Leveraging Public Resource Pools to Improve the Service Compliances of Computing Utilities

Shah Asaduzzaman and Muthucumaru Maheswaran

McGill University, Montreal QC H3A 2A7, Canada,
{asad, maheswar}@cs.mcgill.ca,
WWW home page: http://www.cs.mcgill.ca/∼anrl/

Abstract. Computing utilities are emerging as an important part of the infrastructure for outsourcing computer services. One of the major objectives of computing utilities is to maximize their net profit while maintaining customer loyalty in accordance with the service level agreements (SLAs). Defining the SLAs conservatively might be one easy way to achieve SLA compliance, but this results in underutilization of resources and loss of revenue in turn. In this paper, we show that inducting unreliable public resources into a computing utility enables more competetive SLAs while maintaining higher level of runtime compliance as well as maximizing profit.

1 Introduction

Constant improvements in computer communications and microprocessor technologies are driving the development of new classes of network computing systems. One such system is the *computing utility* (CU) that brings large number of resources and services together in a virtual system to serve its clients. Typically, CUs are built by connecting the resources or services to a *resource management system* (RMS) that itself is implemented either centrally or federally. The RMS allocates resources to the client requests such that some measure of delivered performance is maximized subject to fairness constraints. The organization of the RMS, which impacts the scalability, extensibility, and fault tolerance of the CU is a key consideration in CU design. Support for services with *quality of service* (QoS) assurances is another important design issue in order to attract business critical applications.

This paper is concerned about augmenting CUs using "public" resources (i.e., resources that wish to contribute their computing, storage, and network capacities without subjecting themselves to any contractual agreements). Several large-scale network computing systems such as Gnutella, SETI@home have demonstrated the tremendous potential of using public resources. Our proposed CU architecture augments the deployed dedicated resources with public resource for additional capacity and we refer to it as a *public computing utility* (PCU).

Although different applications can potentially use a PCU, here we consider only high-throughput computing applications. In this situation, job requests belonging to different clients arrive at the PCU at arbitrary times. In a practical

PCU setting, the RMS has to take the allocation decision as soon as the jobs arrive.

In this paper, we devise an online scheduling heuristic for the RMS of the PCU. The PCU online heuristic needs to decide what class of resources (public or private) should be used for servicing a given request. Because the PCU is bound by the SLAs when delivering services to the clients, we need to consider the SLAs in the resource allocation process as well. Section 2 of the paper discusses the related results found in the literature. Section 3 explains the proposed system architecture in detail. Section 4 defines the resource scheduling problem being dealt with in the PCU. Section 6 discusses the results from the simulations performed to evaluate the resource allocation alternatives.

2 Related Work

Multiprocessor job scheduling is a well-studied problem in operations research and computer science. Although several optimal algorithms are available [1] for simpler scheduling problems, most of the interesting and practical scheduling problems are computationally intractable. Scheduling jobs with arrival time and deadline constraints is proven to be a NP-hard problem for more than two processors [2]. In fact [3] proved that optimal scheduling of jobs in multiple processors is impossible if any of the 3 parameters - arrival time, execution time or deadline is unknown. Because in an online scheduling scenario, resource allocations have to be carried out with incomplete information regarding jobs, heuristic solutions are appropriate for this situation. A good survey of online scheduling heuristics can be found in [4].

One major goal of the RMS of a PCU is to enforce QoS according to the SLAs signed up with its clients. Architectures of SLA compliant resource management for cluster of dedicated machines has been studied in several research projects like Oceano [5], Globus Grid [6][7], etc. However, study of scheduling algorithms with detailed performance evaluations were not carried in the above works. Performance evaluation of scheduling heuristics for cluster based hosting centers are found in [8][9][10] with different optimization goals in different cases.

The Condor project [11] focuses on harvesting unused resources from heterogeneous public machines, but their resource management mainly emphasizes on discovery and co-allocation of resources through matchmaking and gangmatching. They do not support SLA driven QoS aware resource management on the public resource pool.

One work that is very close to our work is [12], which examines stochastic QoS on a similar architecture using dedicated private resource and stochastic public resources. Nevertheless, our work is significantly different from theirs in several dimensions. Instead of modelling the public resources with homogeneous performance and stochastic idle times, we have modelled their throughput to be stochastic which captures the real behavior more closely. They have assumed the QoS requirement (cycle length) of applications has distribution identical to that of underlying public reources, but we have relaxed that assumption.

Furthermore, their scheme does not have any long term SLA with the clients, whereas Our scheduling heuristic is devised to simultaneously maximize the net-profit of the service provider and the level of compliance with the long term SLAs. Our investigation also includes job streams arriving from multiple clients that have different SLAs with the PCU.

3 The PCU System Model and Assumptions

The underlying substrate of the PCU is a proximity aware planetary scale P2P network such as the Pastry [13] that connects all the resources that participate in the system. The public resources are expected to be dispersed throughout the network and the private resources can be concentrated as clusters at certain locations. The P2P network enables efficient discovery of the public resources.

Several issues like resource co-allocation, trust and incentive management, load-balancing, etc. should be addressed in developing the resource allocation process in a PCU system. As a first cut at the problem, we consider allocation of only one resource – the processors. The allocation decisions of the RMS are influenced by several parameters including: (a) current utilization of the private resource pool administered by the PCU, (b) current load offered by the different clients, (c) current value of the expected performance of the best-effort resources, and (d) throughput guaranteed to the particular client by the PCU in its SLA. In our current PCU RMS design, there is no progress monitoring of public resources, omly process completions are notified. Inclusion of progress indicators can improve the contribution from public resources towards overall throughput, albeit at the cost of high communication overhead.

4 The Resource Management Problem

Computational jobs arrive from each client of the PCU service provider at arbitrary points in time with each job consisting of arbitrary number of mutually independent parallel components of possibly different but known sizes. An overall deadline is defined for the job before which all the components must finish their execution.

The SLA that is signed off-line between the provider and a client reserves a throughput guarantee for the corresponding client. The SLA defines various parameters including:

- ρ, the ratio of the client-offered workload that is guaranteed to be carried out by the PCU service provider.
- V, the maximum limit on the workload that can be offered by the client.

From these parameters it can be deduced that when the offered load is $v \leq V$, the delivered throughput should be $\geq \rho v$ to be compliant with the SLA. If offered load v is greater than V, it is sufficient for the PCU to deliver ρV amount of throughput.

The PCU provider earns revenue in proportion to the total delivered computational work for the jobs that finish completely within their deadline (with all of its components). There is penalty for violation of the SLA terms and the penalty is proportional to amount of deviation of the delivered throughput from guaranteed throughput, measured over a specified time window. The optimization goal of the job scheduler is to maximize the net revenue (i.e., revenue − penalty) of the PCU service provider.

5 Heuristic Solutions for Resource Management

In this section, we present three heuristic solutions to resource management in a PCU environment. The first solution, the PCU heuristic, is proposed as part of this work. The next two solutions are adopted from the scheduling literature for the PCU environment for comparison purposes.

5.1 PCU Heuristic: An Online Resource Allocator

The scheduler of the RMS uses an online heuristic to take decisions about allocating available resources to incoming jobs. To reduce the scheduling overhead, the RMS executes the scheduling rules at discrete points of time (i.e., at the end of each scheduling epoch δ). Another component of the RMS, the SLA monitor measures the current deviation D_c of delivered throughput from required throughput for each client c, according to the SLA specified time-window τ_c and moving average factor α_{sla}. Say the total arrived workload in a time-window is W_a and total completed and delivered workload is W_d, both W_a and W_d being smoothed by moving average with the past values. Then,

$$D_c = \max(V_c, W_a \rho_c) - W_d$$

In the above equation, V_c and ρ_c are SLA defined maximum load and acceptance ratio for client c. The current value of D_c is available to the scheduler at the end of every epoch. There are two parts of the decision taken by the scheduler at the end of every epoch (i) accept newly arrived jobs and start them on public and/or private resources, and (ii) relocate and restart the deadline vulnerable jobs from public resource to the private resource pool (in absence of checkpointing and progress monitoring, it is impossile to migrate without restarting).

Acceptance of jobs For each client, the scheduler maintains a priority queue for newly arrived jobs, ordered by highest contributing job first. For a job with total workload W and total available time T_a before deadline, the throughput contribution is $\frac{W}{T_a}$. Every time the foremost job from the queue of the client having highest $D_c - W_c$ value is chosen, where W_c is is the amount of workload so far accepted for client c in current SLA window.

All the jobs are ultimately accepted, and each of them are assigned one of the two different levels of *launch-time-priority*, which is used for restarting decisions. The jobs are accepted according to the following rules:

1. As long as available dedicated resources allow, schedule jobs with high launch-time-priority with *critical* components on dedicated and the rest on public resources. The components that are expected to violate deadline if scheduled on a public resource according to its currently estimated expected throughput μ, are identified as critical components. Among the M private resources, M_r are reserved for restarting phase (the ratio $\frac{M_r}{M}$ is a design parameter). If M_o resources are already occupied and the selected job has m critical components, this phase continues as long as $M_o + m \leq M - M_r$,
2. For the rest of the enqueued jobs all components are scheduled on public resources. For any client c, as long as total accepted workload in the current SLA window is below $\rho_c V_c$, the launch-time-priority of the accepted job is high, otherwise it is low.

Restart jobs At the end of every epoch, the scheduler restarts some deadline vulnerable job-components from public resources. The job-components that have reached a point where it can be completed before deadline only if run on a dedicated machine, is identified as vulnerable. A priority queue is maintained for all the vulnerable components. The queue is ordered desceding primarily by launch-time-priority (explained earlier) and secondly by violation probablity (p_v). p_v is computed at the job-launch time from the available information (distribution of the public resource throughput, component size and the deadline). From the queue, high launch-time-priority components are restarted as long as any dedicated resource is available. Low launch-time-priority are restarted as long as available dedicated resource is $> M_r$. The rest of components are left on public resource.

5.2 Least Laxity First and Greedy Heuristics

For performance evaluation we compare our PCU heuristic with the well known *Least Laxity First* (LLF) [4] heuristic and a Greedy heuristic. We use the LLF heuristic to schedule the jobs only in the private pool of resources. The laxity is the slack between possible execution finish time and deadline. New jobs form each client enter a separate priority queue ordered by laxity and at every epoch jobs popped from the queue that fits in available dedicated resources are started there, otherwise the job is deferred until it becomes infeasible to execute before deadline. As a fairness scheme the queue of the client with highest deviation from SLA is favored when choosing every job.

The Greedy heuristic, another one that we used for comparison, works on the same PCU architecture with a combination of private and public resource pools. The greedy scheduling policy chooses jobs from the arrival queues in every scheduling epoch in the order of highest contributing job of the highest deviating client first. It schedules all components of incoming jobs on private resources in the order of longer component first, as long as there is spare capacity in the private resource pool. All the remaining job-components are scheduled on public resources until all the arrival queues are exhausted.

6 Simulation Results

Here we evaluate the performance of the PCU heuristic through a simulator written in Parsec [14] by changing different parameters and comparing it with the Greedy and LLF heuristics. In our simulation setup, the service provider had a pool of 100 dedicated machines and an infinite pool of public machines. There were five independent clients each feeding a stream of parallel jobs that should be completed within the given deadlines and having its own SLA. Jobs arrival is a Poisson process, with each job having a random number (k) of parallel components (geometrically distributed). Each component of a job also has a random workload that is from a geometric distribution. Each job has a feasible deadline, i.e., it can always be completed if all the parallel components run on dedicated machines. Unless stated otherwise, the deadline was computed with a uniform random laxity between 0.5 and 2 times the mean component length, from the longest component. This tight deadline allows one trial on the public pool and failing that it should be restarted on a private resource.

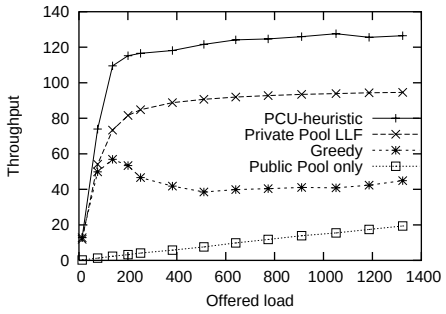

Fig. 1. Variation of mean throughput with offered load values for mean public resource throughput $\mu = 0.80$, mean number of parallel components $P = 25$, total number of private resources $M = 100$, and total SLA booking, $\sum \rho V = 100$.

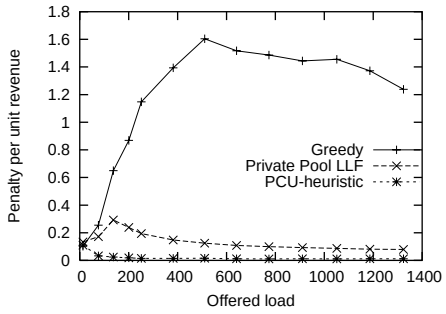

Fig. 2. Variation of penalty per unit revenue with offered load for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$.

All private machines have homogeneous throughput, completing 1 unit of workload of a component per second. The public resource throughput is sampled from Lognormal distribution with standard deviation 1.0 and mean less than 1.0. Justification behind using lognormal distribution is that being left skewed it closely resembles the behavior of the resources in a PCU setting, where most of the public resources may have very low or even 0 throughput.

In the first set of experiments the PCU heuristic is compared with LLF and Greedy using throughput (Figure 1), SLA compliance (measured using penalty per unit revenue in Figure 2). The PCU heuristic delivers better throughput than

LLF, which implies it useful to augment public resource in a CU. Also the PCU-heuristic is superior in performance to the greedy heuristic in similar setting. Figure 3 shows that a much higher gain in throughput is achievable, if the exact knowledge of throughput of each public machine is available at schedule time, because then there is no need for restarting jobs. How far of this gain can be achieved without apriori knowledge remains a problem for future research.

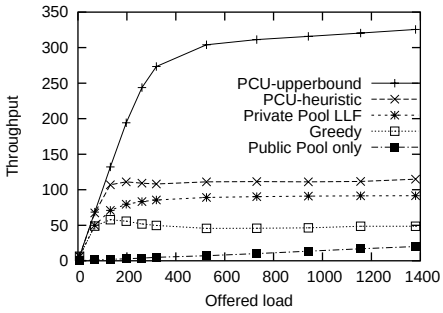

Fig. 3. Upper bound on PCU throughput assuming future behavior of public resources is known for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$.

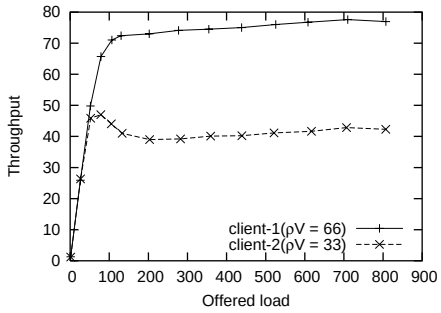

Fig. 4. Comparing delivered throughput to 2 clients having different max-load defined in SLA for $\mu = 0.80$, $P = 25$, and $M = 100$.

Demonstrating the fairness of PCU heuristic figure 4 shows that for 2 different clients, who offers load at the same rate, but has SLA maxload (V) defined at $2:1$ ratio, the delivered throughput is proportional to the maxload of the clients for overloaded situations.

The penalty is higher with the LLF algorithm on private pool only system than the PCU heuristic, because jobs are not deprioritized when the client is offering more workload than the SLA upper bound. In case of the greedy algorithm, penalty grows even higher when the client is overloading, because the dedicated pool gets fully occupied and most of the newly arriving jobs are put on public resources. Consequently, only a small portion of the newly arriving jobs can finish before their deadlines.

As Figure 5 shows, the utilization of dedicated resources is higher for the greedy policy. This is because Greedy uses the dedicated resources exhaustively. The PCU heuristic tries to execute a job-component primarily using public resources unless it becomes vulnerable for deadline violation. Also, in PCU, to allow the restarting of vulnerable components, it reserves a portion of the dedicated resources (25%) as contingency resources. These factors lower the utilization of dedicated resources in the PCU heuristic. Greedy's utilization is even more than LLF, because, in LLF jobs are not allocated unless the all the components fit in the private resources, whereas, Greedy may put part of a job in private pool and rest in public pool.

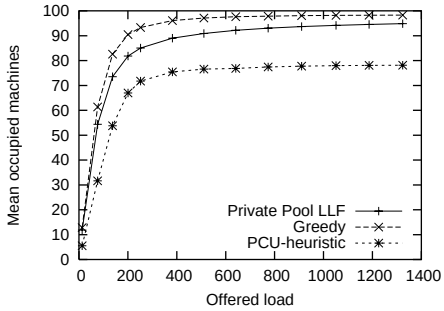

Fig. 5. Utilization of dedicated resources versus offered load for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$.

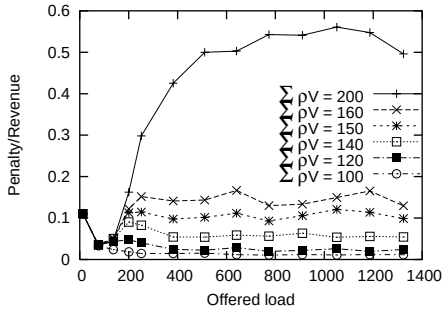

Fig. 6. Penalty per unit revenue earned at different levels of SLA booking for $\mu = 0.80$, $P = 25$, and $M = 100$.

To consider the flexibility in SLA overbooking, if and total agreed upon deliverable throughput (ρV) is higher than the maximum system capacity, the SLA deviation goes very high leading to correspondingly high penalties. This in turn reduces the net profit earned by the service provider. From Figures 6 it can be observed that SLA booking should be at 140% of the dedicated pool capacity to maximize the performance for the given PCU configuration.

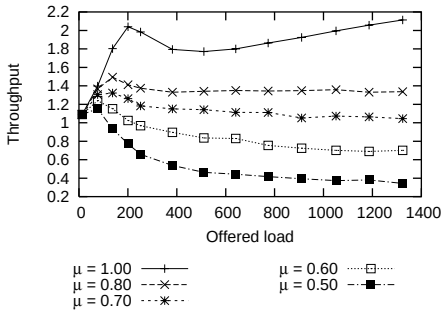

Fig. 7. Throughput gain at different public resource characteristics, with respect to a dedicated pool only system for $P = 25$, $M = 100$, and $\sum \rho V = 100$.

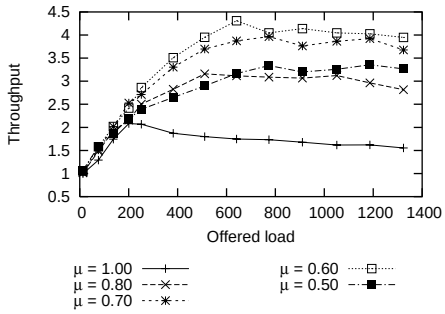

Fig. 8. Throughput gain at different public resource characteristics, with respect to the greedy resource allocation policy on combined pools for $P = 25$, $M = 100$, and $\sum \rho V = 100$.

Figure 7 shows that use of PCU-heuristic brings gain in delivered throughput in most region of the spectrum of public resource behavior. It should be noted that with lognormal distribution, even if the mean throughput is equal to that of a dedicated machine, 62% of the public resources have throughput less than that of a dedicated machine. For very low public ressource throughput, almost all of

the jobs scheduled there needs restart, and since restart is subject to availability in the limited capacity private pool, many jobs get discarded. This explains the less than one throughput-gain with poor quality of public resources. Figure 8 shows that PCU-heuristic outperforms the greedy heuristic accross the whole spectrum.

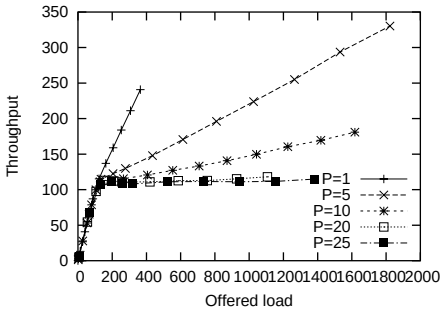

Fig. 9. Mean throughput at varying degree of parallelism for $\mu = 0.80$, $M = 100$, and $\sum \rho V = 100$.

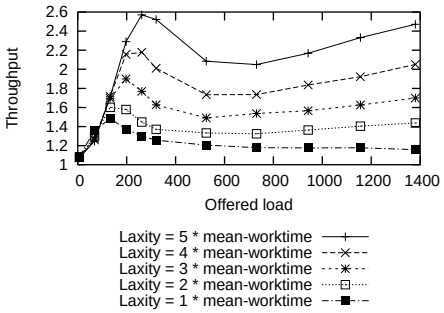

Fig. 10. Throughput gain at different amount of laxity in deadline, with respect to a dedicated pool only system for $\mu = 0.80$, $P = 25$, $M = 100$, and $\sum \rho V = 100$.

Studying the effect of parallelism figure 9 shows that the effect is insignificant in underloaded situations, but when the system is overloaded, high number of parallel components increase the probability of failure of a whole job due to failure of only one or few components which could not be restarted when necessary. Hence, the total delivered throughput becomes low.

Study on the effect of laxity before deadline (Figure 10)f shows that throughput gain is much higher with relaxed laxity jobs. This is because with relaxed laxity the probability of getting a job component completed before deadline on a public resource increases, which incurs less restarts and better contribution from public resources.

7 Conclusion

In this paper, we presented the idea of creating a public computing utility by augmenting computing utilities of dedicated resources with public resources. A resource management strategy for such an augmented system was presented. We proposed a resource allocation heuristic that uses the public and private (dedicated) pools of resources in an efficient manner. We carried out extensive simulations to evaluate the performance of the proposed heuristic and compare it with two other heuristics.

The results indicate that the use of public resources can lead to significant performance improvements both in terms of obtainable throughput and the com-

pliance with client SLAs. Further, the results indicate that the performance gain from PCU increases if the job has fewer components or relaxed deadlines. The performance of the PCU heuristic may be further improved by incorporating these parameters in the decision process.

One of the significant features of our PCU architecture is the minimal monitoring on the public resources. Because public resources are plenty this helps to keep the overhead low. It might be possible to selectively enable performance monitoring for high capacity public resources and increase the delivered performance levels even further.

References

1. Lawler, E.L., Lenstra, J.K., Kan, A.H.G.R., Shmoys, D.B.: 9. In: Handbooks in Operations Research and Management Science. Volume 4. Elsevier Science Publishers (1993) 445–522
2. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the theory of NP-Completeness. W H Freeman and Company, New York (1979)
3. Dertouzos, M.L., Mok, A.K.L.: Multiprocessor on-line scheduling of hard-real-time tasks. IEEE Transactions on Software Engineering **15** (1989) 1497–1506
4. Sgall, J. In: On-Line Scheduling – A Survey. Springer Verlag (1997) 196–231
5. Appleby, K., Fakhouri, S., Fong, L., Goldszmidt, G., Kalantar, M., Krishnakumar, S., Pazel, D., Pershing, J., Rochwerger, B.: Oceano – SLA based management of a computing utility. In: Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management. (2001)
6. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. International J. Supercomputer Applications **15** (2001)
7. Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S.: SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. Lecture Notes in Computer Science **2537** (2002) 153–183
8. Chase, J.S., Anderson, D.C., Thakar, P.N., Vahdat, A.M., Doyle, R.P.: Managing energy and server resources in hosting centers. In: 18th ACM Symposium on Operating Systems Principles. (2001)
9. Ranjan, S., Rolia, J., Knightly, E.: QoS driven server migraion for internet data centers. In: Proceedings of IWQoS 2002. (2002)
10. Aron, M., Druschel, P., Zwaenepoel, W.: Cluster reserves: A mechanism for resource management in cluster-based network servers. In: Proceedings of ACM SIGMETRICS. (2000)
11. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: The condor experience. Concurrency and Computation: Practice and Experience (2004)
12. Kenyon, C., Cheliotis, G.: Creating services with hard guarantees from cycle harvesting resources. In: Proceeings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03). (2003)
13. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany (2001) 329–350
14. Bagrodia, R., Meyer, R., Takai, M., Chen, Y., Zeng, X., Martin, J., Park, B., Song, H.: Parsec: A parallel simulation environment for complex systems. Computer **31** (1998) 77–85