scheme is a simple packet repetition scheme.

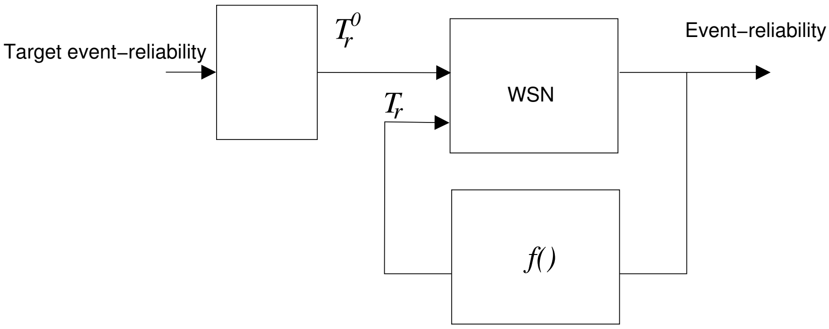


Fig. 2.5. *Representation of the transport based on the event-reliability.*

Table 4.1
Topology settings.

LATTICE	
Step(m)	$d = \frac{L}{\sqrt{N}-1}$
Service Area Size(m^2)	L^2 = (800x800)
Number of Nodes	$N \in \{12, 64, 100, 256\}$
Transmission Range(m)	$r_0 = d$

3. Goodput and Routing Efficiency. In this section, we introduce the performance evaluation metrics: goodput and RE. We consider that after a sensor node detects the physical phenomenon, it sends the packets to the sink node via a routing protocol. The ability for transmitting packets for different protocols is different. Also, the RE of a protocol is affected by many network parameters such as wireless transmission radio model, network topology, and transmission frequency [4]. In order to compare the performance of different protocols, we consider the same simulation environment. For our system, we used TwoRayGround radio model and the network topology is regular [10, 11].

The goodput is defined at the sink, and it is the received packet rate divided by the sent packets rate. Thus:

$$G(\tau) = \frac{N_r(\tau)}{N_s(\tau)} \tag{3.1}$$

where $N_r(\tau)$ is the number of received packet at the sink, and the $N_s(\tau)$ is the number of packets sent by sensor nodes which detected the phenomenon. Note that the event-reliability is defined as $G_R = \frac{N_r(\tau)}{R(\tau)}$, where R is the required number of packets or data in a time interval of τ seconds.

We defined the RE parameter as the ratio of sent packets from sensing node with sent packet by routing protocol. Thus:

$$RE(\tau) = \frac{N_{sent}(\tau)}{N_{routing}(\tau)} \tag{3.2}$$

where $N_{routing}(\tau)$ is the number of sent packets by routing protocol, and $N_{sent}(\tau)$ is the number of sent packets by sensor nodes which detect the phenomenon.

4. Simulation Results. In this section, we present the simulation results. We implemented the simulation system using ns-2 simulator, with the support of NRL libraries [12]. For each routing protocol, the sample results of Eq. (3.1) and Eq. (3.2) are computed over $20s$ simulation runs.

The settings of our lattice are shown in Table 4.1. The sensing range is assumed to be half of the transmission range. The radio model parameters are listed in Table 4.2.

In Fig. 4.1 is shown the average value of RE using TwoRayGround model and AODV in case of stationary event. The RE is an increasing function of T_r, because as T_r increases, the number of sent packet by sensing

TABLE 4.2
Radio model and system parameters.

RADIO MODEL PARAMETERS	
Path Loss Coefficient	$\alpha = 2.7$
Variance	$\sigma^2_{\text{dB}} = 16\text{dB}$
Carrier Frequency	916MHz
Antenna	Omni
Threshold (sensitivity)	$\gamma = -118\text{dB}$
OTHER PARAMETERS	
Reporting Frequency	$T_r = [0.1, 1000]$pps[1]
Interface Queue Size	50 packets
UDP Packet Size	100 bytes
Detection Interval τ	30s

[1] packet per seconds

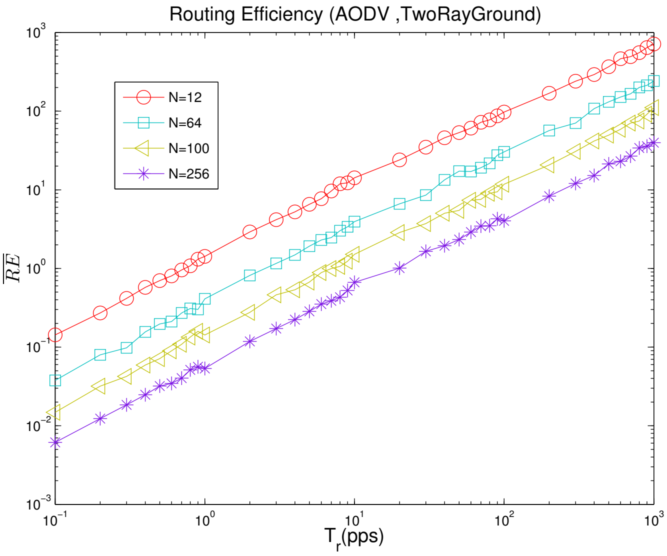


FIG. 4.1. *RE for stationary event (AODV).*

node is higher than the number of packets used by routing protocol. In case of AODV, the RE decreases with the increase of number of sensor nodes. It should be noted that when the number of sensor nodes is increased, the number of routes is increased, thus the searching time to find a route also is increased. When the number of nodes is 256, the RE of AODV is the worst in our simulation. The simulation results for the case of mobile event are shown in Fig. 4.2. The RE is stable and better than in case of stationary event, especially when the number of nodes is increased. When $N = 12$, the performance is almost the same for both cases. But, in the cases of $N = 64, N = 100, N = 256$, the performance is better for mobile event case.

In case of DSR, the simulation results are shown in Fig. 4.3 and Fig. 4.4. Comparing with AODV for the same time interval and the number of nodes, the RE of DSR is better than AODV. In the mobile environment, AODV sends the RREQ and when receives the reply from the sink it sends the packet to the sink. However, in the case of mobile event, when the transmission rate is lower the value of RE is not stable because of congestion situation. When the transmission rate is larger than 10pps, the RE becomes stable. However, the performance is still better than AODV. This is because the DSR invokes the local route repair using the alternative routes in route cache.

In following, we explain the results of goodput for the same environment. In Fig. 4.5 is shown the goodput for stationary event, while in Fig. 4.6 for mobile event in the case of AODV protocol. In both cases, when the

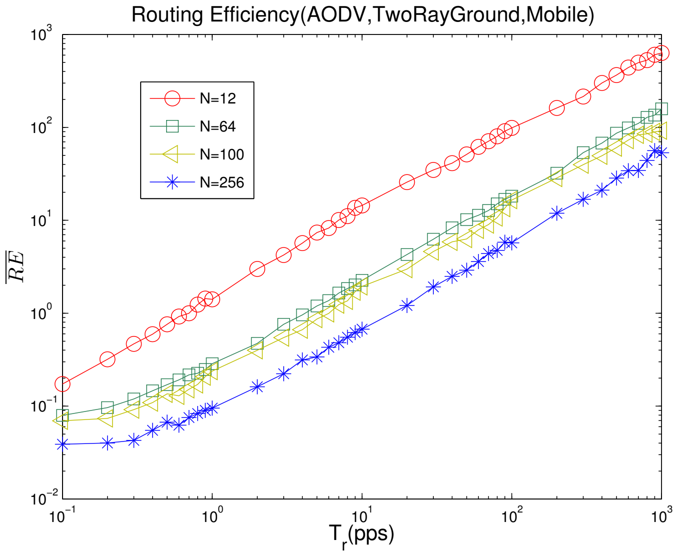


FIG. 4.2. *RE for mobile event (AODV).*

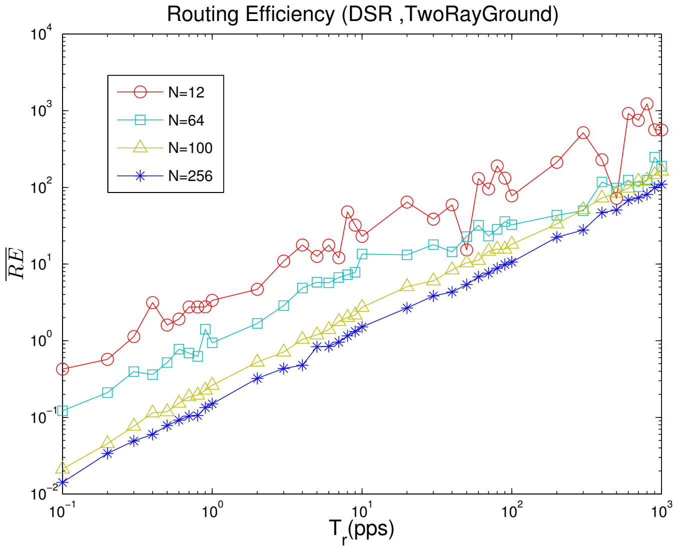


FIG. 4.3. *RE for stationary event (DSR).*

number of nodes is increased, the goodput is increased. However, the goodput of stationary event is better than mobile event, but the goodput of mobile event is more stable.

In case of DSR, as shown in the Fig. 4.7, the goodput of stationary event is almost the same with AODV (see Fig. 4.5). However, in case of mobile event (see Fig. 4.8), with the increase of number of nodes, the value of goodput is decreased much more compared with stationary event case. When the number of nodes is 256 the goodput is the worst and is not stable.

The goodput of AODV is better than DSR in case of mobile event. This is because when AODV performs route discovery, it uses the control messages RREQ and RREP. To control the broadcasts of RREQs, the source node uses an expanding ring search technique. In DSR, many RREQ packets are broadcasted to find the destination route. So, there are many packets loss [13, 14].

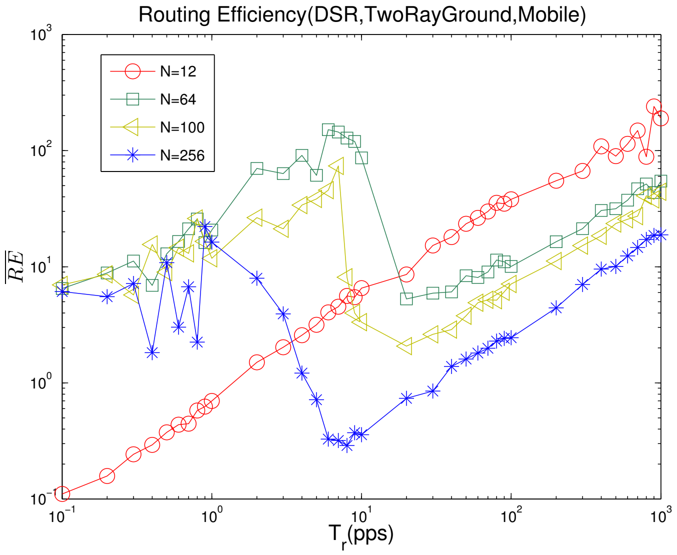


Fig. 4.4. *RE for mobile event (DSR).*

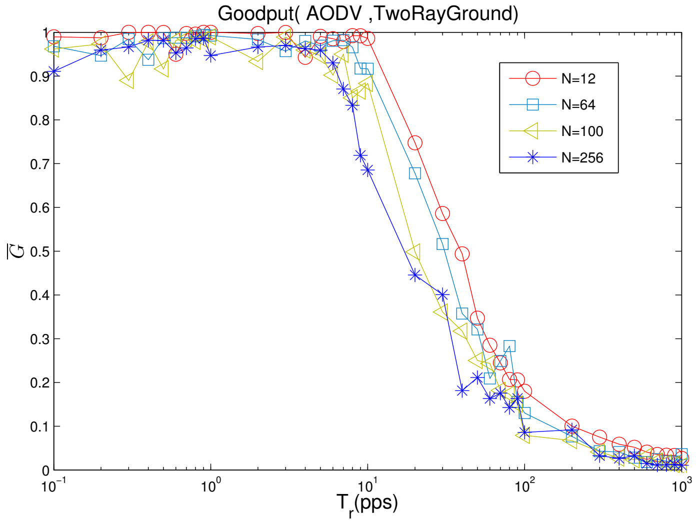


Fig. 4.5. *Goodput for stationary event (AODV).*

5. Conclusions. In this paper, we presented the implementation of a simulation system for WSNs using ns-2. We considered for simulations AODV and DSR protocols and evaluated the proposed system for two cases: stationary event and mobile event. For evaluation, we considered goodput and RE metrics.

From the simulation results, we conclude as follows.

- In case of AODV, the RE decreases with the increase of number of sensor nodes. When the number of nodes is 256, the RE of AODV is the worst.
- In the case of mobile event, for AODV protocol, the RE is stable and better than in case of stationary event, especially when the number of nodes is increased.
- Comparing with AODV for the same time interval and the number of nodes, the RE of DSR is better than AODV, but it is unstable.
- The goodput of AODV protocol for stationary event is better than mobile event, but the goodput of mobile event is more stable.

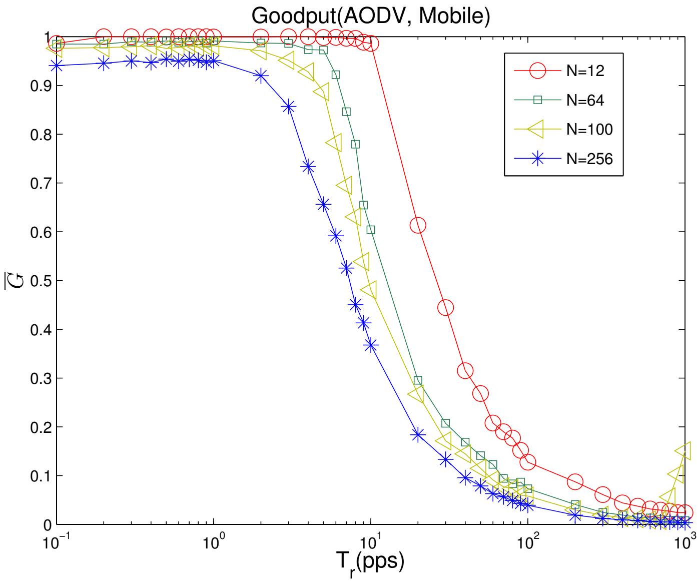


FIG. 4.6. *Goodput for mobile event (AODV).*

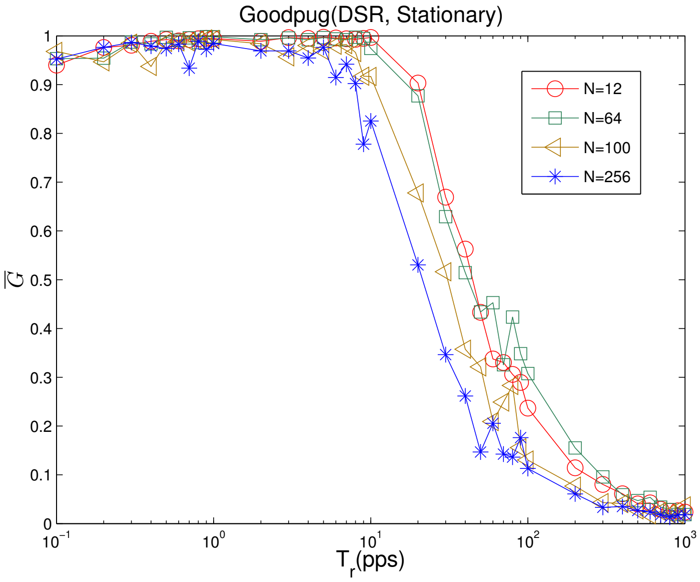


FIG. 4.7. *Goodput for stationary event (DSR).*

- In case of DSR, the goodput of stationary event is almost the same with AODV. However, in case of mobile event with the increase of number of nodes, the value of goodput is decreased much more compared with stationary event case. When the number of nodes is 256, the goodput is the worst and is not stable.
- The goodput of AODV is better than DSR in case of mobile event.

In the future, we would like to carry out more extensive simulations for many mobile events and other protocols. We also would like to consider the case of Shadowing radio model and the random topology.

Acknowledgments. The authors would like to thank International Communications Foundation (ICF) of Japan and Japanese Society for the Promotion of Science (JSPS) for supporting this work.

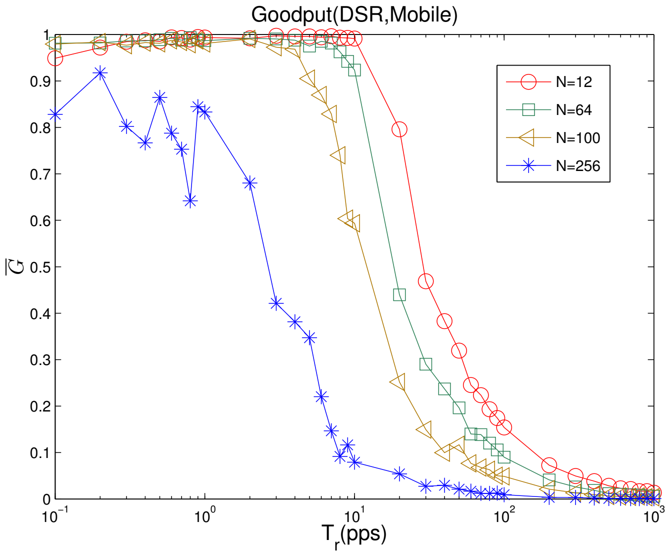


Fig. 4.8. *Goodput for mobile event (DSR).*

REFERENCES

[1] S. Giordano and C. Rosenberg, "Topics in Ad Hoc and Sensor Networks", IEEE Communication Magazine, Vol. 44, No. 4, pp. 97–97, 2006.

[2] J. N. Al-Karaki and A. E. Kamal, "Routing Techniques in Wireless Sensor Networks: A Survey", IEEE Wireless Communication, Vol. 11, No. 6, pp. 6–28, December 2004.

[3] G. W.-Allen, K. Lorincz, O. Marcillo, J. Johnson, M. Ruiz, J. Lees, "Deploying a Wireless Sensor Network on an Active Volcano", IEEE Internet Computing, Vol. 10, No. 2, pp. 18–25, March, 2006.

[4] O. Younis, S. Fahmy, "HEED: A Hybrid, Energy-efficient, Distributed Clustering Approach for Ad-hoc Sensor Networks", IEEE Transactions on Mobile Computing, Vol. 3, No. 4, pp. 366–379, 2004.

[5] T. Yang, G. De Marco, M. Ikeda, L. Barolli, "A Case Study of Event Detection in Lattice Wireless Sensor Network with Shadowing-Induced Radio Irregularities", Proc. of MoMM-2006, Yogyakarta, Indonesia, pp. 241–250, December 2006.

[6] T. S. Rappaport, "Wireless Communications", Prentice Hall PTR, 2001.

[7] Crossbow Technology, Inc., `htpp://www.xbow.com/`

[8] C. Perkins, "Ad Hoc Networksc", Addison-Wesley, 2001.

[9] Özgür B. Akan and I. F. Akyildiz, "Event-to-Sink Reliable Transport in Wireless Sensor Networks", IEEE/ACM Transactions on Networking, Vol. 13, No. 5, pp. 1003–1016, 2005.

[10] W. Ye, J. Heidemann, D. Estrin, "Medium Access Control with Coordinated Adaptive Sleeping for Wireless Sensor Networks", IEEE/ACM Transaction Networking, Vol. 12, No. 3, pp. 493–506, 2004.

[11] G. Zhou, T. He, S. Krishnamurthy, J. A. Stankovic, "Models and Solutions for Radio Irregularity in Wireless Sensor Networks", ACM Transaction on Sensors Network, Vol. 2, No. 2, pp. 221–262, 2006.

[12] I. Donward, "NRL's Sensor Network Extension to NS-2", `http://pf.itd.nrl.navy.mil/nrlsensorsim/` 2004.

[13] V. C. Gungor, M. C. Vuran, O. B. Akan, "On the Cross-layer Interactions Between Congestion and Contention in Wireless Sensor and Actor Networks", Ad Hoc Networks Journal (Elsevier), Vol. 5, No. 6, pp. 897–909, August 2007.

[14] M. Zuniga and B. Krishnamachari, "An Analysis of Unreliability and Asymmetry in Low-power Wireless Links", `http://www-scf.usc.edu/marcozun/` 2006.

Edited by: Fatos Xhafa, Leonard Barolli
Received: September 30, 2008
Accepted: December 15, 2008

Scalable Computing: Practice and Experience
Volume 10, Number 1, pp. 111–114. http://www.scpe.org

ISSN 1895-1767
© 2009 SCPE

SPECIAL ISSUE BOOK REVIEW

EDITED BY FATOS XHAFA AND LEONARD BAROLLI

Ad Hoc Networks: Technologies and Protocols
Editors: Prasant Mohapathra, Srikanth V. Krishnamurthy
Publisher: Springer-Verlag Telos
ISBN: 0387226893
Year: 2005
http://www.springer.com/engineering/signals/book/978-0-387-22689-7

1. Introduction. The book AD HOC NETWORKS: Technologies and Protocols is targeted to engineers, researchers, advanced level students in the areas of ad hoc wireless networks. Special attention is placed on each building block, as well as, design challenges that need to be addressed with care when designing ad hoc networks. The main topics are related with the issues with the medium access protocols (802.11), routing protocols, multicast communications, transport layer protocols, energy conservation, QoS issues, and security.

The book communicates to the reader well and keeps the focus on the essential concepts and principles of ad hoc networking. The authors keep the reader engaged and facilitate the understanding process via use of several applications. After the reader has finished the book, the reader will be able to design complex ad hoc networks, pose new research questions, or tackle the research questions provided on future works at the end of each chapter.

2. Book Structure and Blow-by-Blow Review. The book starts with an excellent introduction by Professor Mario Gerla. First, Professor Gerla presents the main characteristics of ad hoc networks, including mobility, multihopping, self-organization, energy conservation, scalability, and security. Second, the author describes the design challenges of ad hoc networks, including cross layer interactions, mobility, and scalability.

Third, the author provides several examples of the ad hoc networks. For example, Figure 2.1[1] shows an example of the opportunistic ad hoc network. The example illustrates one scenario of the urban grid, which represents vehicle communications in an urban environment. While the cars connect to the cellular system, the applications of the ad hoc networks will span from the need of within the car communication or with other cars on the road.

Lastly, the author provides an overview of the chapters in the book. The book topics include: Mac Layer Issues and Protocols, Routing Protocols, Transport Layer Protocols, Multicasting, QoS issues, Energy Conservation, Directional Antennas, and Security issues. We present here a summary of the rest of the chapters.

Chapter 2: Collision Avoidance Protocols in Ad Hoc Networks. The chapter addresses the issues of designing effective MAC protocols that regulate the access of the wireless nodes to the shared channel, under scarce channel bandwidth that constrains ad hoc networks. They address sender-initiated collision avoidance schemes and show that they are more efficient that the traditional carrier sense multiple access schemes. In additions, the authors reveal the fairness problem in IEEE 802.11 and how to address the problem by using a topology aware fair access scheme.

For example, the authors compare via simulation studies (Please refer to Figure 2.2[2]) the throughput of the collision avoidance vs. CSMA, under different parameters. The experiments showed that the throughput of the CSMA is lower than RTS/CTS protocols under different parameters.

Chapter 3: Routing in Ad Hoc Networks. The chapter classifies the routing protocols in four main groups and describes the design details. We summarize in Table 2.1 the protocol classification.

Specifically, it discusses the location assisted routing protocols, which are important in the urban grid environment. They illustrate the protocol building blocks by using as examples the LAR and DREAM protocols. They provide an in-depth discussion on the flooding mechanism, as well as, other important protocol features, including route request, route error, and route reply.

[1] Reproduced with author's permission
[2] Reproduced with author's permission

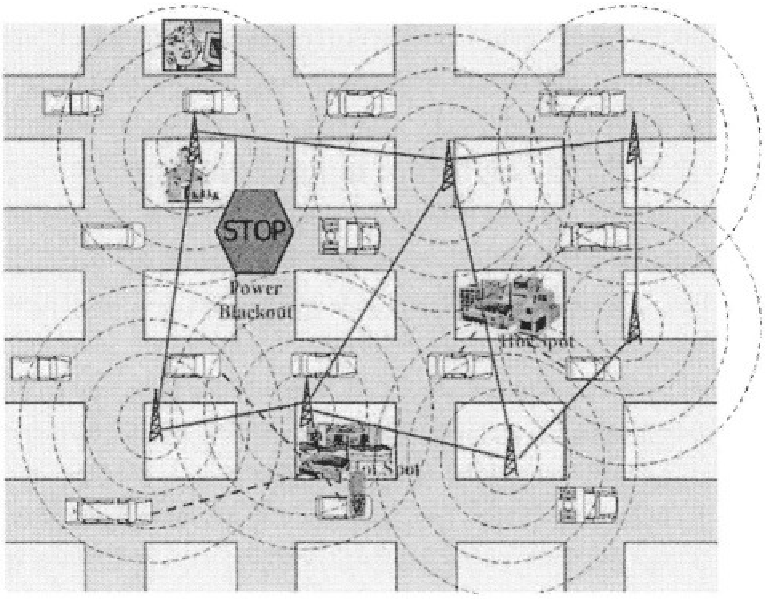


Fig. 2.1. *Opportunistic Ad Hoc Networks.*

Table 2.1
Routing Protocol Classification.

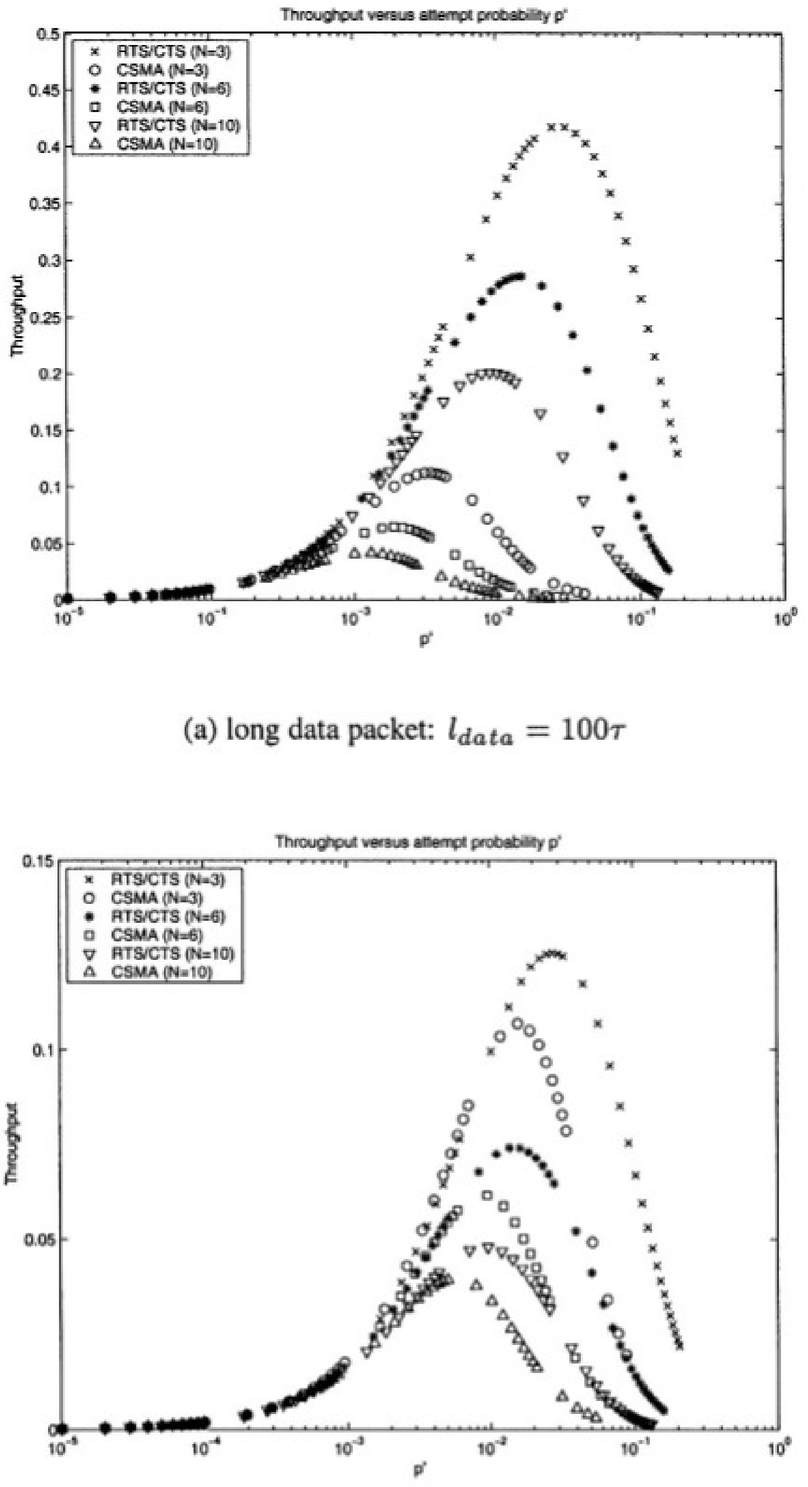
Classifications	Routing Protocols
Proactive	DSDV, OLSR,TBRPF
Reactive	DSR, AODV
Hybrid	ZRP
Location Assisted	GPSR, LAR, DREAM

Chapter 4: Multicasting in Ad Hoc Networks. This chapter provides a survey of the literature on the multicast protocols, which represent the property of broadcasting the data or video to all users that are part of the same mission. First, MAODV and ODMRP that are the most popular multicast protocols are described. Then, the authors, also, review the other protocols, i. e., MCEDAR, AMRoute, Geocast, and Gossip.

Chapter 5: Transport Layer Protocols in Ad Hoc Networks. This chapter addresses the main reasons on the performance degradation of ad hoc networks under TCP. For example, the focus is placed on packet loss, which can be caused due to congestion, jamming, or route breaks. However, the main issue is the inability to differentiate between the packet loss and congestion. Therefore, the authors present several feedback schemes to differentiate between the congestion and packet loss. For example, the Explicit Link Failure Notification is used as a feedback method to notify TCP of the link failures. In addition, other feedback methods are described in the chapter (ATRA and ATP).

Chapter 6: Energy Conservation. In the urban grid the energy conservation is not a major issue, however it is an important issue to be addressed in the applications of small hand held mobile wireless devices. The focus on this chapter is placed on the applications that involve sensors and pedestrian grids. The authors provide a survey of the energy conservation methods: power and topology control, energy routing, and coordinated sleep, and power save management. One example is using scheduling techniques for the powered up wireless devices in such a way that the interaction is most effective for a given recharge cycle.

Chapter 7: Use of Smart Antennas in Ad Hoc Networks. First, an overview of directional antennas and the reasons for using the directional antennas are provided. The directional antennas extend range, fold jamming attacks, and reduce the probability of detection. Second, it describes the smart antennas, which transmit simultaneously on multiple beams. Lastly, it shows the interaction between the antenna directionality, MAC, and the Routing protocols.



(a) long data packet: $l_{data} = 100\tau$

(b) short data packet: $l_{data} = 20\tau$

Fig. 2.2. *Throughput Comparison.*

Chapter 8: QoS Issues in Ad Hoc Networks. Given the bandwidth constraints and mobility in ad hoc networks, the QoS requirements of delay, latency, jitter, and packet loss need to be addressed with care. The authors first provide an overview of the guaranteed QoS in the wired environment and its differences in the wireless case. The author presents QoS techniques based on each layer. In the physical layer they address how ARF, RBAR,and OAR impacts QoS. Then, they move to the improvements on the MAC layer and demonstrate how the methods of PCF Schedule and ITC can be modified to meet QoS requirements. Lastly, they move to the QoS routing by ensuring Call acceptance control and/or service service negotiations.

Chapter 9: Security in mobile Ad Hoc Networks. Due to the open wireless medium, non-infrastructure nature that disables centralized certificate authority and key exchange, and the mobility of roaming nodes are some of the main reasons that make the security of ad hoc networks a challenging task. The authors provides an excellent summary of the potential attacks on the routing layer, i. e., dissemination of false routing information (via Route Request, Route Reply, Route Error), altering the path, or falsifying the sequence numbers.

The authors provide a list of examples that were collected as a result of the literature review:

- Impersonating another node to spoof messages
- Modifying Route Reply message to inject a false route
- Generating bogus Route Error to disrupt a working route
- Suppressing Route Error to mislead other
- Wormwhole attacks
- Rushing Attacks
- Sybil Attacks

In addition, it presents a MANET architecture with an Intrusion Detection System, which uses co-operative node methodologies and places agents at the monitoring nodes. Lastly, it talks about the passive attacks, i. e., position and privacy attacks.

3. Does the book keep its promises?. This is an excellently written book, which covers the major design issues in ad hoc networks. The use of examples makes the book easy to read, follow, and comprehend. The book is aimed for advanced networking people, thus would not be an easy read for a novice reader on the field. In addition, the book can serve as a reference material to the new wireless researchers to understand the current state of the field and lead them to the future research directions that still need to be addressed.

4. Personal reading experience. I have been working on the field of ad hoc networks for three years now and have come to realize that the book AD HOC NETWORKS: Technologies and Protocols has provided me with a solid background and knowledge of the field. The book has been a self-study guide and I felt that was being taught the field, without a teacher; however, given the in-depth treatment of each building block of ad hoc networks, the teacher was not necessary. This book covers and expands on the major design issues in ad hoc networks.

Arta Doci,
Colorado School of Mines,
Colorado, USA
E-mail: `adoci@mines.edu`

Scalable Computing: Practice and Experience
Volume 10, Number 1, pp. 115–130. http://www.scpe.org

ISSN 1895-1767

THE EDGE NODE FILE SYSTEM: A DISTRIBUTED FILE SYSTEM FOR HIGH PERFORMANCE COMPUTING

KOVENDHAN PONNAVAIKKO* AND JANAKIRAM D*

Abstract. The concept of using Internet edge nodes for High Performance Computing (HPC) applications has gained acceptance in recent times. Many of these HPC applications also have large I/O requirements. Consequently, an edge node file system that efficiently manages the large number of files involved can assist in improving application performance significantly. In this paper, we discuss the design of a Distributed File System (DFS) specialized for HPC applications that exploits the storage, computation and communication capabilities of Internet edge nodes and empowers users with the autonomy to flex file management strategies to suit their needs.

Key words: distributed file systems, high performance computing, peer-to-peer systems, scalability

1. Introduction.

Data requirements of High Performance Computing (HPC) applications have been continuously growing over the past few years and are expected to grow even more rapidly in the years to come [23]. There has also been a significant increase in the number of participants, i. e., data producers and consumers, and in their geographical spread. Experimental setups, deployments of sensors, simulators, etc. generate large amounts of data which researchers world over may have use for. I/O libraries that provide necessary abstractions to HPC application programmers require appropriate interfaces and mechanisms for handling the storage, access and transfer of such data. Efficiently sharing large numbers of files among a substantial number of widely distributed users and groups is another related requirement. These requirements create a need for a large scale Distributed File System (DFS) on the Internet edge nodes.

The importance of file management in HPC applications is exemplified by the High Energy Physics (HEP) Data Grid [19]. The project shows the need for collectively using the storage and computing capabilities of widely distributed resources[1] for a specific purpose (in this case, performing next generation HEP experiments). For the HEP Data Grids, there is a need to maintain file catalogs and information services to discover files, locate them (by mapping logical file names to physical addresses) and to provide access. Other examples include Biomedical Informatics Research Network (BIRN) [2], drug discovery on desktop Grids [11], etc.

Traditionally, the set of storage nodes in individual DFS installations comprises of stable and well connected server quality nodes belonging to a few institutional LANs (such as in universities and companies). Advances in hardware technology have made it possible for a much larger set of edge nodes to contribute significantly to the performance of large scale distributed systems. In addition to all the nodes connected to institutional LANs (not just server quality nodes), Internet edge nodes include desktops, laptops, etc. that connect to the Internet using cable, DSL, Wi-Fi, cellular networks, etc. (Fig. 1.1). Our earlier works Vishwa [34] and Virat [40] address issues involved in using edge nodes for providing processor and memory sharing services respectively. In this work, we propose to exploit the resources of Internet edge nodes to build a scalable distributed file system. The presence of a large number of nodes helps in improving I/O scalability, while also boosting the amount of available computational and storage resources. We will, henceforth, refer to our file system as the Edge Node File System (ENFS).

ENFS uses proximity based clustering of edge nodes for the efficient management of resources and balancing of load (storage, computational, query, etc.). A system-wide structured Peer-to-Peer (P2P) network, comprising of a few capable and reliable nodes from each cluster, assists in the efficient discovery and usage of resources across the entire system. Such a two layered platform helps the system to scale to a large number of nodes and also maintain high levels of performance.

Requirements of different HPC applications, in terms of the characteristics and locations of storage sites on which files are stored, vary widely. Access/sharing semantics and consistency needs of users and groups also differ to a great degree. It is, therefore, imperative to consider user-centric preferences while performing file management tasks such as data storage, access and migration. Instead of carrying out such tasks in a rigid manner, in our system, we empower users/applications with the autonomy to flex the system's file management strategies to suit their needs and, hence, improve application performance. For instance, we demonstrate a

*Distributed and Object Systems Lab, Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, India

[1] The terms resource and node will be used interchangeably in this paper.

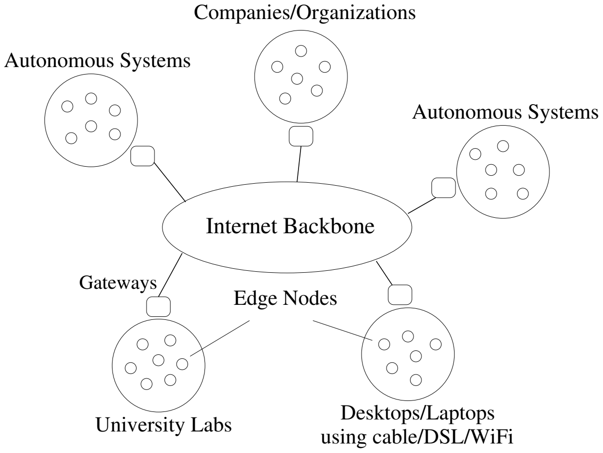


FIG. 1.1. *Internet Edge Nodes*

mechanism that allows a user to continue performing computations on a dataset, while it is being modified by another user.

Important design considerations in the context of a DFS for HPC using edge nodes are discussed in section 2. Section 3 presents the details of our system architecture and the working of ENFS. Section 4 deals with the various aspects of file management in ENFS. Performance studies are discussed in section 5. In section 6, we discuss the features of a few related systems which are relevant to our work. Conclusions and possible future work are presented in section 7.

2. ENFS: Design Considerations. Several features of a distributed file system must be analyzed carefully in order to design one with specific requirements [12]. We will discuss some of the important features in this section in the context of designing a large scale distributed file system that operates in a highly dynamic environment and is specialized for high performance computing applications.

2.1. System Organization. System organization refers to the assignment of specific roles to different system resources and defining how they interact with each other. Distributed file system organization varies from completely centralized [38] to completely P2P [24]. In a completely centralized setup, data and metadata associated with all the files are maintained at a server and clients query the server for all forms of file access. In such an approach, the centralized server severely affects the system's scalability. Making a set of nodes collectively function as data and metadata servers for a much larger number of clients helps in improving scalability to some extent [21]. Pure P2P file systems exist [24], in which every peer makes its own file placement decisions and uses P2P routing techniques to discover files. While such P2P systems scale well, the overheads involved in discovering and accessing files are high as well. Maintaining data and metadata consistency is also expensive in pure P2P systems.

2.1.1. Decoupling Data and Metadata. A recent approach to file system organization has been to decouple file data and metadata [17][44][39][3]. In these systems, only metadata is maintained by a set of servers. File contents are stored on a much larger set of nodes. The total amount of bandwidth that is available to clients to read and write data scales more or less linearly with the number of storage sites available. Such an organization results in a more scalable system.

2.1.2. Metadata Partitioning. A large percentage of file system operations are those on file metadata [36]. Thus, metadata access must be done as efficiently as possible. When a set of nodes function as metadata servers for the entire system, metadata partitioning mechanisms must be used to balance load among the servers. In [44], dynamic subtree partitioning [46] is used to partition the directory space among servers. In addition to namespace management, tasks such as monitoring the status of storage nodes, user/group management, etc. must also be uniformly distributed among a set of nodes.

2.1.3. Data Distribution. It is also necessary to distribute file contents among the storage nodes as evenly as possible to avoid bottlenecks in the system. Systems can either use a probabilistic method or a deterministic method to distribute data [12]. In the probabilistic method, hashing functions such as Secure Hash Algorithm [15] or Message Digest 5 [35] can be used to determine the sites at which files must be stored [24][29]. Since file placement is randomized, such systems will have little or no control over file locations and over other characteristics of storage sites. In the deterministic method, nodes use whatever knowledge they have of the system to place data. Systems with centralized servers that have global knowledge can easily balance storage load. However, such systems do not scale.

2.1.4. Fault Tolerance. While the abundance of edge nodes helps in improving system performance, higher levels of dynamism in resource characteristics and availability amplify the need for efficient monitoring and fault tolerance mechanisms. Replication of data is a widely used mechanism to tolerate system component failures. A file system employing edge nodes must be adept in replicating data on appropriate resources. Failure detection and migration of data must also be suitably handled. When multiple copies exist, file systems are also responsible for the consistent maintenance of data. Sections 3 and 4 discuss how ENFS addresses the above mentioned issues.

2.2. File Management. The primary function of a file system is to manage the placement and access of files. Due to large data set sizes, recent HPC approaches attempt to schedule computations on resources which contain the required data [32][43], thus reducing the amount of data movement. Therefore, placement of files at appropriate storage sites can significantly improve the performance of applications using the files.

2.2.1. File Placement. High performance computing applications are diverse in terms of hardware and software requirements. Applications perform best when the data that they operate on are stored in resources that suit their purpose best. Some of the characteristics that differentiate applications from each other are computational intensity [28], I/O throughputs [30], inter-task communications, memory requirements, etc. For example, in [4], the authors study the scientific applications GYRO (fusion simulation) [16] and POP (climate modeling) [22]. Various characteristics of the two applications are compared and the authors attempt to identify system architectures that will be most suitable for the execution of the two applications. Using similar application profiling techniques or based on the intuitive understanding of application characteristics, users may be able to specify preferences for the kind of resources on which to store their files.

Users may also be able to specify location preferences for the sites at which their files need to be stored. Based on expected access patterns, users may want files to be stored at or in the proximity of certain nodes or clusters. Certain files may most of the times be used by applications along with a few other files. Such files are best placed in the proximity of the associated files in order to reduce the amount of data transfer during application execution.

2.2.2. File Access. Moreover, access semantics preferred by different applications can vary widely. In a system where mutable data is shared among a large number of users[2], the mechanisms used to manage concurrent access to files can have a huge bearing on the performance of applications requiring those files. A rigid set of policies may not be suitable in such a system. Based on users' access requirements and the current state of file access, the DFS must be able to apply different kinds of access semantics so as to improve overall system performance. Section 4 discusses how file management issues are addressed in ENFS.

2.3. Security and Privacy. Several file systems adopt cryptographic techniques to encrypt their data before storing [24][3]. Doing so ensures privacy since users can now access file contents only if they have appropriate keys. In a few systems, content hashes are stored along with the metadata in order to verify the integrity of retrieved files. There are several security issues associated with large scale distributed systems. Standard practices such as the usage of Kerberos authentication protocols for the servers and clients to authenticate each other and establish secure sessions, using Access Control Lists (ACLs) to restrict the usage of files by different users, etc. are popularly used. While we have not specifically addressed any of these privacy/security issues, we have ensured that the above mentioned techniques can be easily incorporated into ENFS.

3. Architecture of the Edge Node File System. Over the last few years, decoupling metadata operations from read/write operations on files has been established as a strategy that can significantly improve a system's scalability and performance. For extremely large distributed systems (Internet scale) handling billions or trillions of files, a huge number of Metadata Servers (MDS) will be required to handle the load without

[2] Depending on the context, in certain places, the term user may actually refer to the user's application.

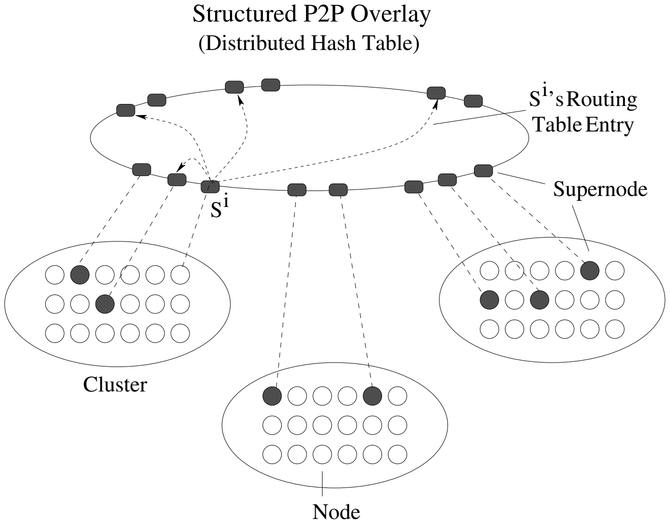


FIG. 3.1. *Two Layered Platform*

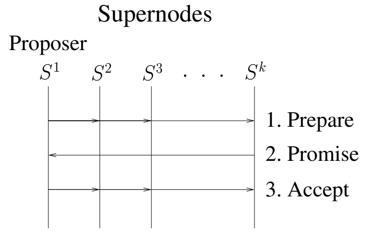
effecting any performance degradation. In contrast to many existing systems, in ENFS, we allow non-dedicated edge nodes to assume the responsibilities of a MDS. Such an approach creates a large pool of servers from which the required number can be chosen. Appropriate measures are put in place to handle the consequent issues of failures and load variations.

Metadata (such as file system namespace) has to be partitioned among the MDSs to ensure balanced query loads. Static partitioning of the directory hierarchy is useful in the sense that nodes can easily identify responsible MDSs. However, changes in demand for different portions of the namespace can result in unbalanced loads. Partitioning based on hashing file names (such as in [5]) can help in keeping the load much more balanced. However, such an approach cannot retain hierarchical locality of the files. Moreover, $O(logN)$ hops may be required to locate the responsible MDS, where N is the number of MDSs in the system. In [46], the issues of changing workloads and retention of locality are attempted to be solved using a dynamically adapting subtree based partitioning strategy. In such a hierarchy traversal system, locating files far down in the hierarchy may require several hops between MDSs. The time taken by a node to identify the MDS responsible for a particular file/directory has considerable bearing on the system performance. In ENFS, we employ a hybrid of hierarchical and hashing based strategies and attempt to minimize MDS location and access times.

3.1. Two Layered System Model. Based on proximity, nodes are grouped into large non-overlapping clusters (order of 10^3 nodes per cluster) by system administrators (Fig. 3.1). New clusters can be formed at any time and nodes can be added to and removed from clusters at any time. A lot of work has been done on network distance measurement [13], topology discovery [18][6] and proximity based node clustering [48][27]. Moreover, statistical data about the number of files, file sizes and generation rates, the number of participating nodes and their characteristics, the number of users and their file access patterns, etc. can be used to determine optimal cluster sizes. In this paper, we do not address issues related to autonomous cluster formation and maintenance for the sake of simplicity.

A few reliable and capable edge nodes from each cluster are chosen to be the metadata servers for that cluster. In addition to maintaining file metadata, information about the nodes within the cluster are also maintained at these servers. We will call these servers *Supernodes*. Supernodes share the load of gathering information about cluster resources, managing users and groups, placing data, maintaining file metadata, scheduling computations, responding to queries, etc. Supernodes from all the clusters form a single system-wide structured P2P overlay network or Distributed Hash Table (DHT) [41][37]. The overlay enables nodes of a cluster to discover supernodes (of other clusters) which are responsible for specific portions of the file namespace. The structured overlay also helps in the efficient discovery of resources with specific characteristics in the entire system.

3.2. Load Balancing. In order to enable efficient distribution of load among supernodes in accordance to their capabilities, a technique similar to the usage of virtual servers in Chord [33] is adopted. In Chord, every



1. Prepare: includes a unique proposal number along with the proposal value
2. Promise: may include an alternate value for the proposal
3. Accept: specifies the value that has been finalized

Fig. 3.2. *Paxos Algorithm: Messages and Arguments*

node in the overlay represents not one but a bunch of identifiers from the identifier space. When the query load on a particular node increases, the responsibility for a few of the virtual identifiers is transferred from the heavily loaded node to a lightly loaded node. In ENFS, we distribute a large virtual identifier space among the supernodes belonging to a cluster, based on their capabilities. When the load on a supernode increases, the responsibility for a portion of the virtual identifier space can be transferred from the heavily loaded supernode to a lightly loaded one. We use the Paxos algorithm [25] (discussed later in this section) to unambiguously rearrange load.

The mapping between the virtual identifier space and the physical address of supernodes is made known to the entire system using the structured layer. A file containing the association is stored in the structured layer with the cluster identifier as the key. We will refer to this file as the *supernodes_map*. Supernodes, primarily, manage entities such as nodes, users, groups and files belonging to their cluster. These entities determine the virtual identifier with which they must associate themselves by performing a standard operation on their unique identifiers (IP address, file path, etc.). For instance, let the maximum value of the supernode virtual identifier be V_{max}. The virtual identifier corresponding to an entity with identifier I can be

$(SHA(I)/SHA_{max}) * V_{max}$

where $SHA(I)$ is the digest obtained by applying the SHA hash function on the identifier and SHA_{max} is the maximum possible value of the 160-bit digest. Entities can then use the *supernodes_map* file to determine the physical address of the supernode responsible for that virtual identifier. For an entity with I as its identifier, the responsible supernode is denoted by S_I. When a node wants to make a query about I or when I wants to perform some file system operation, S_I is contacted.

The load at a supernode refers to the amount of computational power spent on performing different file system operations, including responding to client queries. Each supernode specifies a value for the maximum allowed CPU utilization percentage (measured over a time interval) for file system operations. Since file access frequencies and the amount of computational power required for different file system operations vary, loads on the supernodes keep changing. Whenever the actual load exceeds the limit imposed, a supernode is said to be overloaded. Supernodes periodically send their load information to the other supernodes in the cluster. In addition to the sharing of load information, the message also helps in letting other supernodes know about the source's liveness. When a supernode S^i detects the failure of another supernode S^j, it informs the other supernodes in the cluster about the failure[3]. S^i also initiates the process of replacing S^j with a new supernode. A supernode that is overloaded can propose an adjustment of the virtual identifier space to transfer some of its load to lightly loaded supernodes. When all supernodes in a cluster are heavily loaded, an additional supernode can be added to the existing set. On the other hand, when all the supernodes in a cluster are lightly loaded, one of the supernodes can be removed. A minimum number of supernodes, k_r (discussed in section 3.3), however, must always be available within a cluster.

As mentioned earlier, the Paxos algorithm (Fig. 3.2) is used by supernodes to reach an agreement on the addition/removal of a supernode and on the adjustment of the virtual identifier space. In the first phase of the Paxos algorithm, the proposer sends a *prepare* message to the acceptors and the acceptors may or may not respond with *promise* messages. In the second phase, if the proposer receives *promise* messages from a majority

[3] While S^i denotes the i^{th} supernode in the *supernodes_map* file, S_I denotes the supernode associated with entity I.

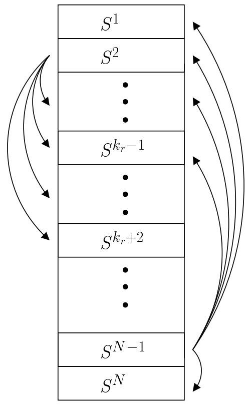


Fig. 3.3. *Supernode Metadata Replication*

of the acceptors, it sends an *accept* message with the decided value to the acceptors. If enough *promise* messages are not received, the proposal fails. Using Paxos helps in achieving consensus among a set of supernodes which are prone to failures. Changes in the set of supernodes and in the mapping of virtual identifiers to supernodes are immediately propagated to the *supernodes_map* file in the structured layer. The file includes a version number which is incremented after every change. The changes are also publicized to all the nodes within the cluster.

3.3. Fault Tolerance. In order to handle failures, the metadata stored in a supernode is replicated in a constant number (k_r) of other supernodes. The number k_r must be selected in such a way that the simultaneous failure of $k_r/2$ supernodes within a cluster is highly improbable. A supernode, S^n, that updates some metadata has to multicast the changes to all its replicas, so that the replicas are always in a state similar to that of S^n. Let there be N supernodes in a cluster and let $\{S^1, S^2, \ldots, S^N\}$ be the order in which the supernodes are listed in the *supernodes_map* file. Then, supernode S^i replicates its metadata in the nodes

$\{S^k, k = (i + j - 1) \bmod N \mid 1 \leq j \leq k_r\}$.

We will refer to this set of supernodes as S^i's replica set. In figure 3.3, all the supernodes pointed at by the arrows originating from supernode S^i constitute its replica set. When a querying resource detects that a supernode is not reachable, it queries the next-in-line supernode in the failed supernode's replica set. Whenever the responsibility for a virtual identifier is transferred from S^i to S^j, appropriate metadata transfer must happen. The metadata of all the resources that map to that virtual identifier must be transferred from S^i to S^j and to S^j's replica set.

Every node, when it becomes a supernode, identifies another node which is capable of being made the supernode, and maintains it as a *spare* supernode. The spare can be made a supernode in case of failures or load surges. Supernodes lazily keep updating metadata information in the spare nodes. This way, the amount of metadata transfer required for bringing up a new supernode is significantly reduced. When a supernode fails, the spare of the failed supernode or that of one of its replica set supernodes can be used as a replacement. Similarly, one of the better prepared spares, i. e., spares with little metadata transfer pending, can be used while adding additional supernodes to a cluster.

3.4. Metadata Consistency. Periodically, a supernode, S^i, sends the collected status information of all the resources it is responsible for to all the other supernodes in the cluster. We do not attempt to achieve sequential consistency among supernodes for such non-critical information because it can be expensive. However, for critical information such as file metadata, S^i and its replica set must always be in a synchronized state, i. e., sequential consistency semantics need to be observed. Otherwise, if S^i performs some file operation, such as providing a write lock to a user, and fails before letting its replica set know about the lock, the available metadata in the replica set becomes inconsistent with the actual state. For similar reasons, user/group metadata must also be maintained consistently among supernodes and their corresponding replica sets.

We adopt the Paxos agreement protocol for the consistent maintenance of critical metadata among S^i and its replica set. S^i sends the proposal to its replica set, waits for a sufficient number of responses, performs the

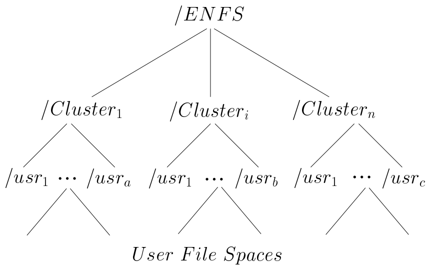


FIG. 3.4. *Hierarchical File System Namespace*

proposed operation and informs the replica set about the completion of the operation. Even if S^i fails before sending the completion message, its replica set can verify if the proposed operation was performed to completion or not. This way, a total order is maintained among the updates made to such metadata. Since the replica set size is a constant, the overhead of maintaining consistency does not increase with an increase in the number of supernodes in a cluster.

3.5. Namespace Management. We employ a single hierarchical file system namespace with the root named as $/ENFS$ (Fig. 3.4). Each filename, including the path, is therefore unique. The first level of namespace partitioning is achieved by making the supernodes of a cluster ($CId1$) operate as metadata servers for all the files and directories belonging to their cluster, i. e., all files and directories having a prefix of $/ENFS/CId1/$. Every user/group in ENFS must belong to one of the clusters. In order to be unique system-wide, user identifiers can be of the form $ClusterId/UserId$ and user home directories can be of the form $/ENFS/ClusterId/UserId/$. Distribution of responsibilities for files and directories among the supernodes in a cluster is done as is discussed in section 3.2. Since filenames including paths are hashed to identify responsible supernodes, metadata of different files in the same directory may be managed by different supernodes. In order to support POSIX directory access semantics, whenever a file or subdirectory is created, updated or deleted, the metadata of the parent directory must also be updated. Symbolic links are supported in ENFS by including the entire name of the original file or directory in the link's metadata. All operations on links are forwarded to the supernodes that manage the linked file's metadata.

Users requiring access to their own files and to the other files belonging to their cluster are served metadata by supernodes of their own cluster. Since cluster formation is based on the proximity of nodes, network latency between clients and metadata servers for such file accesses is small. In order to determine the metadata server (supernode) responsible for any other file, clients will need the current *supernodes_map* file of the cluster to which the required file belongs. This may require $O(logN)$ hops, where N is the size of the structured overlay. This cost is, however, amortized over future accesses to files belonging to the same clusters. Supernodes of a cluster prefetch and maintain the *supernodes_map* files of all clusters, whose files have been recently accessed by their users, thus thwarting the need to query the structured layer for every single access.

Though hashing helps in balancing query loads to a large extent, sudden increases in demand for individual files, referred to as flash crowds, can still overload the corresponding supernodes. In ENFS, when a supernode, S^i, notices an increase in the number of queries for a particular file, the cluster's *supernodes_map* file is immediately updated with a note suggesting that queries for the popular file must be forwarded through the client's supernode. Supernodes of other clusters that receive such queries, then, fetch and cache the popular file's metadata locally. These supernodes respond to future queries themselves, until the note is removed from the *supernodes_map* file of S^i's cluster (when demand for the file falls). The distribution of the responsibility of serving the metadata of popular files is shown in figure 3.5. Flash crowds can be handled in this manner just for read-only accesses. When the requests from flash crowds are largely update requests, performance is likely to get affected.

Network partitioning within a cluster is taken care of by the active replication and migration of metadata and data across the various supernodes and storage nodes of a cluster. When a small portion of the cluster becomes unreachable, it is unlikely that any data/metadata will become inaccessible.

3.6. Resource Monitoring. Supernodes of a cluster are responsible for monitoring the status of cluster resources. Nodes periodically send their status information to the appropriate supernodes. In addition to

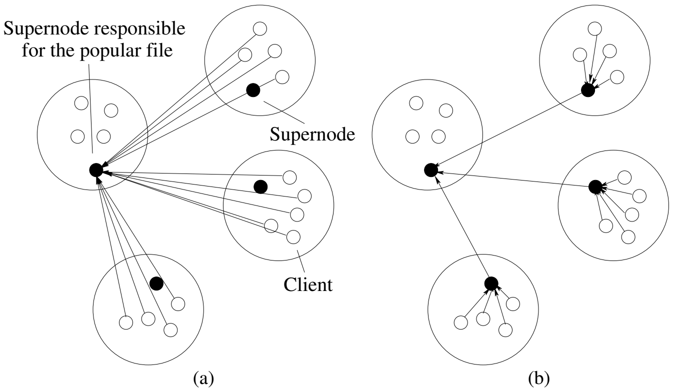


FIG. 3.5. *Handling Flash Crowds (a) Occurrence of flash crowds (b) Distribution of responsibility*

the current values of vital node characteristics, the status information also includes the version number of the *supernodes_map* file that is known to the nodes. If there is mismatch between the actual version number and the received number, the supernode sends the node the current file. This is done to ensure that the nodes in a cluster have current knowledge about the supernodes responsible for that cluster. When a supernode, S^i, does not receive the status information from a node for a certain period of time, it checks if that node has failed. If the node has failed, S^i periodically attempts to ping the failed node. When the node becomes active again, S^i sends the node the current *supernodes_map* file. The frequency and duration of node failures are used to quantify the reliability of a node. This measure is maintained at the supernodes along with other node characteristics. Node status information and failures are immediately made known to the supernodes in S^i's replica set.

3.7. Resource Discovery. The two layered system model of ENFS enables the efficient discovery of system-wide resources with specific characteristics. Since supernodes periodically broadcast status information about the nodes they are responsible for, to the other supernodes in their cluster, all the supernodes of a cluster maintain more or less up-to-date information about all the nodes in their cluster. Supernodes also maintain information about the clusters closest to them in the system. When a supernode is not able to locate enough resources with specific characteristics (to handle a user request) within its cluster, it can send appropriate queries to the supernodes of nearby clusters. However, in certain scenarios, a supernode may not be able to locate enough suitable resources in any of its nearby clusters. Flooding the system with queries loads the network and also does not provide any guarantees. ENFS exploits the reach of the structured overlay to efficiently discover resources located anywhere in the system.

A node is usually characterized by different capabilities or features such as processing speed, memory size, storage capacity, uplink/downlink bandwidths, system architecture, etc. Each characteristic c_i can be quantified by values belonging to m_i categories. For example, memory sizes can be categorized as being less than mr_1 MBs, between mr_1 and mr_2 MBs, ..., greater than mr_m MBs, etc. Standard system-wide policies can be used for the discretization of the values of the different characteristics. For every category of each characteristic, a file is maintained in the structured layer that includes the identifiers of all the clusters that have a large number of resources of that capability available for use. The file also includes the approximate number of resources in each cluster. JuxMem [5] uses a similar strategy for the available memory alone.

The *characteristic_category* file in the structured layer must be updated by supernodes only when their clusters have a large number of resources (with the corresponding capability) on a regular basis. A pessimistic estimate must be used for the number of resources listed as available. This is important because every small variation in the number of resources available in a cluster should not necessitate a change to the file in the structured layer. Using such an approach, the number of clusters to which queries must be sent for resource discovery is significantly reduced. Discovering resources with a combination of characteristics is also possible by making individual queries for each characteristic and then joining the results.

4. File Management. A useful, yet ignored feature in distributed file systems, is that of allowing users to have some level of control over choosing appropriate resources for the storage of their files. Most of the times, users generating data files, to be used by high performance computing applications, have some knowledge about the characteristics and requirements of the applications that are going to use their data. Information about the set of users who are going to perform computations on the files may also be sometimes available. Data placement based on such knowledge can result in significant improvements in application performance.

4.1. File Placement.

4.1.1. User Preferences. As discussed in section 2.2.1, high performance computing applications differ widely in their characteristics, and hence, resource requirements. Resource requirements can be specified in terms of the processing speed, network bandwidth, memory size, storage capacity, architecture, etc. Location preferences, on the other hand, can be specified based on expected access patterns. Users may want files to be stored in the proximity of specific nodes or clusters. For example, if certain files are expected to be used by the members of a particular group, the files are best placed in the proximity of the nodes frequented by those group members. Certain files may most of the times be used by applications together with a few other files. Such files are best placed in the proximity of the associated files in order to reduce the amount of data transfer required during the execution of the application.

To register preferences about resource characteristics, users must specify values for the different characteristics that storage nodes are expected to possess. The preferred characteristics must be specified as a prioritized list. To indicate location preferences, users can specify the set of users, groups, clusters, files or directories in the proximity of which their files must be placed. As in the previous case, location preferences must also be specified in the order of favor. Replication of user data is necessary to distribute the load on storage sites, handle storage site failures and also to enable parallel access. Users can explicitly specify the required levels of replication for their files based on expected demands.

The following shell command shows how a user can load a file into ENFS:

```
$ enfs put [options] local_filename enfs_filename
```

User preferences can be specified using command line options, as is done in POSIX commands. For example an option such as –cpu=5 can be used to specify that the file must be stored on nodes with processing speeds greater than 5 GHz. It is not practical to specify several options for each file. The required options can be set at the level of a directory and all the files created in that directory can inherit the set options. On the other hand, users can create combinations of options that they use often (locally on their nodes) and just specify the name of the combination while loading files. The DFS component on the node can replace the name of the combination with the entire set of options.

4.1.2. Block Management. In ENFS, files are split into data blocks. As much as possible, the blocks are stored on different nodes. Among other benefits, this provides for parallel access and fault tolerance. The size of a block can be configured by the user based on application requirements. By default, the system sets the block size in accordance to the file size. ENFS uses the native file system present in a node's Operating System (OS) for the storage of data blocks as files in the local storage devices. The local file names are stored in the ENFS file's metadata. Block storage in the local file system and block transfers between nodes happen as a binary stream of data. This way, differences in the file storage formats of different file systems cannot corrupt the contents of data blocks. The network subsystem and other hardware interfaces of the OS are used for purposes such as communication, monitoring, etc.

4.1.3. Placement Strategy. Each supernode maintains a database with entries for all the resources in its cluster. Such a database assists in quickly identifying resources with required characteristics. Resources from other clusters can be discovered as discussed in section 3.7. Supernodes keep track of the set of nodes that each user frequents. Aggregating this information from all the members of a group, supernodes can associate a set of nodes with each group. From the metadata of files belonging to a directory, supernodes can determine the set of nodes that store most of the data belonging to files in that directory. All such information is gathered by supernodes lazily (as low priority tasks) and is used to make locality based placement decisions.

When users do not specify any preference for resource characteristics, supernodes select the set of storage nodes considering just the available storage space. When no location preference is indicated, supernodes try to store all the replicas within the same cluster and in nearby clusters. While placing data blocks, supernodes also take into consideration the reliability of nodes. Since files have to be available at all times, at least a few of the replicas must be stored on nodes with high reliability, while the other replicas can be stored on nodes with lesser reliability. At a high level, our file placement algorithm performs the steps shown in figure 4.1.

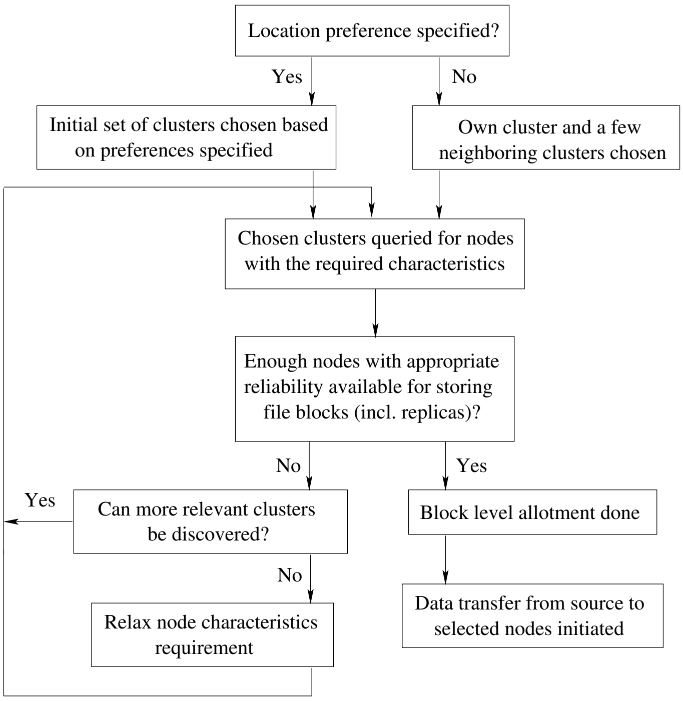

Location preference specified?
Yes
No
Initial set of clusters chosen based on preferences specified
Own cluster and a few neighboring clusters chosen
Chosen clusters queried for nodes with the required characteristics
Enough nodes with appropriate reliability available for storing file blocks (incl. replicas)?
No
Yes
Yes
Can more relevant clusters be discovered?
Block level allotment done
No
Relax node characteristics requirement
Data transfer from source to selected nodes initiated

FIG. 4.1. *Bird's Eye View of the File Placement Strategy*

4.2. File Access. In ENFS, file metadata is maintained on reliable and capable nodes, with a single point of access for each file. Such a design makes it convenient for the system to support a large spectrum of access semantics. The single interface allows clients to use the data in various modes. Concurrent accesses can also be easily coordinated. For example, concurrent accesses to files can be controlled by associating files with exclusive and shared locks. All exclusive locks must have been released before shared locks can be obtained by clients and all kinds of locks (shared and exclusive) must be released before an exclusive lock can be obtained. Lock granularity (block level or file level) and lifetime can also be configured. Applications can use ENFS library functions such as *lockf()* and *lockb()* to lock/unlock files and blocks respectively. These functions take as arguments the filename, block identifier, type of lock, lifetime, etc. Supernodes ensure that locks are not violated while responding to read and write requests from multiple users.

4.2.1. Access Semantics. By selecting appropriate access modes, applications can improve their performances significantly. Selecting the right kind of lock can reduce wait times to access data. Due to concurrent accesses, it is possible that multiple versions of a file are available. A user, not too particular about requiring the latest version, can request for read access to any version of a file [42]. This way, the user is more likely to gain access faster. Users can also specify whether the order in which their updates are applied on files is important or not. When the updating order is not important, multiple updates can be simultaneously applied on the various replicas and hence high performance can be achieved. Commutative updates such as counter increments/decrements can be unordered. The *setattr* shell command can be used to modify such options for a file:

```
$ enfs setattr [options] enfs_filename
```

Caching file contents locally helps in reducing data access times during subsequent usages. In order to ensure that contents have not been modified, files and blocks maintain version numbers (included in the file metadata). Version numbers in the metadata can be compared with version numbers of data items cached by users to determine if cache contents are valid or not. Block level version numbers are useful, especially in the case of large files. Without block level version numbers, caches can be invalidated even when a small portion of a large file is altered.

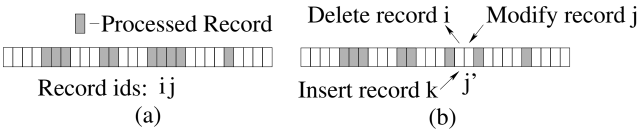


FIG. 4.2. *(a) Progress of a Computation (b) Changes Made to the File*

4.2.2. Computation Scheduling. Most high performance computing applications involve operations such as filtering the contents of a file to extract useful information, enforcing data transformations or performing CPU intensive scientific computations on file contents. Since supernodes have information about all the resources in their cluster and are capable of discovering resources from other clusters, they are well suited for the job of scheduling different kinds of operations on file contents. Based on the specified access preferences, supernodes schedule tasks on appropriate nodes and have the results returned to the user. For example, in order to execute a divisible load application [7] on the contents of a file, supernodes can efficiently set up computation specific dynamic overlay networks, such as the one discussed in our earlier work [31].

In a large scale system involving several users and long running applications, scenarios can arise in which user U_a makes changes to a file while another user U_b is performing a high performance computation on the file contents. A few examples for scenarios where computation results need to be kept up-to-date with changing datasets are web crawlers providing fresh content, re-runs of experiments resulting in better values for a portion of the dataset, sensors detecting new events, etc. Requiring U_b to restart the computation after every change made to the file is not efficient since file sizes are usually huge and computations take a significant amount of time. Similarly, not allowing changes to be made to the file when a computation is being performed can lock the file for long periods of time. In ENFS, we address this issue by enabling the user to provide certain functions (similar to MapReduce [14]) to the responsible supernode which can then handle simultaneous updates to the file while computations are in progress on the data. For certain classes of application, such as embarrassingly parallel applications with commutative updates, the scheduler can continue processing one portion of the file when another portion is being altered, because there is little or no inter-task communication involved. Thus, when changes are made to a file, if it is possible to undo the effects of deleted records from the result, the computation can be made to be consistent with the current contents of the file. For such applications, it may be possible for the user to provide the following details/functions to the supernode:

- The types of *record* and *result*
- The function $compute(record) => result$—represents the computation that is performed on every record.
- The function $aggregate(< result > *) => result$—aggregates a set of results into one.
- The function $recompute(result, record) => result$—reverses the effect of the computation of a single record from the current result.

Let us assume that the computation initiated by U_b has progressed as shown in figure 4.2(a). Each small box represents a record. Shaded boxes represent records that have been processed and unshaded blocks represent records that have not yet been processed. Let us also assume that U_a modifies the file as shown in figure 4.2(b). U_a deletes the record labeled i, inserts record k and modifies record j, the modified record being labeled j'. In order to make U_b's computation consistent with the current contents of the file, the following functions must be invoked by the supernode:

1. $result = recompute(result, i)$
2. $result = recompute(result, j)$
3. $result_1 = compute(k)$
4. $result_2 = compute(j')$
5. $result = aggregate(result, result_1, result_2)$

Steps 1 and 2 are required to undo the effects of old records i and j on the result. Steps 3 and 4 obtain the results of performing the computation on new records k and j'. The final step combines the three results into one. The updates will then have been incorporated into the result and the computation can continue with the records which have not been processed yet. With such user defined functions, ENFS can be customized to support a wide variety of access semantics.

4.3. Data Migration. Data stored in a storage node may have to be migrated to other nodes due to several reasons such as changing access patterns, failure of storage nodes and changes in the characteristics of nodes. Data migration and creation of new replicas based on access patterns help in improving application performance. In ENFS, data transfer time is saved by migrating blocks closer to the nodes/clusters where they are being frequently transferred to. This reduces the effects of high network latency on access times. Moreover, based on the frequency of access, new replicas are autonomously created and unused ones deleted.

A supernode ascertains that a node has abnormally failed only if the node remains offline for extended periods of time. When a supernode detects an abnormal node failure, it passes on the information to the supernodes responsible for the files stored in the failed node. The respective supernodes can then create new replicas. Changes in node characteristics are also handled in a similar manner.

5. Performance Studies. A prototype version of ENFS has been implemented and in this section, we present preliminary performance results. We have used the Andrew Benchmark [20] to assess ENFS. Andrew Benchmark (AB) is widely used to study file system performances and consists of five phases, namely: 1) creation of directories, 2) copying files into the directories, 3) reading file attributes, 4) reading file contents, and 5) performing a hybrid set of operations on the files, such as reading, computing and writing. All the five phases of the workload are relevant in the context of a DFS for HPC applications. Phases 1 and 3 characterize metadata creation and access performance. Phases 2 and 4 characterize data write and read performance. Phase 5 measures the support provided to computation tasks. The performance of a DFS relative to popular file systems such as NFS can be used to make indirect comparisons between itself and any other DFS for which results exist.

Our experimental setup includes a set of 80 machines belonging to two labs in our department. Most of the machines have Pentium 4 processors (3.8 GHz), 1 GB RAM and 80 GB hard disk drives. The predominant operating system is Linux version 4. The machines are all connected to a 100 Mbps LAN, with 30 machines in one subnet and the remaining 50 in another. The machines are split into 2 to 5 clusters, each cluster having between 16 and 70 nodes, depending on the experiment. On an average, each cluster maintains 4 supernodes. The NFS server is a Pentium 4 machine of 3.8 GHz processor speed and 4 GB RAM, running NFSv3.

Table 5.1
Andrew Benchmark: ENFS vs. NFSv3

Phase	ENFS (ms)	NFSv3 (ms)
Phase 1	40	8
Phase 2	72066	83227
Phase 3	202	51
Phase 4	34905	86487
Phase 5	3141	28627

Table 5.2
Overheads of the Paxos Algorithm

No. of Supernodes	Time taken for Consensus (ms)
2	47
16	62
32	108
64	269

Table 5.1 shows the running times of the different AB phases for ENFS and NFSv3. Phases 2 (write) and 4 (read) involve a lot of data transfer (800 MB in each phase). Since data storage is distributed among several nodes, the running times of these phases are lesser in ENFS than in NFS. ENFS also performs significantly better than NFS in phase 5 because it exploits the computing power of storage nodes to parallelize computations. Overall, in this setup, ENFS performs about forty percent better than NFS. In comparison, Pastis [9] achieves execution times lesser than two times that of NFS, while Ivy [29] and OceanStore [24] perform two to three times worse than NFS.

Multiple instances of AB are started in different nodes simultaneously and the performance studied. Figure 5.1 shows the average execution time per run for the different number of simultaneous AB runs (1 to 5). In ENFS, when the nodes on which AB is started belong to different clusters, the average execution times is more or less a constant. When the nodes belong to the same cluster, a small increase in the execution time is noticed with every additional node. This is because, in the first case, a larger set of supernodes cater to client requests than in the second case, where only a single cluster's supernodes handle all the load. NFS performance deteriorates rapidly with the number of runs because of the single server which serves data and metadata to all the nodes.

The same workload is used to measure the execution times of AB runs in the Hadoop Distributed File System (HDFS) [8]. HDFS is an open source file system with an architecture similar to that of the Google File

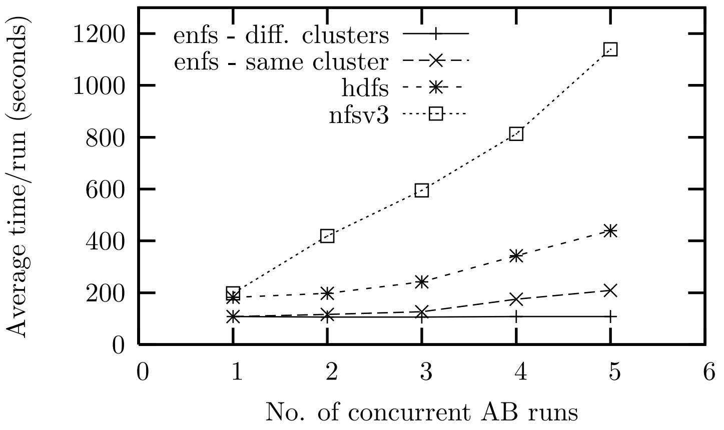


Fig. 5.1. *Concurrent AB Runs*

System (GFS) [17] (discussed in section 6). A HDFS cluster maintains a single *namenode*, which manages the namespace, and several *datanodes*, which store data blocks and serve read and write requests. Since data storage is decentralized, HDFS also performs much better than NFS. However, since the metadata is still managed by a central server in HDFS, its performance is effected when the number of simultaneous file system operations increases. When five concurrent AB runs are started, HDFS's average benchmark time is more than two times that of ENFS.

We measure the overhead introduced by the Paxos algorithm while trying to achieve consensus among supernodes. Table 5.2 shows the average time taken to achieve consensus among varying numbers of supernodes within a cluster. We observe that, though agreement time increases with the number of supernodes, the overheads are not too high (about a quarter of a second for 64 supernodes in a LAN setting).

A flash crowd scenario is created and the response of ENFS is studied. Table 5.4 shows the average response times for metadata requests as observed by clients, with and without load distribution. As expected, the average response time reduces significantly when the responsibility of supplying popular file's metadata is distributed among the supernodes of the clusters from which requests are being issued.

Table 5.3 shows the execution times for a computation on a 600 MB dataset while it is being updated by other users. The time taken to *compute* a single record, of size 1 MB, on a 3.8 GHz Linux machine is 102 ms and the time taken to *recompute* the result when a record is updated is 151 ms. It may be noted that the *recompute* function is applied only on the updated records and not on the entire file. Execution times increase linearly with the rate of updates to the file. Also, for the entire computation, we observe that execution times are lesser for smaller block sizes. As the block size increases, the number of storage nodes used to store the file decreases. In other words, the number of nodes, whose processing power is directly usable for the computation decreases, thus increasing computation times.

Table 5.3
Computation Execution Times

Updates	Execution Time (seconds)		
/second	Block size (MBs)		
	20	50	100
4	4	8	15
10	6	11	17
20	10	15	22
40	17	23	31

Table 5.4
Flash Crowds: Average Response Times

	Avg. response time (ms)
Flash crowds	29
After load distribution	11

6. Related Work. Certain distributed file system protocols such as *Network File System* (NFS) [38], *Andrew File System* (AFS) [21] and *Common Internet File System* (CIFS) [26] employ the client/server model. File data and metadata are both managed by a single or a limited set of servers. Since the servers are potential bottlenecks, these systems do not scale with increasing numbers of simultaneous accesses. Moreover, these systems support parallelism at the level of directories only, which is not as beneficial to applications as parallelism at the level of files.

A recent approach to file system design has been to decouple data and metadata [17][39][47][10]. In these systems, metadata and data are served by different sets of servers. *Google File System* (GFS) [17] is a DFS for data intensive applications, custom-built for the application workload and technical environment at Google. A single master maintains all file metadata and several chunkservers store file contents. *Lustre File System* [39] and *Panasas File System* [47] are similar systems that decouple data and control and use Object based Storage Devices (OSD) for their storage needs. Requiring specialized object storage nodes precludes the usage of the storage capacities of most edge nodes. *Parallel Virtual File System* (PVFS) [10] is an open source DFS that attempts to provide high performance and scalable file system services for node clusters. While data is spread out among a large number of cluster nodes, metadata is still served by a single server in PVFS1, and by a small set of servers in PVFS2. All these systems maintain a single, or a fixed set of metadata servers. This imposes a constraint on the scalability of the system and also affects performance. For several applications, performance depends on the speed with which file and directory metadata can be accessed and updated.

Farsite [3] is another DFS which decouples data and metadata. File system namespace is maintained using a collection of replica groups arranged in the form of a tree. File metadata is replicated and stored on the directory group nodes in a Byzantine fault tolerant manner. As with other hierarchy traversal systems, locating the directory group for a file deep in the hierarchy may require several hops, thus making metadata access expensive. *Ceph* [44] is an OSD based DFS which uses a pseudo random function called CRUSH [45] for the distribution of data among nodes in the object storage cluster. The usage of CRUSH rules out the possibility of considering specific node characteristics while making data placement decisions.

OceanStore [24] is a global scale data storage utility that uses untrusted infra-structure. Data objects are stored on primary and secondary tier nodes. Updates are serialized among the primary tier nodes using a Byzantine fault tolerant algorithm and propagated to the secondary tier nodes using a dissemination tree. High churn rates among the primary tier nodes can result in expensive maintenance overheads. The overheads involved in the maintenance of two tiers of nodes and a tree for each data object can also be significantly high. Moreover, every update results in the creation of a new version which is archived in the system, making the system inefficient for large sized files. *Ivy* [29] is a P2P read/write file system based on logs. Each participant maintains a log with information about all the changes made to the data and metadata in the file system. A participant appends changes only to its own log. However, it reads from the logs of other participants and maintains a total order by using version vectors. As a result, Ivy is suitable only for a small number of participants.

JuxMem [5] is a hierarchical platform that supports mutable data sharing services for Grid computing. The hierarchy includes a set of peer groups (clusters) and an overlay network consisting of cluster managers from the peer groups. A data group is created for each data block that is stored in the system. The group handles tasks such as replication, consistency maintenance, data access, etc. The overheads of maintaining individual groups for the data items can affect the system's scalability with respect to the number of data items. Moreover, JuxMem largely ignores namespace management issues and the task of optimizing metadata access.

7. Conclusions and Future Work. In this paper, we have discussed the design of a distributed file system built on top of Internet edge nodes. The DFS attempts to address the needs of widely spread scientific communities that share and run high performance computing applications on large data files. The system allows data placement decisions to be based on user's preference of storage resource characteristics and locations. The two layered architecture allows for the efficient discovery and usage of system-wide storage resources with specific characteristics. ENFS supports flexible consistency management and access semantics to improve application performance. We demonstrate a mechanism that allows a user to continue performing computations on a large dataset, while it is being modified by another user. The limitation, however, is that it can be applied only to certain types of application.

ENFS needs to be further tested by obtaining performance measurements from a larger and wider network of nodes, resembling a real-world scenario. The design of ENFS can be exploited for efficient file discovery. Files can be associated with tags or keywords that describe their contents. The tags to clusters mapping can be stored in the structured overlay in much the same way as it is done for the discovery of far-off resources with specific characteristics. This significantly reduces the number of clusters that must be queried for discovering relevant files. A group of HPC enthusiasts are working towards defining a few POSIX file system APIs conducive to high performance computing applications [1]. A possible future work is to make ENFS support the proposed POSIX extensions.

REFERENCES

[1] *POSIX extensions.* http://www.pdl.cmu.edu/posix, 2006.

[2] *Biomedical Informatics Research Network.* http://www.nbirn.net, 2007.

[3] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer, *Farsite: Federated, available, and reliable storage for an incompletely trusted environment*, SIGOPS OSR, 36 (2002), pp. 1–14.

[4] S. R. Alam and J. S. Vetter, *An analysis of system balance requirements for scientific applications*, in Proc. of ICPP '06, 2006, pp. 229–236.

[5] G. Antoniu, L. Boug, and M. Jan, *Juxmem: An adaptive supportive platform for data sharing on the grid*, Scalable Computing: Practice and Experience, 6 (2005), pp. 45–55.

[6] Y. Bejerano, Y. Breitbart, M. Garofalakis, and R. Rastogi, *Physical topology discovery for large multisubnet networks*, in Proc. IEEE INFOCOM 03, 2003, pp. 342–352.

[7] V. Bharadwaj, T. G. Robertazzi, and D. Ghose, *Scheduling Divisible Loads in Parallel and Distributed Systems*, IEEE Computer Society, 1996.

[8] D. Borthakur, *The Hadoop Distributed File System: Architecture and design.* http://hadoop.apache.org/core/docs/current/hdfs_design.html, 2008.

[9] J.-M. Busca, F. Picconi, and P. Sens, *Pastis: A highly-scalable multi-user peer-to-peer file system*, in Euro-Par, 2005, pp. 1173–1182.

[10] P. H. Carns, Walter, R. B. Ross, and R. Thakur, *PVFS: A parallel file system for linux clusters*, in Proc. 4th Annual Linux Showcase and Conference, 2000, pp. 317–327.

[11] A. Chien, B. Calder, S. Elbert, and K. Bhatia, *Entropia: Architecture and performance of an enterprise desktop grid system*, JPDC, 63 (2003), pp. 597–610.

[12] B. Cochran, *Distributed file systems: History, taxonomy, and cutting edge ideas*, tech. report, University of Illinois at Urbana-Champaign, 2004.

[13] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, *Vivaldi: A decentralized network coordinate system*, in Proc. of the ACM SIGCOMM'04 Conference, 2004, pp. 15–26.

[14] J. Dean and S. Ghemawat, *MapReduce: simplified data processing on large clusters*, Commun. ACM, 51 (2008), pp. 107–113.

[15] D. Eastlake and P. Jones, *US Secure Hash Algorithm 1 (SHA1).* RFC Editor, USA, 2001.

[16] M. R. Fahey and J. Candy, *GYRO: A 5-D Gyrokinetic-Maxwell Solver*, in Proc. of Supercomputing '04, 2004, p. 26.

[17] S. Ghemawat, H. Gobioff, and S.-T. Leung, *The Google File System*, SIGOPS OSR, 37 (2003), pp. 29–43.

[18] R. Govindan and H. Tangmunarunkit, *Heuristics for Internet map discovery*, in Proc. IEEE INFOCOM 00, 2000, pp. 1371–1380.

[19] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger, *Data management in an international data grid project*, in Proc. of First IEEE/ACM International Workshop on Grid Computing, Springer-Verlag, 2000, pp. 77–90.

[20] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. West, *Scale and performance in a distributed file system*, SIGOPS OSR, 21 (1987), pp. 1–2.

[21] J. H. Howard, *An overview of the Andrew File System*, in USENIX Winter, 1988, pp. 23–26.

[22] D. J. Kerbyson and P. W. Jones, *A performance model of the Parallel Ocean Program*, Int. J. High Perform. Comput. Appl., 19 (2005), pp. 261–276.

[23] T. Kosar and M. Livny, *Stork: Making data placement a first class citizen in the grid*, in Proc. of ICDCS '04, 2004, pp. 342–349.

[24] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao, *Oceanstore: An architecture for global-scale persistent storage*, SIGARCH Comput. Archit. News, 28 (2000), pp. 190–201.

[25] L. Lamport, *The part-time parliament*, Trans. Comput. Syst, 16 (1998), pp. 133–169.

[26] P. J. Leach and D. C. Naik, *A Common Internet File System Protocol (CIFS).* Internet draft, Internet Engineering Task Force (IETF), 1997.

[27] E. Lua, J. Crowcroft, and M. Pias, *Highways: Proximity clustering for scalable peer-to-peer networks*, in Proc. 4th Int. Conference on P2P Computing, 2004, pp. 266–267.

[28] Y. Luo and K. W. Cameron, *Instruction-level characterization of scientific computing applications using hardware performance counters*, in WWC '98: Proc. of the Workload Characterization: Methodology and Case Studies, IEEE Computer Society, 1998, p. 125.

[29] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen, *Ivy: A read/write peer-to-peer file system*, SIGOPS Oper. Syst. Rev., 36 (2002), pp. 31–44.

[30] B. K. Pasquale and G. C. Polyzos, *Dynamic I/O characterization of I/O intensive scientific applications*, in Proc. of Supercomputing '94, 1994, pp. 660–669.

[31] K. Ponnavaikko and D. Janakiram, *Overlay network management for scheduling tasks on the grid*, in ICDCIT, 2007, pp. 166–171.

[32] K. Ranganathan and I. Foster, *Simulation studies of computation and data scheduling algorithms for data grids*, Journal of Grid Computing, 1 (2003), pp. 53–62.

[33] A. Rao, K. Lakshminarayanan, S. Surana, R. M. Karp, and I. Stoica, *Load balancing in structured P2P systems*, in Proc. of IPTPS 2003, 2003, pp. 68–79.

[34] M. V. Reddy, A. V. Srinivas, T. Gopinath, and D. Janakiram, *Vishwa: A reconfigurable P2P middleware for grid computations*, in Proc. of ICPP '06, 2006, pp. 381–390.

[35] R. Rivest, *The MD5 Message-Digest Algorithm*. RFC Editor, USA, 1992.

[36] D. Roselli, J. R. Lorch, and T. E. Anderson, *A comparison of file system workloads*, in Proc. of USENIX Annual Technical Conference, 2000, pp. 4–4.

[37] A. Rowstron and P. Druschel, *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*, in Middleware '01, Nov. 2001, pp. 329–350.

[38] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, *Design and implementation of the Sun Network Filesystem*, in Proc. Summer 1985 USENIX Conf., 1985, pp. 119–130.

[39] P. Schwan, *Lustre: Building a file system for 1000-node clusters*, in Proc. of Linux Symposium, 2003, pp. 380–386.

[40] A. V. Srinivas and D. Janakiram, *Scaling a shared object space to the Internet: Case study of Virat*, Journal of Object Technology, 5 (2006), pp. 75–95.

[41] I. Stoica, R. Morris, D. Karger, F. M. Kaashoek, and H. Balakrishnan, *Chord: A scalable peer-to-peer lookup service for Internet applications*, in Proc. of ACM SIGCOMM '01, vol. 31, October 2001, pp. 149–160.

[42] J. Stribling, E. Sit, M. F. Kaashoek, J. Li, and R. Morris, *Don't give up on distributed file systems*, in Proc. of the 6th IPTPS, Feb 2007.

[43] A. Takefusa, O. Tatebe, S. Matsuoka, and Y. Morita, *Performance analysis of scheduling and replication algorithms on grid datafarm architecture for high-energy physics applications*, in Proc. of 12th IEEE HPDC, 2003, pp. 34–43.

[44] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, *Ceph: A scalable, high-performance distributed file system*, in OSDI, 2006, pp. 307–320.

[45] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, *CRUSH: Controlled, scalable, decentralized placement of replicated data*, in Proc. of Supercomputing '06, 2006, pp. 31–31.

[46] S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller, *Dynamic metadata management for petabyte-scale file systems*, in Proc. of Supercomputing '04, 2004, pp. 4–4.

[47] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, *Scalable performance of the Panasas parallel file system*, in Proc. of the 6th USENIX Conference on File and Storage Technologies, 2008, pp. 1–17.

[48] Q. Xu and J. Subhlok, *Automatic clustering of grid nodes*, in Proc. of GRID '05, IEEE Computer Society, 2005, pp. 227–233.

Edited by: Thomas Ludwig
Received: June 16, 2008
Accepted: March 9, 2009

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:
- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:
- programming environments,
- debugging tools,
- software libraries.

Performance:
- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:
- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:
- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in $\LaTeX 2_\varepsilon$ using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.