Topology-Aware Resource Allocation for Data-Intensive Workloads

Gunho Lee
University of California, Berkeley
gunho@cs.berkeley.edu

Niraj Tolia
HP Labs, Palo Alto
niraj.tolia@hp.com

Parthasarathy Ranganathan
HP Labs, Palo Alto
partha.ranganathan@hp.com

Randy H. Katz
University of California, Berkeley
randy@eecs.berkeley.edu

ABSTRACT

This paper proposes an architecture for optimized resource allocation in Infrastructure-as-a-Service (IaaS)-based cloud systems. Current IaaS systems are usually unaware of the hosted application's requirements and therefore allocate resources independently of its needs, which can significantly impact performance for distributed data-intensive applications.

To address this resource allocation problem, we propose an architecture that adopts a "*what if*" methodology to guide allocation decisions taken by the IaaS. The architecture uses a prediction engine with a lightweight simulator to estimate the performance of a given resource allocation and a genetic algorithm to find an optimized solution in the large search space. We have built a prototype for Topology-Aware Resource Allocation (TARA) and evaluated it on a 80 server cluster with two representative MapReduce-based benchmarks. Our results show that TARA reduces the job completion time of these applications by up to 59% when compared to application-independent allocation policies.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Distributed Systems; E [**Data**]: Miscellaneous

General Terms

Design, Management, Performance

Keywords

Virtualization, Performance, Modeling, MapReduce, Hadoop, Topology Awareness, Infrastructure-as-a-Service, IaaS

1. INTRODUCTION

Recently, there has been a dramatic increase in the popularity of cloud-based Infrastructure-as-a-Service (IaaS) systems [1] that rent compute resources *on-demand*, bill on a pay-as-you-go basis, and multiplex many users on the same physical infrastructure. Current IaaS systems usually provide Virtual Machines (VMs) that are subsequently customized by the user. As the placement of these VMs can significantly impact application performance, we believe that both the workload's resource usage characteristics and the topology and utilization of the IaaS need to be carefully considered to come up with an optimized allocation policy. Since IaaS providers today are unaware of the hosted application's requirements, they allocate resources independently of an application's requirements. Similarly, as IaaS users do not have fine-grained visibility into or control over the underlying IaaS infrastructure, they can only rely on application-level optimization. While coarse-grained resource selection (e.g., restricting task execution to a particular data center) can be used, it is insufficient for optimizing performance.

Further, even though application-level optimization techniques [10, 20] can be used, they only alleviate the problem within an existing resource allocation. For example, when dealing with communication-intensive workloads, allocating VMs without considering network topology reduces performance by requiring inter-VM traffic to traverse bottlenecked network paths. It is therefore critical to optimize the initial resource allocation that could be responsible for the majority of performance anomalies.

One possible alternative might require users to explicitly specify their resource requirements, or "hints," to guide resource allocation. However, if based on incomplete information, hints can be incorrect or, depending on IaaS resource availability, impossible to satisfy. A successful solution therefore requires the IaaS to derive the information necessary for optimization with minimal or no user input. Further, when compared to application-independent allocation policies, it should improve performance with low latency and high confidence.

To address this issue, we propose an architecture that adopts a "*what if*" methodology. Our solution gathers information without explicit user input, and uses the information to forecast the performance of any particular resource allocation. Our prototype for Topology-Aware Resource Allocation (TARA) is composed of a prediction engine that uses a lightweight simulator to estimate the performance of a given resource allocation and a genetic algorithm to find an optimized solution in the large search space.

To check the feasibility of our general approach, we evaluate it in a particular context. Specifically, in this paper, we concentrate on data-intensive applications, exemplified by MapReduce [6], that are sensitive to the network topology underlying their allocated resources and the background traffic from co-located workloads. We selected these workloads because they represent an important and emerging trend of computation, and are showing increased use on public IaaS systems.

APSys 2010, August 30, 2010, New Delhi, India.

Section 2 describes TARA's architecture. Our prototype-based evaluation in Section 3 demonstrates the accuracy and scalability of the prediction engine and TARA's effectiveness when compared to application-independent resource allocation policies. Sections 4 and 5 conclude with a discussion of related work and an overview of our future work.

2. ARCHITECTURE

TARA is composed of two major components: a prediction engine and a fast genetic algorithm-based search technique. The prediction engine is the entity responsible for optimizing resource allocation. When it receives a resource request, the prediction engine iterates through the possible subsets of available resources (each distinct subset is known as a *candidate*) and identifies an allocation that optimizes estimated job completion time. However, even with a lightweight prediction engine, exhaustively iterating through *all* possible candidates is infeasible due to the scale of IaaS systems. We have therefore developed a genetic algorithm-based search technique that allows TARA to guide the prediction engine through the search space intelligently.

2.1 Prediction Engine

The prediction engine maps resource allocation candidates to scores that measures their "fitness" with respect to a given objective function, so that TARA can compare and rank different candidates. The inputs used in the scoring process can be seen in Figure 1. We describe these three inputs in greater detail below, show how they are obtained without manual input, and then describe how they are used within a lightweight MapReduce simulator.

2.1.1 Objective Function

The objective function defines the metric that TARA should optimize. For example, given the increasing cost and scarcity of power in the data center, an objective function might measure the increase in power usage due to a particular allocation. Our prototype's objective function uses MapReduce job completion time as the optimization metric because it indirectly maps to the monetary cost of executing the job on an IaaS system. The output value for the objective function is calculated using the MapReduce simulator described in Section 2.1.4.

2.1.2 Application Description

The application description consists of three parts: 1) the framework type that identifies the framework model to use, 2) workload-specific parameters that describe the particular application's resource usage and 3) a request for resources including the number of VMs, storage, etc.

The prediction engine uses a model-based approach to predict the behavior of the given application on the selected framework. As each framework behaves differently, it requires a model for the framework being optimized, and the user specifies the framework type. Currently, our prototype only supports the Hadoop-based MapReduce framework.

For MapReduce-based applications, TARA needs runtime-specific information to predict performance (as defined by the objective function). This information is further divided into two groups: framework-specific configuration parameters and job-specific resource requirements.

Framework-specific parameters define the configuration of the application-level environment within which the job executes. Examples include the number of map and reduce slots configured in the MapReduce framework. This information can usually be automatically derived from configuration files.

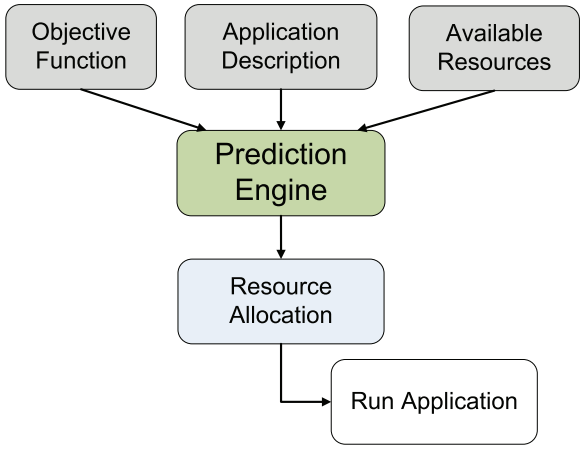

Figure 1: TARA's Architecture

Job-specific resource requirements for the MapReduce framework include selectivity (input/output ratio) during the map and reduce phases, CPU cycles required per input record for both map and reduce tasks, and CPU overhead per task. For our prototype, we analyzed Hadoop logs to derive information on the job. These logs are usually gathered from a previous application run or a test run with a small subset of data.

2.1.3 IaaS Information on Available Resources

The final input required by the prediction engine is a resource snapshot of the IaaS data center. This includes information derived from both the virtualization layer and the IaaS monitoring service. The information gathered ranges from a list of available servers, current load and available capacity on individual servers to data center topology and a recent measurement of available bandwidth on each network link.

2.1.4 Lightweight MapReduce Model/Simulator

As the relationship between a set of selected servers, available network bandwidth, and MapReduce performance is not straightforward, we adopted a simulation-based approach to predict MapReduce job completion time, the objective function's metric. The simulator we built uses the application description, IaaS resource information, and an allocation candidate provided by the search algorithm to simulate the execution flow of the Hadoop MapReduce job. The simulator imitates the behavior of the existing Hadoop scheduler to place tasks on nodes, calculates the resource share of each task, and advances tasks until the job finishes.

As there is no benefit of using TARA's prediction-based approach if finding an optimized resource allocation takes an inordinate amount of time, we decided to trade absolute accuracy for speed. Given that the main objective of the simulator is to score candidates for comparison, we focused of ensuring that the relative performance trend between different candidates was representative of the trends that would be observed on real hardware. To enable fast scoring for each candidate, we used a simplified execution model instead of attempting a full-system simulation. For example, we used a stream-based approach to simulate network traffic. While this is not as accurate as a packet-level simulation, our results showed that it was sufficient to compare aggregate performance. Similarly, we used a simple disk model instead of a DiskSim-based approach [5].

As input, the simulator uses the application description, IaaS resource information, and an allocation candidate, provided by the search algorithm described in Section 2.2. The simulator mod-

els the application by following the control flow of the Hadoop MapReduce framework. As a result, the simulator outputs the predicted completion time of the job.

For every map or reduce task, the simulator will allocate CPU cycles that are proportional to the input size instead of performing the actual computation. Each map task will consume a fraction of the input data and generate intermediate output. The size of intermediate output is determined by the selectivity, or input/output ratio, that was obtained from the job-specific information defined in Section 2.1.2. Following the map step, each reducer then performs a network copy of the intermediate output generated by the map tasks. The job finishes when the last reduce task finishes.

2.2 Search Algorithm

In any large IaaS system, a request for r VMs will have a large number of possible resource allocation candidates. If n servers are available to host at most one VM, the total number of possible combinations is $\binom{n}{r}$. Given that $n \gg r$, exhaustively searching through all possible candidates for an optimal solution is not feasible in a computationally short period of time.

To efficiently identify an approximate solution, we chose a genetic algorithm (GA) [7] to generate possible candidates for the prediction engine to evaluate. GA is a search technique inspired by evolutionary biology for finding solutions to optimization and search problems. Candidates are represented as genes and they evolve toward better solutions. In comparison to other search techniques, we found that GA was a good match for the resource allocation problem. It was natural to map server selection in an IaaS system to GA's gene representation, and to apply operations during the GA's evolution process.

To represent each possible candidate, we use a bit string with the length of n, the number of servers available to host a single VM. The binary value of each bit means whether the corresponding server is selected or not. For each bit in the string, a value of 1 represents the physical server being selected for hosting a VM and a 0 represents the server being excluded. To evaluate a candidate, the prediction engine described above is used.

Once we have the genetic representation and the fitness function, GA initializes a population of candidates. It then goes through the evolution process of reproduction and selection until it terminates. In the reproduction step, mutation, swap, or crossover operations are applied at random to the candidate population to create offspring, i.e., the next generation of candidates. Once each candidate has been evaluated, a stochastic process is used to select a majority of the "fitter" candidates along with a small percentage of "weak" candidates to maintain population diversity.

For our implementation, the population size and the offspring ratio were selected after performing a sensitivity analysis.

3. EVALUATION

There are three main goals of our evaluation. First, we quantify the scalability and performance of TARA's prediction engine. Second, we quantify application performance using a TARA-based resource allocation vs. other heuristic-based resource allocation policies. Finally, we confirm that the performance trends predicted by TARA's simulation-based approach matches those observed by running the application on real hardware.

3.1 Experimental Setup

We used the HP Labs Open Cirrus cluster [2] to evaluate TARA. The testbed used is a subset of the larger cluster and was composed of 111 machines with a single-socket quad-core Intel 3.0 GHz Xeon X3370 processor, 8 GB of RAM, a Gigabit Ethernet port, and four 750 GB disks. There were eight top-of-rack switches with a 10 GigE uplink, each connecting to a maximum of 16 machines.

All machines in the test bed ran the Ubuntu 9.04 Linux distribution with Xen 3.4.1 [4] as the virtualization layer. VMs were configured with access to 4 GB of RAM and 4 Virtual CPUs. For our MapReduce framework, we used Cloudera's Hadoop 0.18.3 distribution with an HDFS replication factor of 3 and Sun's Java 1.6.0.

Even though we have a moderately large test cluster, it is still considerably smaller than the size of most IaaS systems. Therefore, we created virtual topologies and used them as the input to TARA and the different resource allocation policies described below. Based on the resource allocation decision taken, we then used Linux's traffic control and routing mechanisms to create the underlying network topology. While we experimented with a number of different topologies, we only present the results from a representative topology here. The virtual topology used consisted of 20 racks with each rack containing 40 servers available to host a single VM. To model different background workloads and rack-utilization levels, we restricted each rack's uplink bandwidth ranging from 800 Mbit/s to 200 Mbit/s with a step of 100 Mbit/s. Excluding nodes used as virtual switches, we ran benchmarks that spanned 80 nodes.

To evaluate TARA, we used two benchmarks. For Sort, a widely used synthetic benchmark [6], we generated 160 GB (2 GB/node) of random data as input. Analytics, the second benchmark, operates on Wikipedia access logs to track popularity of each article. It is comprised of two phases, and we ran our experiments with a 438 GB dataset that covers three months of logs.

3.2 Resource Allocation Policies

We used four different allocation policies for comparison. These policies, that can be used as manually-generated hints, included:

- **RR-R**: allocates VMs in a round-robin (RR) manner across racks (-R).
- **RR-S**: allocates VMs in a round-robin (RR) manner across servers (-S). This is the default policy used by Eucalyptus [14] and, based on work by Ristenpart et al. [15], we also believe that it is closest to what is used by Amazon's EC2 for a single job. In order to positively bias RR-S results, we also enabled this policy to select racks with the highest available bandwidth first.
- **H-1**: A hybrid policy that combines RR-S and RR-R with a preference for selecting servers in the rack with the greatest available bandwidth but will only select a maximum of 20 servers per rack.
- **H-2**: A hybrid policy similar to H-1 but only selects a maximum of 10 servers per rack.

The above four policies were compared to:

- **TARA**: uses the best allocation found by TARA.

As stated earlier, each physical node never contains more than one benchmark VM for all of the above allocation policies. Further, input data for all benchmarks is only copied into the VMs after they are launched but before the benchmarks are executed.

3.3 Results

3.3.1 Prediction Engine Microbenchmarks

We conducted two different experiments to individually quantify the scalability of both the simulator and the search algorithm.

First, we timed how long the simulator took to predict job completion time for the sort benchmark using a given resource alloca-

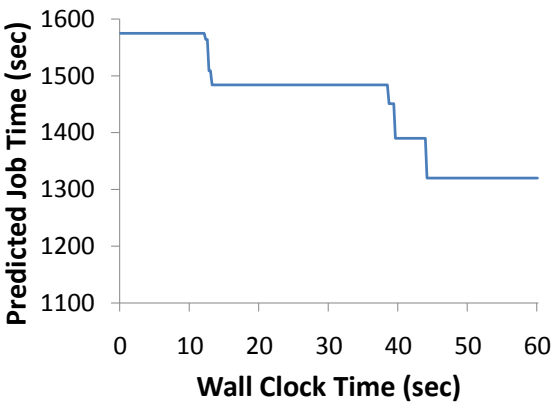

Figure 2: Search Algorithm Efficiency

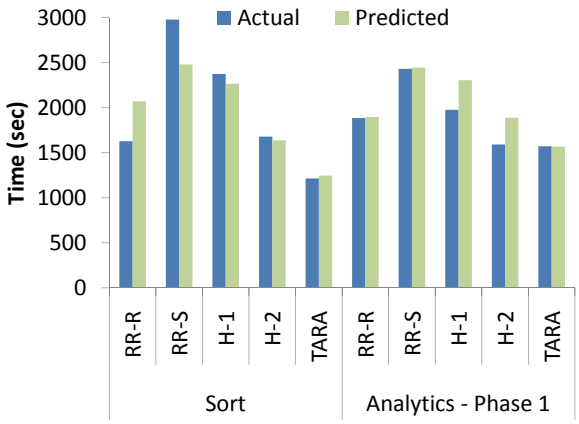

Figure 3: Benchmark Results

tion of 80 VMs. We repeated this experiment 20 times and discovered that, on average, the simulator took 0.96 seconds to predict completion time with very little variation ($\sigma = 0.019$).

Second, we examined the efficiency of the search algorithm in finding an optimized resource placement for the same benchmark and topology. Figure 2 shows that the algorithm discovers a better allocation than the initial population in 12 seconds, and a significantly better allocation in 45 seconds. In addition, we should note that TARA's prediction engine is currently unoptimized and runs on a single server. We expect to reduce the prediction overhead by using more machines and an island model [12] to evaluate a larger number of the candidates in parallel.

3.3.2 Benchmark Results

The results from the benchmarks can be seen in Figure 3, represented by dark (blue) bars. The left side is the result from Sort benchmark, and the result from the first phase of Analytics benchmark is on the right side. The second phase of Analytics is omitted, but the figure is similar. Among the application-independent resource allocation policies, we see that RR-R is the best for Sort, and H-2 for Analytics. It also suggests that no single heuristic is a good fit for all jobs.

Once TARA is introduced, its application and topology-aware resource allocation policy performs considerably better than the application-independent policies. In the case of Sort, it reduces the completion time by 25–59%. For Analytics, TARA reduces the completion time of phase 1 by 1% when compared to H-2 and by as much as 35% when compared to the RR-S. After combining phase 1 and 2, TARA's overall performance gain ranges from 8% to 41% when compared to the completion times of H-2 and RR-S.

In addition, we can compare the prediction engine's output to the results obtained from running on real hardware. The light (green) bars in Figure 3 show the completion time predicted by TARA. As can be seen, the simulation-based predicted completion time doesn't always track the results observed on real hardware. The difference ranges from 2–27% for the Sort benchmark and from 0.3–19% for Phase 1 of the Analytics benchmark. However, as mentioned previously in Section 2.1.4, the goal of the lightweight simulator was not to be an accurate predictor of completion time but instead to correctly identify performance differences between different resource allocations. As seen in the figure, predicted times follow the trends of the actual completion times on real hardware. Further, given the approximate nature of our GA-based algorithm, we believe that the search for fit candidates is not unduly sensitive to small simulation accuracies and are quantifying this as a part of our ongoing work.

4. RELATED WORK

Like TARA, there has been work in the Grid Computing community on communication-aware job placement. However, this work has predominantly concentrated on the overheads of WAN communication between Grid sites [13, 18]. More closely related is the work by Santos et al. [17] on combining administrator policieswith resource allocation decisions but it only looked at a coarse-grained network usage and the simulation-based evaluation examined allocation decisions but did not consider application performance.

Regarding MapReduce, there have been various application-level optimization efforts [16, 20]. While TARA can be used to achieve the same goal, it does so at the infrastructure level and is therefore complementary to the application-level optimization.

Similar to TARA's prediction engine, Wang et al. [19] have also built a Hadoop simulator to predict completion time on different network topologies. However, it would be too slow to be used while determining an optimized resource allocation. In a similar vein, Kambatla et al. [11] attempt to determine optimal values for MapReduce's configuration parameters (e.g., number of map and reduce slots) using resource consumption signatures of similar applications.

Finally, given that network congestion has become a concern in data centers today, there have been hardware-intensive efforts to rewire or rearchitect Layer-2 data center topologies [3, 8, 9]. While we believe that some of these technologies will eventually see wider adoption, they are currently expensive in terms of port costs, extra cabling, end-host modifications, or the requirement to replace existing infrastructure. Further, TARA will benefit data centers that decide to only partially rearchitect the network to provide more bandwidth than currently available but less than full-bisection bandwidth.

5. CONCLUSION AND FUTURE WORK

Cloud-based Infrastructure-as-a-Service models are gaining in popularity. However, the potentially huge variations in performance due to the application-unaware resource allocation in these environments is likely to pose a key challenge for their increased adoption. In this paper, we propose and evaluate a topology-aware resource allocation solution that addresses this problem.

Our approach derives application-specific information with little manual input (retaining the simplicity of interfaces that have

made cloud computing popular) and finds an optimized allocation with low latency and high confidence. In this paper, we focus on MapReduce-based data-intensive workloads and build a solution based on a lightweight MapReduce simulator and a genetic-algorithm based search optimization to guide resource allocation. We have developed a prototype of our architecture and demonstrated the benefits of our architecture on a cluster with 80 nodes on a sort and analytics-based benchmark. Our results show that TARA can reduce completion time by up to 59% when compared to simple allocation policies.

While our work focused on topology-aware resource allocation for data-intensive workloads, the general architecture can be extended to other classes of applications and resource types. We plan on quantifying these benefits as a part of future work. Similarly, we expect to study more sophisticated objective functions including infrastructure costs, power, and reliability. We are also extending our evaluation to more realistic cases in which a node hosts multiple VMs and task completion time can vary due to "noisy neighbors." In addition, while we verified the feasibility of our simulation and the GA-based approach, it requires an explicit model and a sophisticated simulator. We therefore also plan on exploring alternate methodologies for prediction and search, including machine-learning based approaches. Finally, we are also attempting to define a general interface between the IaaS and applications to allow fine-grained control while preserving the flexibility of the IaaS in managing its infrastructure.

Overall, as "cloud-based" platforms see wider adoption, future IaaS systems will increasingly need better resource management architectures. We believe that approaches like ours that address this problem without a significant increase in interface complexity and with low overhead will be a key component of future systems.

Acknowledgments

We would like to thank Jichuan Chang, Stephen Fleischman, Jeffrey Mogul, Matteo Monchiero, Vanish Talwar, and the anonymous reviewers for their feedback on this paper. We are also grateful to Peter Haddad, Jeff Hammerbacher, Hernan Laffitte, Kevin Lai, Dejan Milojicic, Krishnan Narayanan, and Eric Wu for their help with our experimental testbed.

References

[1] `http://aws.amazon.com/ec2/`.

[2] `http://www.opencirrus.org/`.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 63–74, Seattle, WA, Aug. 2008.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 164–177, Bolton Landing, NY, 2003.

[5] J. S. Bucy, J. Schindler, S. W. Schlosser, G. R. Ganger, and et al. The disksim simulation environment version 4.0 reference manual. Technical Report CMU-PDL-08-101, Parallel Data Laboratory, Carnegie Mellon University, Pittsburgh, PA, May 2008.

[6] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI '04)*, pages 137–150, San Francisco, CA, Dec. 2004.

[7] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[8] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 51–62, Barcelona, Spain, Aug. 2009.

[9] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, C. T. Yunfeng Shi, Y. Zhang, and S. Lu. BCube: A high performance, server-centric network architecture for modular data centers. In *SIGCOMM*, pages 63–74, Barcelona, Spain, Aug. 2009.

[10] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair scheduling for distributed computing clusters. In *Proceedings of 22nd ACM Symposium on Operating Systems Principles (SOSP '09)*, Big Sky, MT, Oct. 2009.

[11] K. Kambatla, A. Pathak, and H. Pucha. Towards optimizing hadoop provisioning in the cloud. In *Proceedings of the 1st USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '09)*, San Diego, CA, June 2009.

[12] S.-C. Lin, W. Punch, and E.D.Goodman. Coarse-grain parallel genetic algorithms: Categorization and new approach. In *Proceedings of the Sixth IEEE Symposium on Parallel and Distributed Processing*, pages 28–37, 1994.

[13] C. Liu, L. Yang, I. T. Foster, and D. Angulo. Design and evaluation of a resource selection framework for grid applications. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, pages 63–72, Edinburgh, Scotland, July 2002.

[14] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09)*, pages 124–131, Shanghai, China, May 2009.

[15] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *Proceedings of the ACM Conference on Computer and Communications Security*, Chicago, IL, Nov. 2009.

[16] T. Sandholm and K. Lai. Mapreduce optimization using regulated dynamic prioritization. In *Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 299–310, 2009.

[17] C. A. Santos, A. Sahai, X. Zhu, D. Beyer, V. Machiraju, and S. Singhal. Policy-based resource assignment in utility computing environments. In *Proceedings of the 15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM '04)*, pages 100–111, Davis, CA, Nov. 2004.

[18] O. Sonmez, H. Mohamed, and D. Epema. Communication-aware job placement policies for the KOALA grid scheduler. In *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, page 79, Amsterdam, Netherlands, Dec. 2006.

[19] G. Wang, A. R. Butt, P. Pandey, and K. Gupta. A simulation approach to evaluating design decisions in mapreduce setups. In *Proceedings of the 17th International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2009)*, London, UK, Sept. 2009.

[20] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Job scheduling for multi-user mapreduce clusters. Technical Report UCB/EECS-2009-55, University of California, Berkeley, Apr. 2009.