QoS-Aware Admission Control in Heterogeneous Datacenters

Christina Delimitrou, Nick Bambos and Christos Kozyrakis
Stanford University
{cdel, bambos, kozyraki}@stanford.edu

Abstract

Large-scale datacenters (DCs) host tens of thousands of diverse applications each day. Apart from determining where to schedule workloads, the cluster manager should also decide when to constrain application admission to prevent system oversubscription. At the same time datacenter users care not only for fast execution time but for low waiting time (fast scheduling) as well. Recent work has addressed the first challenge in the presence of unknown workloads, but not the second one.

We present ARQ, a multi-class admission control protocol that leverages Paragon, a heterogeneity and interference-aware DC scheduler. ARQ divides applications in classes based on the quality of resources they need and queues them separately. This improves utilization and system throughput, while maintaining per-application QoS. To enforce timely scheduling, ARQ diverges workloads to a queue of lower resource quality, if no suitable server becomes available within the time window specified by its QoS. In an oversubscribed scenario with 8,500 applications on 1,000 EC2 servers, ARQ bounds performance degradation to less than 10% for 99% of workloads, while significantly improving utilization.

1. Introduction

An increasing amount of computing is performed in the cloud, primarily due to cost benefits for both the end-users and the operators of datacenters (DC) that host cloud services [3]. The operator of a cloud service must schedule the stream of incoming applications on available servers in a *resource-efficient* manner, i.e., achieving fast execution (user's goal) at high resource utilization (operator's goal). This scheduling problem is particularly difficult for several reasons, including diverse application characteristics [3, 19], insufficient workload knowledge, co-scheduled application interference and platform heterogeneity. An additional challenge occurs during periods of adversarial traffic, i.e., intervals with very high load, when the system can become oversubscribed, resulting in poor performance. Most DCs employ some admission control to minimize such effects.

DC users are interested in two performance metrics; how fast the application starts running (*waiting time*) and how fast it completes thereafter (*execution time*). While recent work has shown how to improve execution time in the presence of unknown workloads, varying interference sensitivities and heterogeneous servers [14], it does not solve the "head of line blocking" problem [27]. Additionally, some applications have strict scheduling deadlines, while others can tolerate delays in order to be assigned to preferred servers. In all cases, resource requirements should be taken into account at admission point [8].

We propose *ARQ* (*Admission control with Resource Quality-awareness*), a QoS-aware admission control protocol that builds on Paragon and accounts for the resource quality an application needs to preserve its QoS. Resource quality reflects the additional load a server can support without violating application QoS, given its configuration and the applications it currently hosts. ARQ divides workloads to multiple classes and directs them to different queues. This way demanding workloads do not block easy-to-satisfy applications, as they wait for an appropriate server to become available. On the other hand, since DC applications have strict QoS guarantees, they can only be queued for limited amounts of time, while waiting for an appropriate server. ARQ detects when an application is about to violate its performance requirements and re-directs it to a different queue before the QoS violation occurs. We explore the trade-off between waiting time and quality of resources and solve the corresponding optimization problem to find the optimal switching point.

We evaluate ARQ both in small and large-scale experiments. First, we compare the system without and with ARQ in a local cluster with 40 machines and show the benefits in performance and efficiency. We also evaluate ARQ on a 1000-server cluster on Amazon EC2. For an oversubscribed scenario with 8500 applications, Paragon with ARQ guarantees that 99% of workloads have less than 10% performance degradation, while improving utilization by 46%.

2. Background

2.1. Paragon Overview

Paragon is a heterogeneity and interference-aware DC scheduler [14]. It assigns applications to heterogeneous servers based on the platform they benefit from and the co-scheduled applications that minimize destructive interference to preserve QoS. Paragon has two components, a *classification engine* and a *greedy scheduler*. We briefly describe their operation in the following paragraph.

The first component of Paragon performs fast classification of incoming applications, in terms of the server configuration (SC) they perform better on and the interference they cause and tolerate in various shared resources, such as the processor, cache hierarchy, memory, storage and networking subsystems. The interference profile is obtained through targeted microbenchmarks of tunable intensity that create contention in specific shared resources. These microbenchmarks are called sources of interference (SoIs). The classification engine is built as a recommendation system, similar to Netflix [5] or e-commerce systems and leverages the knowledge the system already has about previously-scheduled applications, keeping profiling overheads low. Then, the greedy scheduler searches for a machine of desired SC, that minimizes destructive interference between existing and new load. Paragon scales to tens of thousands of applications and improves utilization, while maintaining per-application QoS.

2.2. Current Limitations

While Paragon shows that accounting for heterogeneity and interference improves resource efficiency without QoS losses, it does not decide when applications should be admitted and scheduled. Paragon accounts for workload characteristics to decide where to assign a workload, but it does not solve the "head of line blocking" problem that can cause high waiting times. By default, applications are scheduled in a simple FIFO order. This has two shortcomings; first, easy-to-satisfy workloads can get trapped behind demanding applications, e.g., workloads that require exclusive instances of high-end, multi-socket servers to preserve their QoS. Second, in the event of an oversubscribed scenario, i.e., when the required resources are more than the total resources available in the system, Paragon implements an application-agnostic admission control protocol. It queues applications in a single queue until the first server becomes available, and then resumes FIFO-ordered scheduling. This ignores the fact that applications need resources of a certain quality to meet their QoS, and can result in performance degradation.

3. Admission Control

3.1. Overview

Large cloud providers such as Amazon EC2 and Windows Azure, typically deploy some admission control protocol. This prevents machine oversubscription, i.e., the same core servicing more than one applications, resulting in high interference and QoS violations.

We design ARQ, a *QoS-aware admission control protocol* that queues and schedules applications based on the quality of resources they need. This solves two problems; first, applications that demand few, easy-to-satisfy resources are not blocked behind demanding workloads. Second, if no suitable servers are available for a given application, the workload waits for a server of appropriate quality to be freed. Alternatively, the application would be directed to the first free server to avoid queueing delays, with the risk of performance losses.

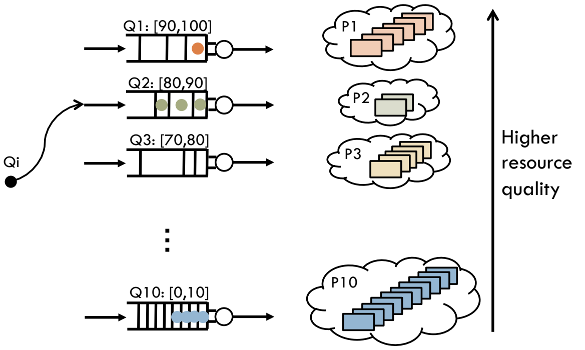

Figure 1: ARQ design. Each queue corresponds to applications with different resource quality requirements.

Resource quality: The resource demands of a workload reflect the load a server should support for the application to meet its QoS. This is a function of the interference the server can tolerate from the new application, and the interference the new workload can tolerate from applications already running on the machine. We use the classification engine in Paragon to derive the per-server tolerated (t_i) and caused (c_i) interference over a set of shared resources. Shared resources include the cache and memory hierarchy, CPU modules, and storage and networking devices. Details on how c_i's and t_i's are obtained can be found in [14]. The interference profile of a server is updated upon initiation or completion of an application's execution. Similarly, upon application arrival, an interference profile is obtained for each new workload. This information guides scheduling decisions by assigning applications to suitable servers. Given the interference profile of a server or application, we define *resource quality* as:

$$Q_i = avg(\sum_i c_i + \sum_i (100 - t_i)) \qquad (1)$$

where c_i and t_i are summed over all shared resources for which interference is measured. Conceptually, higher Q_i reflects applications with high demands (high caused and low tolerated interference) that need high-quality system resources. Low Q_i on the other hand, corresponds to workloads that are insensitive to interference, and can satisfy their QoS even when assigned to servers with poor resource quality, e.g., highly-loaded machines, or machines with few cores.

Multi-class admission control: We design ARQ as an admission control protocol with multiple classes of "customers" [1, 6, 17, 20, 21], where customers in this case correspond to applications. The class an application belongs to is determined by its Q_i value. Applications with Q_i values that fall in the same range are assigned to the

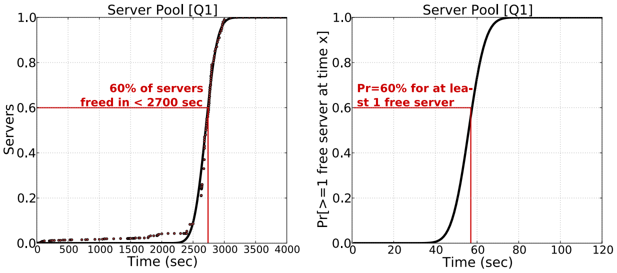

Figure 2: CDF of server busy times and CDF of the probability that there will be at least one free server within a specific time window from an application's arrival.

same class. Q_is range from 0 to 100%. We assume ten classes of applications for now, and justify this selection in the evaluation section (see sensitivity study in Section 5). Fig. 1 shows an overview of ARQ. Each queue corresponds to applications of a specific class. From top to bottom we move from more to less demanding applications. Upon arrival, the cluster manager determines the class an application belongs to and queues it appropriately. Each class has a corresponding server pool of appropriate resource quality. Separating applications based on their resource quality requirements helps ARQ resolve bottlenecks where applications that are sensitive to interference block workloads that are not. On the other hand, applications cannot be queued indefinitely waiting for the perfect server. We address this issue by diverging workloads to queues with better or worse resource qualities.

3.2. Waiting Time versus Resource Quality

Diverging an application to a different queue creates a trade-off between the time an application is waiting in a queue, and the quality of resources it is allocated. We approach this trade-off as an optimization problem.

Queue bypassing: When there is no available server in the pool of a class, queued workloads should be diverged to another queue. There are two possible options for where a workload can be redirected. First, it can be *diverged to a higher queue*. If the queue directly above the queue the workload was originally placed in is empty, the workload is assigned to one of its servers. This hurts utilization, since resources of higher quality than necessary are allocated, but preserves the workload's QoS requirements. In the opposite case the workload is *diverged to a lower queue*. In that case, performance may be degraded, since the application receives resources of lower quality than required. However, the scheme guarantees that in all cases the application will be assigned to a server within the time window dictated by its QoS constraints.

Free-server probability distributions: ARQ needs to know the likelihood that a server of a specific class will become available within the time an application can be queued for, to decide when the workload should be diverged to the next queue. We statistically analyze the server busy time periods for each server pool to obtain these probability distributions. Busy periods are defined as the per-server time intervals from the moment a server is assigned a workload, until that workload completes.

We first use *distribution fitting* to represent the per-pool server busy time in a closed form using known distributions. Fig. 2a shows the CDF of server busy time for the first server pool (highest quality servers) in a 1,000 server experiment. More details on the methodology can be found in Section 4. We show the experimental data (dots) and the closed form representation, derived from distribution fitting. In this case, the data is fitted to a curve resembling a normal distribution. The CDF reflects the fraction of servers that are freed within some time after they have been allocated to an application. For example, 60% of servers in this server pool are freed within 2700 sec from the time an application is scheduled to them.

Using this closed form CDF we easily derive the free-server CDF, which reflects the *probability that within a time interval from an application's arrival, at least one server of the corresponding pool will be available*. Fig. 2b shows the free-server probability CDF for the first server pool. The highlighted point shows that there is a 60% probability that within 56 sec from an application's arrival to that queue, there will be at least one free server in the pool. Free-server CDFs are updated during workload execution to capture changes in application behavior.

Switching between queues: ARQ determines the switching point between queues with the objective to maximize the probability that a server becomes available within a certain window from an application's arrival. For simplicity of explanation we assume that an application's QoS is defined at $0.95x$ of the application's optimal performance. This means that the workload can tolerate at most a 5% performance degradation. Scheduling deadlines or queries-per-second (QPS) can also serve as queueing constraints. Given the free-server CDFs for each server pool, ARQ solves the following optimization problem for application a, switching between queues i and j:

$$max\,\{(S_a - wt_i(t))\cdot Q_i \cdot Pr_i[t], (S_a - wt_j(t))\cdot Q_j \cdot Pr_j[t]\}$$
$$s.t.\,(wt_i(t) + wt_j(t) + P_a) < 0.05\cdot CT_a$$

where $Pr_i[t]$ is the probability that there is a free server in queue i, Q_i is the resource quality of queue i, CT_a is the optimal execution time for application a, P_a is the classification overhead of Paragon, and $S_a = 1.05\cdot CT_a - P_a$ is the available "slack" that can be used for queueing, before the application violates its QoS constraints. ARQ finds the switching time that maximizes the probability that a server of either queue i or j will become available such that the application preserves its QoS guarantees. It also promotes waiting longer for a server of the same class rather than eagerly switching to the next queue ($Q_i > Q_j$).

Server Type	GHz,	cores,	L1(KB),	LLC(MB),	mem(GB)	#
Xeon L5609	1.87	2x8	32/32	12	24 DDR3	1
Xeon X5650	2.67	2x12	32/32	12	24 DDR3	2
Xeon X5670	2.93	2x12	32/32	12	48 DDR3	2
Xeon L5640	2.27	2x12	32/32	12	48 DDR3	1
Xeon MP	3.16	4x4	16/16	1	8 DDR2	5
Xeon E5345	2.33	1x4	32/32	8	32 FB-DIMM	8
Xeon E5335	2.00	1x4	32/32	8	16 FB-DIMM	8
Opteron 240	1.80	2x2	64/64	2	4 DDR2	7
Atom 330	1.60	1x2	32/24	1	4 DDR2	5
Atom D510	1.66	1x2	32/24	1	8 DDR2	1

Table 1: Server characteristics of the local cluster. The total core count is 178 for 40 servers of 10 different SCs.

In our analysis we assume batch, single-node applications. In the case of interactive or transactional workloads additional care must be taken to accommodate load changes, e.g., through VM migration. The scheduler detects such changes and adjusts workload placement to preserve QoS. Detection is based on SoI injection and application reclassification.

4. Methodology

Server systems: We evaluated Paragon on a 40-machine local cluster (Table 1) and a 1000-machine cluster with 14 server types on EC2. We used exclusive (reserved) server instances, i.e., there is no interference from external workloads. We also verified that no external scheduling decisions or actions such as auto-scaling or migration are performed during the course of the experiments.

Schedulers: We compared Paragon with ARQ to four schedulers. First, *Paragon* without admission control, second, a *heterogeneity-oblivious* scheme that only accounts for interference but not heterogeneity. Third, an *interference-oblivious* scheme and finally, a scheduler that is both heterogeneity and interference-agnostic, and assigns applications to *least-loaded* machines.

Workloads: We used 29 single-threaded, 22 multi-threaded, 350 multi-programmed and 12 I/O-bound workloads. We use the full SPEC CPU2006 suite and workloads from PARSEC [7], SPLASH-2 [32], BioParallel [18], Minebench [22] and SPECjbb. For multiprogrammed workloads, we use 350 mixes of 4 applications each [26]. The I/O-bound workloads are data mining applications in Hadoop and Matlab. For scenarios with more than 413 applications we replicated these workloads with equal likelihood and randomized their interleaving.

Workload scenarios: For the *small-scale* experiments we examine three workload scenarios. First, we examine a *low-load* scenario with 178 applications, selected randomly from the workload pool, and submitted with 10 sec inter-arrival times. Second, a *high-load* scenario where 178 applications arrive following a Gaussian distribution (μ=10, σ^2=1) that experience significant *phases* during their execution. Finally, we examine a scenario, where 178 applications arrive with 1 sec intervals. This is an *oversubscribed* scenario, since after a few seconds there are not enough resources to execute all applications concurrently. For the *large-scale* experiments on EC2 we examine an oversubscribed scenario where 7,500 workloads arrive with 1 sec intervals and an additional 1,000 applications arrive in burst after the first 3,750 workloads.

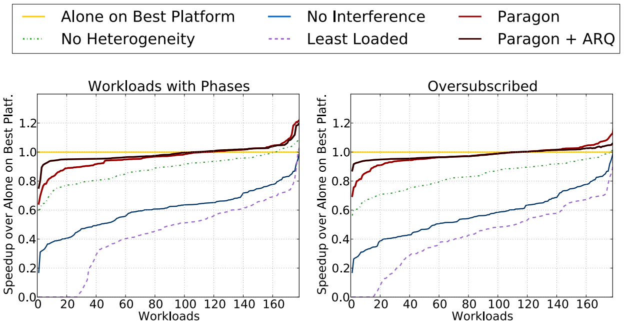

Figure 3: Performance comparison of Paragon and ARQ, across two workload scenarios, against Paragon without admission control, a heterogeneity-oblivious, an interference-oblivious and a least-loaded scheduler.

5. Evaluation

5.1. Small-scale Experiments

Performance: Fig. 3 shows the performance comparison between the different schedulers for the second and third scenarios in the small-scale cluster. The differences for the low-load scenario where resources are plentiful are small. We focus on the differences between Paragon without and with the use of ARQ. Applications are ordered from worst to best performing. For the scenario with workload phases the applications that preserve their QoS increase from 66% to 91%, and the average performance improves to 99.3%. For the oversubscribed system, while without ARQ only 64% of applications maintain their QoS, with ARQ 88% of workloads preserve their performance requirements. This shows that accounting for resource quality at admission point drains the backlog of queued workloads much faster.

Overheads: ARQ limits waiting time to preserve QoS. Fig. 4 shows the breakdown of execution time for selected applications in the oversubscribed scenario. Time is divided in useful execution time, overheads from training and classification, overheads from the greedy server selection [14] and overheads from queueing. *mcf* and *blackscholes* do not have a bar for the least-loaded (LL) scheduler because they did not complete successfully due to memory exhaustion in the server. In all cases overheads are very low and execution time for most workloads is very close to one (optimal). The overheads from queueing are less than 5% at all times. The cases where queueing is high correspond to workloads that had to be diverged

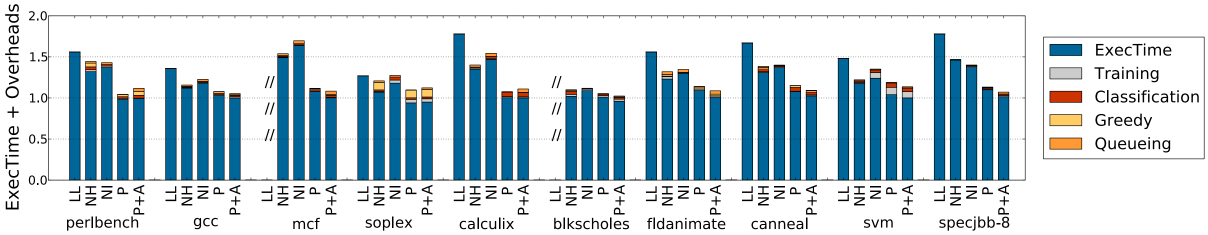

Figure 4: Overheads from classification, queueing and scheduling compared to useful execution time. Overall, the overheads in Paragon with ARQ are less than 5% for most applications.

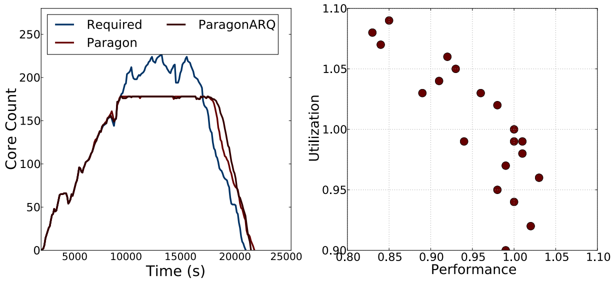

Figure 5: Required versus allocated core count for the oversubscribed scenario in the small-scale system and sensitivity of ARQ to the number of queues. Performance and utilization are normalized to the values for 10 queues.

to queues of lower resource quality, in which case useful execution time is also suboptimal.

Resource allocation: Fig. 5a shows the required versus allocated core count for Paragon with and without ARQ for the oversubscribed scenario. Once the system enters the oversubscribed phase ([9000-17000]sec), Paragon without ARQ allocates all available cores and then queues applications, while Paragon with ARQ will only dispatch applications if an appropriate server is freed. This drains the backlog faster since, even though applications are queued for longer, they run in higher quality platforms.

Server utilization: We also measure server utilization before and after the use of ARQ. We focus on the oversubscribed scenario where ARQ has the highest impact. Paragon without ARQ improves utilization by 47% compared to a LL scheduler. Adding ARQ slightly reduces this improvement since applications are queued instead of being dispatched immediately. Despite this, utilization still improves by 45.5%. This means that the performance benefits of ARQ do not incur an efficiency penalty.

Sensitivity to design parameters: Fig. 5b shows the performance - utilization tradeoff for different numbers of queues. Both metrics are normalized to the values for 10 queues. More queues result in fewer cases of workloads being blocked behind demanding applications, therefore they improve performance, but reduce the number of servers in the corresponding pools, hurting utilization. In contrast, few queues revert to the default scheduler where many applications are scheduled in FIFO order, increasing utilization and hurting performance. 10 queues achieve both high performance and efficiency.

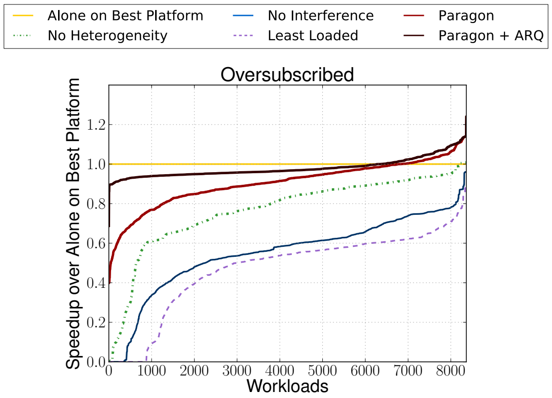

Figure 6: Performance for the different schedulers in the oversubscribed scenario on 1,000 EC2 machines.

Large-scale experiments: Fig. 6 compares the performance of the different schedulers for the large-scale scenario. While Paragon without ARQ only preserves QoS for 61% of workloads, introducing admission control increases that fraction to 83%. Additionally, it bounds degradation to less than 10% for 99% of workloads. This shows that the protocol scales well with the number of servers and applications, while maintaining overheads similar to the ones for the small-scale experiments.

6. Conclusions

We have presented ARQ, a QoS-aware admission control protocol for heterogeneous datacenters. ARQ divides applications to classes based on their resource quality requirements and queues them separately in a multi-class network. ARQ is derived from validated queueing models, and it improves system throughput by reducing application waiting time, and diverging workloads to different queues when necessary. In an oversubscribed scenario with 8,500 applications on 1,000 servers, 99% of workloads experience less than 10% degradation compared to 79% of workloads without ARQ.

Acknowledgements

We sincerely thank Christopher Stewart, Daniel Sanchez, and the anonymous reviewers for their feedback on earlier versions of this manuscript. Christina Delimitrou was supported by a Stanford Graduate Fellowship.

References

[1] R. Atar, A. Mandelbaum, M. Reiman. “Scheduling a Multi Class Queue with Many Exponential Server: Asymptotic Optimality in Heavy Traffic”. *In the Annals of Applied Probability, Vol. 14, No. 3, 2004.*

[2] L. Barroso. "Warehouse-Scale Computing: Entering the Teenage Decade". *ISCA Keynote, SJ, June 2011.*

[3] L. A. Barroso, U. Holzle. "The Datacenter as a Computer". *Synthesis Series on Computer Architecture, May 2009.*

[4] L. A. Barroso and U. Holzle. "The Case for Energy-Proportional Computing". *Computer, 40(12):33–37, 2007.*

[5] R. M. Bell. Y. Koren, C. Volinsky. "The BellKor 2008 Solution to the Netflix Prize". Technical report, AT&T Labs, Oct 2007.

[6] D. Bertsimas, D. Gamarnik, J. Tsitsiklis. “Performance of Multiclass Markovian Queueing Networks via Piecewise Linear Lyapunov Functions”. *In Annals of Applied Probability, Vol. 00, No. 0, 1-45, 2001.*

[7] C. Bienia, S. Kumar, et al. ”The PARSEC benchmark suite: Characterization and architectural implications”. *In Proc. of the international conference on Parallel Architectures and Compilation Techniques (PACT), Toronto, 2008.*

[8] J. Carlstrom, R. Rom. “Application-aware Admission Control and Scheduling in Web Servers”. *In Proc. of Infocom, NY, 2002.*

[9] H. Chen. “Fluid Approximations And Stability Of Multiclass Queueing Networks: Work-Conserving Disciplines”. *In Annals of Applied Probability, Vol. 5, No. 3, 1995.*

[10] S. T. Cheng, C. M. Chen, I. R. Chen. “Performance evaluation of an admission control algorithm: dynamic threshold with negotiation”. *In Performance Evaluation 52, 2003.*

[11] J. G. Dai. “On positive Harris recurrence of multiclass queueing networks: A unified approach via fluid limit models”. *In Annals of Applied Probability, 5:49-77, 1995.*

[12] J. G. Dai. “A fluid-limit model criterion for instability of multiclass queueing networks”. *In Annals of Applied Probability, 6:751.757, 1996.*

[13] J. Dean and S. Ghemawat. ”MapReduce: Simplified Data Processing on Large Clusters”. *In Proc. of Symposium on Operating Systems Design and Implementation (OSDI), SF, 2004.*

[14] C. Delimitrou and C. Kozyrakis. “Paragon: QoS-Aware Scheduling in Heterogeneous Datacenters”. *In Proc. of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Houston, March 2013.*

[15] Amazon Elastic Compute Cloud-EC2. `http://aws.amazon.com/ec2/`

[16] Google Compute Engine. `http://cloud.google.com/products/compute-engine.html`

[17] J. J. Hasenbein. “Stability, Capacity and Scheduling of Multiclass Queueing Networks”. Ph.D. Thesis. Georgis Institute of Technology, 1998.

[18] A. Jaleel, M. Mattina, B. Jacob. ”Last Level Cache (LLC) Performance of Data Mining Workloads On a CMP - A Case Study of Parallel Bioinformatics Workloads”. *In Proc. of int’l conference on High Performance Computer Architecture (HPCA), TX, 2006.*

[19] C. Kozyrakis, A. Kansal, S. Sankar, K. Vaid. "Server Engineering Insights for Large-Scale Online Services". *In IEEE Micro, vol.30, no.4, July 2010.*

[20] V. G. Kulkarni, N. Gautam. “Admission Control of Multi-Class Traffic with Service Priorities in High-Speed Networks”. *In Journal of Queueing Systems: Theory and Applications archive Vol. 27, No. 1/2, 1997.*

[21] B.L. Miller. “A queueing reward system with several customer classes”. *Management Science, 16(3):234-245, 1969.*

[22] R. Narayanan, B. Ozisikyilmaz, et al. ”MineBench: A Bench- mark Suite for DataMining Workloads”. *In Proc. of the IEEE Int’l Symposium on Workload Characterization (IISWC), SJ, 2006.*

[23] Rackspace. `http://www.rackspace.com/`

[24] A. Rajaraman and J. Ullman. ”Textbook on Mining of Massive Datasets”, 2011.

[25] Amazon EC2: Rightscale. `https://aws.amazon.com/solution-providers/isv/rightscale`

[26] D. Sanchez, C. Kozyrakis. ”Vantage: Scalable and Efficient Fine-Grain Cache Partitioning”. *In Proc. of the International Symposium on Computer Architecture (ISCA), SJ, 2011.*

[27] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, J. Wilkes. “Omega: flexible, scalable schedulers for large compute clusters”. *In Proc. of the European Conference on Systems (EuroSys’13), Prague, Czech Republic, April 2013.*

[28] Tanenbaum, A. S. “Modern Operating Systems” (3rd ed.). *Pearson Education, Inc. p. 156.*

[29] vMotion™. "Migrate VMs with Zero Downtime". `http://www.vmware.com/products/vmotion`

[30] VMWare vSphere. `http://www.vmware.com/products/vsphere/`

[31] Windows Azure. `http://www.windowsazure.com/`

[32] S. Woo, M. Ohara, et al. ”The SPLASH-2 Programs: Characterization and Methodological Considerations”. *In Proc. of the Int’l Symposium on Computer Architecture (ISCA), Italy, 1995.*

[33] Xen Hypervisor 4.0. `http://www.xen.org/`

[34] XenServer. `http://www.citrix.com/English/ps2/products/product.asp?contentID=683148`