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. The DC's cluster manager must determine "where" and "when" an application should run, and "how many resources" it should be allocated. This is challenging in cloud environments for several reasons, including workload interference, dynamic resource demands, and hardware platform heterogeneity. Recent work has addressed "where" an application should be scheduled, namely, which of the tens of thousands of servers has the configuration and interference profile that preserves the workload's QoS guarantees.

However, significant challenges remain in terms of "when" an application is scheduled. DC applications have diverse QoS constraints and resource requirements. Typically workloads in DCs are scheduled in admission order, provided all workloads have the same priority. This is suboptimal for two reasons. First, demanding applications, i.e., non interference-tolerant applications, can block non-demanding workloads that can tolerate resource contention. Second, when system resources are fully occupied, the applications are queued until the system exits the oversubscribed phase, and resume scheduling as the first servers become available, to minimize scheduling delays. This can induce significant performance degradations as these servers might not reflect the workload's true resource needs.

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 server utilization and system throughput, while maintaining per-application QoS guarantees. 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 the application's 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]. Large-scale providers such as Amazon EC2 [18], Microsoft Windows Azure [42], Rackspace [32] and Google Compute Engine [21] host tens of thousands of applications on a daily basis. Several companies also organize their IT infrastructure as private clouds, using management systems such as VMware vSphere [41] or Citrix XenServer [45].

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. First, DCs accommodate a diverse set of workloads in terms of performance and resource requirements [3, 26], which typically change over time. Second, the DC operator often has no a priori knowledge of workload characteristics. Third, different workloads have widely varying sensitivities to interference due to contention in shared resources. Fourth, DC server platforms are heterogeneous, both in terms of configuration and server generation, which further complicates efficient cluster management. Finally, an additional challenge comes from the fact that during periods of adversarial traffic, i.e., intervals with very high load, the system can become oversubscribed, resulting in poor performance and QoS violations. Most DCs employ some admission control to minimize core oversubscription. However, these schemes typically do not differentiate applications and servers in terms of their sensitivity to interference and heterogeneity.

Recent work has shown how to address the four five challenges in the presence of unknown workloads, varying interference sensitivities and heterogeneous server platforms [17]. Paragon is a QoS-aware DC scheduler that accounts both for interference between co-scheduled applications, and heterogeneity in server platforms. It is designed as a recommendation engine that uses a minimal signal about a new workload and leverages the large amount of information the system has about previously-scheduled applications. Paragon uses collaborative filtering techniques, such as SVD, to classify applications based on the platforms they benefit from, and the type/amount of interference they tolerate and cause. Then, a greedy scheduler finds the best DC servers for a given workload. Paragon scales to tens of thousands of servers and applications with minimal overheads.

While Paragon addresses the challenge of "where" an application should be scheduled, it does not answer "when" it should be scheduled. Some applications have strict scheduling deadlines, which must be met, even when the system is oversubscribed. This can result in suboptimal scheduling, since no suitable servers are available. Other workloads, can tolerate scheduling delays in order to be assigned to servers of appropriate quality. In all cases, resource requirements should be taken into account at admission point, such that easy-to-satisfy workloads are not blocked behind demanding applications.

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 guarantees. Resource quality reflects the additional load a server can support without violating QoS, given its configuration and the applications it currently hosts. ARQ leverages the heterogeneity and interference information in the scheduler, to divide incoming applications in multiple classes and direct them to different workload queues. For example, workloads that need minimal interference to meet their QoS guarantees are added in one queue, while applications that can tolerate high interference in shared resources without performance degradation in a separate queue. This way a demanding workload will not block easy-to-satisfy applications, as it waits for a server of appropriate quality 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 also analyze the stability region of the multi-class queueing network.

We evaluate ARQ both in small and large-scale experiments. First, we compare the system without admission control and the system with ARQ, in a local cluster with 40 machines and show the benefits in performance and efficiency. While without admission control 36% of applications violate their QoS guarantees for an oversubscribed scenario; when applying ARQ, only 12% of applications experience a degradation higher than 5% compared to their ideal performance. Additionally, admission control preserves the efficiency benefits of Paragon. With Paragon, utilization increases by 47% of average compared to a scheduler that disallows colocation, while with Paragon and ARQ utilization still improves by 45.5%.

We also evaluate ARQ on a large-scale cloud provider. We use 1,000 exclusive instances on Amazon EC2 to run different workload scenarios. Again, Paragon with ARQ outperforms the system without admission control, especially for the case of an oversubscribed scenario, where 99% of workloads have less than 10% performance degradation, while maintaining the previous efficiency gains.

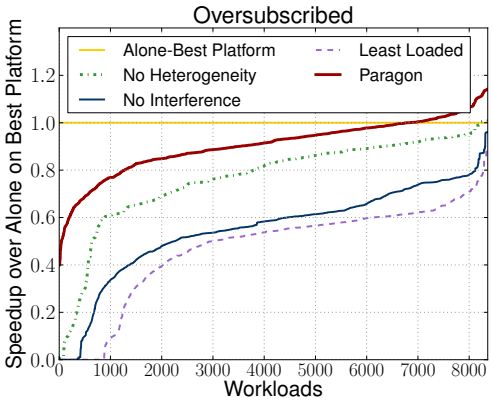

Figure 1: Performance achieved by different schedulers for an oversubscribed scenario with 8,500 applications on 1,000 EC2 servers. Paragon outperforms schedulers that ignore heterogeneity and/or interference, but still violates QoS for 39% of workloads.

2. Background

2.1. Paragon Overview

Paragon is a heterogeneity and interference-aware DC scheduler [17]. It assigns applications to heterogeneous servers based on the hardware platform they benefit from and the co-scheduled applications that minimize destructive interference and preserve QoS. Paragon has two components, a *classification engine* and a *greedy scheduler*. We briefly describe their operation in the following paragraphs.

The first component of Paragon performs fast and accurate 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 [6] or e-commerce. It leverages robust collaborative filtering methods, namely Singular Value Decomposition (SVD) and PQ-reconstruction and adds marginal overheads to the system, since it only requires a minimal signal about a new workload. Instead it takes advantage of the knowledge the system already has about previously-scheduled applications. Within two minutes of its arrival, an incoming workload is scheduled efficiently on a DC server.

The second component of Paragon is a greedy scheduler. The scheduler takes the classification results as input and searches in the server pool for a machine of desired SC, that minimizes destructive interference between existing and new load. In [17] we show that the scheduler can scale to tens of thousands of applications and can improve server utilization significantly, while maintaining per-application QoS constraints.

2.2. Current Limitations

While Paragon shows that accounting for heterogeneity and interference improves resource efficiency without QoS losses, there are still significant challenges that remain. Paragon accounts for the characteristics of a workload to decide in which server it should be assigned, but does not use this information to decide "when" the application should be admitted and scheduled. Instead, 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. Figure 1 shows the case of an oversubscribed scenario, where 8,500 applications are submitted in a 1,000-server EC2 cluster with inter-arrival times of 1 sec. Although Paragon outperforms schedulers that ignore heterogeneity, interference, or assign workloads to the least-loaded servers, it still violates QoS for several workloads. This happens as applications are sent to suboptimal servers that do not have sufficient resources to service them.

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, which can induce interference, resulting in serious performance losses.

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, in the case where no suitable servers for a given application are available, the system waits for a server

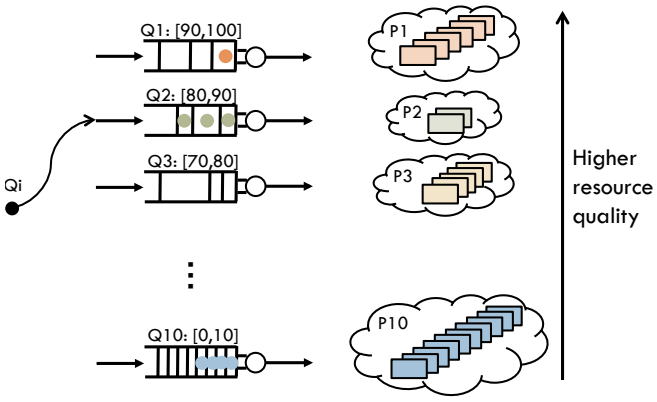

Figure 2: ARQ design. Each queue corresponds to applications with different resource quality requirements.

of appropriate quality to be freed before it schedules that workload. Alternatively, the application would be directed to the first free server to avoid queueing delays, with the risk of performance losses. In the discussion of ARQ's design we assume single-node applications for simplicity. The design holds for multi-node and distributed workloads with some changes as well.

Resource quality: The resource demands of a workload reflect the load a server should support such that the application meets its QoS constraints. 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 in the same machine. We use the classification engine in Paragon to derive the tolerated (t_i) and caused (c_i) interference of each server in the system on a set of shared resources. Shared resources include the cache and memory hierarchy, CPU modules, and storage and networking devices. Details on the method c_i's and t_i's are obtained can be found in [17]. The interference profile for a server is updated upon initiation or completion of an application's execution. Similarly, upon application arrival, an interference profile is obtained for each workload. This information guides the scheduling decisions by assigning applications to appropriate 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" [7, 29], 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 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. 2 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 derives the interference profile of an incoming workload, determines the class it belongs to and queues it appropriately. Each class has a corresponding pool of servers of appropriate resource quality that service the queue's applications. By separating applications based on their resource quality requirements, ARQ avoids bottlenecks where applications that are sensitive to interference block workloads that are not. However, limiting the resources an application can access to the subset of servers of the corresponding pool can also result in performance violations. DC workloads are driven by strict QoS and SLA guarantees and cannot be queued indefinitely waiting for a suitable server to be freed. 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. The more time it waits, the higher the chance it will be scheduled to a preferred server. At the same time, as waiting time increases, there is a higher chance the application will violate its QoS requirements. 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 diverged workload is assigned to one of the queue's available 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.

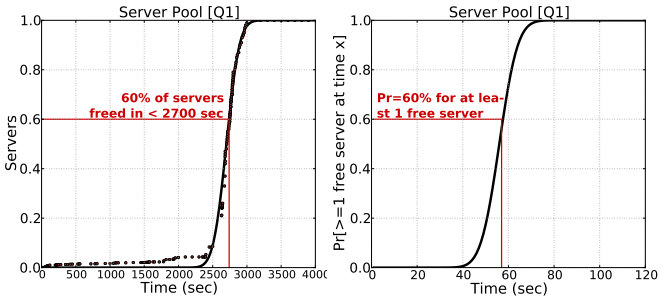

Figure 3: CDF of server busy times (Fig. 3a) and CDF of the probability that there will be at least one free server within a specific time window from an application's arrival (Fig. 3b).

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 an application 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 server busy-time per server pool in a closed form using known distributions. Fig. 3a 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 an interval after they have been allocated to an application. For example, 60% of servers in that 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. 3b 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.

Switching between queues: ARQ determines the switching point between queues to maximize the probability that a server will be available within a specific time window from an application's arrival. Let's assume for simplicity that the QoS constraint of an application is defined at 95% of the application's optimal performance. This means that the workload can tolerate at most a 5% performance degradation. Given the free-server probability CDFs for each server pool, ARQ solves the following optimization

problem for an application a, switching between queues i and j:

$$max\{(S_a - wt_i(t)) \cdot Q_i \cdot Pr_i[t], (S_a - wt_j(t)) \cdot Q_j \cdot Pr_j[t]\}$$
$$s.t.\ (wt_i(t) + wt_j(t) + P_a) < 0.05 \cdot CT_a \quad (2)$$

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 computation time for application a, P_a is the classification overheads 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 will preserve its QoS guarantees. It also promotes waiting longer for a server of the appropriate class rather than switching eagerly to the next queue ($Q_i > Q_j$).

3.3. Stability Analysis

A queueing network is stable if the total expected number of jobs in the network remains bounded as a function of time. Stability of multi-class queueing networks has been studied extensively [14, 15, 19, 24] both in the context of deterministic and stochastic arrival processes, using fluid limit models or Lyapunov functions. Here we briefly outline the conditions that guarantee stability in ARQ. Since each future state in the network depends only on the current, and not on past states, the system can be modeled as a Markov Chain (MC). Suppose λ_i is the external arrival rate for application class i and μ_i the corresponding service rate. P is the routing matrix, where P_{ij} the traffic diverged from class i to class j. Then the traffic equation is: $\bar{\lambda} = \lambda + P^T\bar{\lambda}$, with the explicit solution $\bar{\lambda} = [I - P^T]^{-1}\lambda$. In this formulation we assume for simplicity that diverged jobs are placed in the tail of the new queue as opposed to the head. This assumption does not affect that stability conditions of the network. Also, $\rho_S \triangleq \sum_{i \in S} \bar{\lambda}_i / \mu_i$ is the load factor of server S. For Poisson arrival and service processes, the stability of the Markov process is equated with positive (Harris) recurrence for the corresponding queue length process $Q(s) = (Q_i(s), s \geq 0)$. Assuming no jobs are dropped from the network, this reduces to the simple traffic condition: $\rho_S < 1\ \forall S$ to guarantee stability.

3.4. Additional Policies

ARQ can incorporate additional optimizations to prioritize application scheduling based on e.g., their expected *computation time* or their *priorities*. These optimizations are orthogonal to the principal design of the admission control protocol and may require additional information about the scheduled workloads.

[Computation time] Shortest Job First (SJF) is a well-known algorithm [39] that prioritizes the execution of short over long-running tasks. It improves the system's throughput by completing more tasks in a shorter time while preserving their corresponding QoS requirements. SJF can be implemented in ARQ. Jobs with a short expected computation time in each queue are scheduled before long-running jobs. However, given that the QoS requirements of every application must be preserved, SJF is applied with the additional constraint that scheduling of long-running jobs can only be delayed for as long as their QoS guarantees dictate.

[Priorities] ARQ can incorporate the concept of priorities by scheduling more critical applications first. Although the scheduler attempts to preserve QoS for all submitted workloads, in the event where only some workloads can meet their performance requirements the scheduler will prioritize the critical over the non-critical applications.

In Section 5 we show how ARQ behaves when implementing shortest-job-first and application priorities in the case of an oversubscribed workload scenario.

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

4. Methodology

Server systems: We evaluated Paragon on a small local cluster and a major cloud provider. Our local cluster includes servers of ten different configurations shown in Table 1. We also show how many servers of each type we use. Note that these configurations range from high-end Xeon systems to low-power Atom-based boards. There is a wide range of core counts, clock frequencies, and memory capacities and speeds present in the cluster.

For the cloud-based cluster we used exclusive (reserved) server instances on Amazon EC2, i.e., no other users had access to these servers and no interference from external workloads exists. We also verified that no external scheduling decisions or actions such as auto-scaling or workload migration are performed during the course of the experiments. We used 1,000 servers with 14 different SCs, ranging from small, low-power, dual-core machines to high-end, quad-socket, multi-core servers

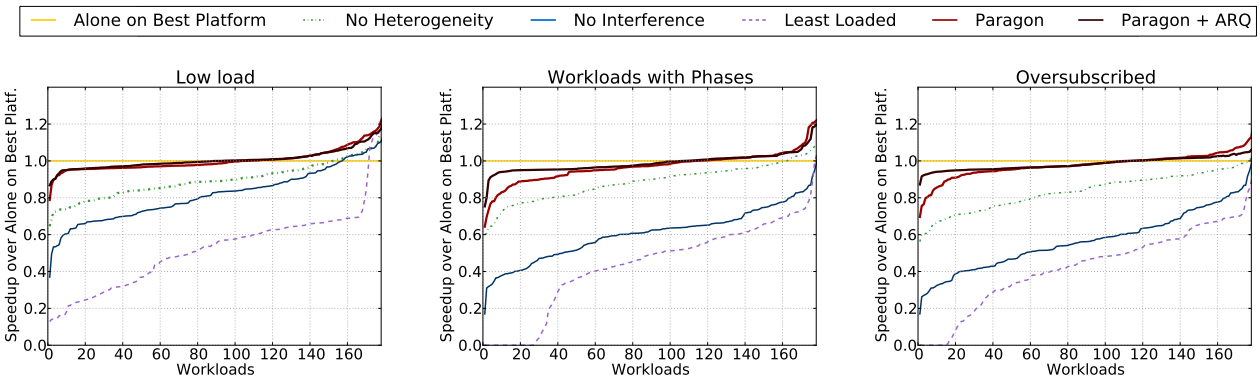

Figure 4: Performance impact from scheduling with Paragon and ARQ, across three workload scenarios, compared to Paragon without admission control, a heterogeneity-oblivious, an interference-oblivious and a least-loaded scheduler.

with hundreds of GBs of memory.

Schedulers: We compared Paragon with ARQ to four alternative schedulers. First, *Paragon* without the use of admission control, to isolate the benefits from using ARQ. Second, a *heterogeneity-oblivious* scheme that uses the interference classification to allocate servers but has no visibility of their SCs. Third, an *interference-oblivious* scheme that uses the heterogeneity classification but has no insight on workload interference. Finally, we evaluate a scheduler that is both heterogeneity and interference-agnostic, and assigns applications to the *least-loaded* machines in the system. The overheads for the heterogeneity and interference-oblivious schemes are the corresponding classification and server selection overheads.

Workloads: We used 29 single-threaded (ST), 22 multi-threaded (MT), 350 multi-programmed (MP) and 12 I/O-bound workloads. We use the full SPEC CPU2006 suite and workloads from PARSEC [8] (*blackscholes, bodytrack, facesim, ferret, fluidanimate, raytrace, swaptions, canneal*), SPLASH-2 [43] (*barnes, fft, lu, ocean, radix, water*), BioParallel [25] (*genenet, svm*), Minebench [30] (*semphy, plsa, kmeans*) and SPECjbb (*2, 4 and 8-warehouse instances*). For multiprogrammed workloads, we use 350 mixes of 4 applications each, based on the methodology described in [36]. The I/O-bound workloads are data mining applications in Hadoop and Matlab running on a single-node. For workload scenarios with more than 413 applications we replicated these workloads with equal likelihood (1/4 ST, 1/4 MT, 1/4 MP, 1/4 I/O) and randomized their interleaving.

Workload scenarios: To explore a wide range of behaviors, we used the applications listed above to create multiple workload scenarios. Scenarios vary in the number and type of submitted applications, their inter-arrival time and the existence or not of burstiness.

For the *small-scale* experiments on the local cluster we examine three workload scenarios. The load in the system is classified based on its relation to available resources; low: the required core count is significantly lower than the available processor resources; high: the required core count approaches the load the system can support but does not surpass it; and oversubscribed: the required core count often exceeds the system's capabilities, i.e., certain machines are oversubscribed. First, we examine a *low-load* scenario with 178 applications, selected randomly from the workload pool, which are submitted with 10 sec inter-arrival times. Second, a *high-load* scenario with 178 applications, where applications change *phases* twice throughout their execution, i.e., experience significant behavior changes. Workloads are submitted in the system following a Gaussian distribution with μ = 10 sec and σ^2 = 1.0. Finally, we examine a scenario, where 178 randomly-chosen applications arrive in the system with 1 sec intervals. Note that the last scenario is an *oversubscribed* one. After a few seconds, there are not enough servers (or cores) in the system to execute all applications concurrently.

For the *large-scale* experiments on EC2 we examine one workload scenario where 7,500 workloads are submitted with 1 sec intervals and an additional 1,000 applications arrive in burst (less than 0.1 sec intervals) after the first 3,750 workloads have been submitted. This results in an oversubscribed system, where the required cores are more than available in the cluster.

5. Evaluation

5.1. Small-scale Experiments

Performance: Fig. 4 shows the performance comparison between the different schedulers for the three workload scenarios in the small-scale cluster. We focus more on the differences between Paragon without and with the use of ARQ. Detailed evaluation on the differences between the other schedulers can be found in [17]. For the low-load scenario, where system resources are plentiful, the difference from the use of admission control is small. Without the use of ARQ, Paragon preserves QoS for 94% of applications, while with ARQ for 95%. The benefits from ARQ become more obvious as we move to scenarios

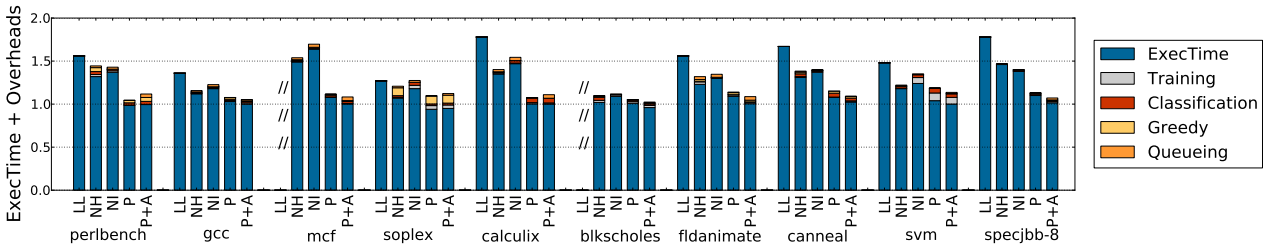

Figure 5: Overheads from classification, queueing and scheduling compared to useful execution time. Overall, the overheads in Paragon with ARQ are less than 5% for most applications.

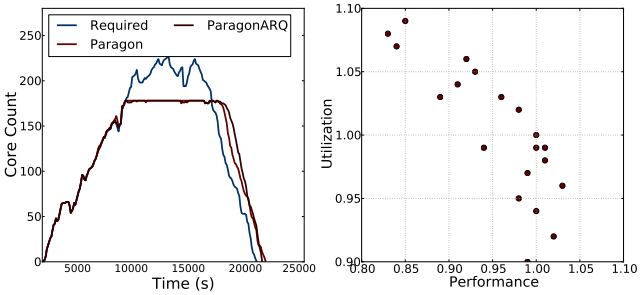

Figure 6: Required versus allocated core count for the oversubscribed scenario in the small-scale system (Fig. 6a) and sensitivity of ARQ to the number of queues. Performance and utilization are normalized to the values for 10 queues (Fig.6b).

Workloads	*hmean*	*standard deviation*
Shortest 5%	0.20%	0.15%
Shortest 10%	0.30%	0.08%
Shortest 25%	1.20%	0.30%
Shortest 50%	1.45%	0.34%
Shortest 75%	1.78%	0.26%
Shortest 90%	2.31%	0.55%
Shortest 95%	2.32%	0.57%
All	2.68%	0.53%

Table 2: Deviation between expected and achieved *computation time* for workloads in the oversubscribed scenario when ARQ implements SJF. Applications are ranked by increasing expected computation time.

of higher load. Both in the scenario where applications have phases, i.e., their behavior changes significantly at runtime and in the case of the oversubscribed system, using ARQ improves performance significantly. For the scenario with workload phases the applications that maintain their QoS requirements increase from 66% to 91%, and the average performance improves from 96% to 99.3%. Even more for the oversubscribed system, while without ARQ only 64% of applications maintain their QoS guarantees, with ARQ 88% preserve their performance requirements. This shows that queueing applications separately, and accounting for the resource quality they need, reduces the backlog of queued workloads much faster than when ignoring resource quality. In this experiment we do not apply the shortest job first (SJF) algorithm and the use of priorities for two reasons: first, to decouple the design of ARQ from specific scheduling policies and second, because all workloads have similar computation times and similar priorities. Adding these optimizations can further improve the performance benefits of ARQ. The other three schedulers (NH, NI and LL) have significantly lower performance than Paragon due to the fact that they do not account for platform heterogeneity and workload interference. Their differences become more intense for scenarios with higher loads.

Overheads: ARQ limits waiting time such that per-application QoS is preserved. Fig. 5 shows the breakdown of execution time for selected applications in the oversubscribed case. Time is divided in useful execution time, overheads from training and classification (first component of Paragon) [17], overheads from the greedy server selection (second component of Paragon) and overheads from queueing until a suitable server is freed. *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 is dominated by the useful execution of the workload, which for most workloads is very close to one (optimal execution time). The overheads from queueing are less than 5% of execution time at all times. The cases where queueing is high correspond to workloads that had to be diverged to queues with lower resource quality, in which case the useful execution time is also higher than 1 (suboptimal). This only happens for a very small fraction of workloads, even in the oversubscribed scenario. Also, there are a few cases, e.g., *perlbench*, where Paragon achieves lower total execution time than Paragon with ARQ. This typically happens when waiting for a suitable server does not result in better performance for the application (workload is insensitive to resource quality). Finally, there are some workloads for which the total overheads exceed 5% of execution time, e.g., *soplex*. This happens because the greedy server selection can be temporarily trapped in a local minimum between equally good selections. Although we solve this problem by inserting a timeout in the algorithm, it increases the total scheduling overheads in some cases. The greedy server selection happens after the application is

Workloads	*hmean*	*standard deviation*
High-priority (20%)	0.80%	0.06%
Low-priority (80%)	2.74%	0.32%

Table 3: Deviation between expected and achieved *completion time* for workloads in the oversubscribed scenario when ARQ implements priorities. Applications are grouped in high priority and low priority ones.

ready to be dispatched to a server, therefore its overhead is not accounted for by ARQ. We plan to address this as part of future work by preemptively selecting one of the available servers.

Resource allocation: Fig. 6a shows the required versus allocated core count for Paragon with and without ARQ for the oversubscribed scenario. During the ramp-up period ([0-9000]sec) both schedulers follow the resource requirements closely. However, once the system enters the oversubscribed phase ([9000-17000]sec), i.e., when the required cores are more than the total resources in the system, the two schedulers behave differently. While Paragon without ARQ allocates all available cores and then queues applications until the first server becomes available, Paragon with ARQ will only dispatch applications if an appropriate server is freed. This results in faster backlog draining since, even though applications are queued for longer, they run in higher quality platforms. This way, Paragon with ARQ exits the oversubscribed phase almost at the same time as Paragon without ARQ, while maintaining per-application QoS guarantees.

Server utilization: We also measure the average server utilization before and after the use of ARQ. We focus on the oversubscribed scenario where ARQ has the highest impact. Utilization for the other scenarios is similar with and without the use of admission control. Paragon without the use of ARQ already improves utilization by 47% compared to a least-loaded scheduler. Adding ARQ slightly reduces this improvement since applications that are queued due to lack of suitable servers will wait for a machine with the appropriate resource quality rather than the first available server. However, despite the queueing effects of ARQ, utilization for the oversubscribed scenario still improves by 45.5%. This means that the performance benefits from ARQ do not incur an efficiency penalty.

Sensitivity to design parameters: Fig. 6b shows how performance and utilization change when we vary the number of queues in ARQ. Both performance and utilization are normalized to the values for 10 queues. More queues result in more fine-grained application classification, and 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. Few queues, on the other hand, revert to the previous state of the scheduler where many applications are scheduled in FIFO order, increasing utilization and decreasing performance. We vary the number of queues from 1 to 20. For 10 queues we achieve both high performance and high efficiency. We use this number for all the experiments in this work.

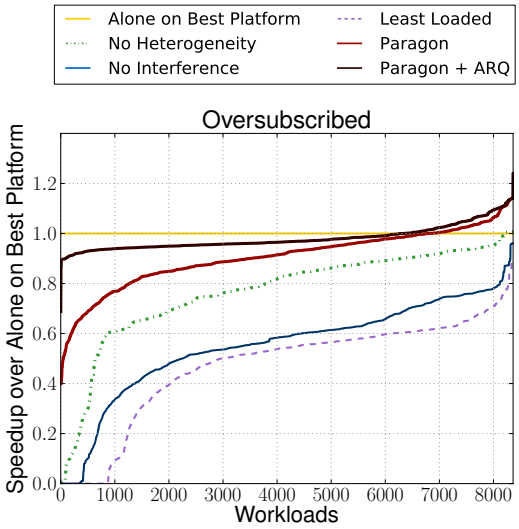

Figure 7: Performance achieved with the different schedulers for the oversubscribed scenario in 1,000 EC2 machines.

Additional policies: ARQ can implement policies such as Shortest Job First and application priorities. Table 2 shows the *harmonic mean* (hmean) and *standard deviation* of the difference between the expected and achieved computation time when ARQ implements SJF. In each queue the shortest applications are scheduled first, subject to the constraint that QoS is not violated for *any* application, regardless of its computation time. This means that long-running applications cannot be bypassed indefinitely. Workloads in the table are grouped based on their ideal computation time from short-running to long-running. Results are shown for the oversubscribed scenario. Short running jobs experience minimal performance degradation, while the long running applications have higher degradations, but still within their QoS requirements (5% execution time increase).

Additionally, we evaluate ARQ in the presence of workload priorities. We set "high priority" for 20% of workloads in the oversubscribed scenario and compare the expected and achieved completion time for them (see table 3). As seen in the table, the high-priority workloads complete within 2% of their ideal completion time. Low-priority applications also complete within their QoS constraints, in most cases, but experience higher performance degradations than high-priority workloads.

Large-scale experiments: Fig. 7 shows the performance

comparison for the different schedulers for the large-scale scenario. Applications are again ordered from worst to best-performing. There are 1,000 servers in the system and 8,500 applications submitted over a period of 2 hours, following the inter-arrival times specified in Section 4. While Paragon without ARQ only preserves QoS for 61% of workloads, introducing admission control increases that fraction to 83% of applications. Additionally, it bounds performance degradation for 99% of workloads to less than 10%. 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. Related Work

We discuss work related to ARQ in terms of admission control in computer systems and analysis of multi-class queueing networks.

Admission control systems: A lot of work has highlighted the importance of admission control in computer systems, including datacenters (DCs). Cherkasova et al. [12, 13] propose a predictive and a session-based admission control scheme respectively for overloaded web servers. The schemes monitor the utilization and QoS achieved at runtime and preemptively adjust the admission policy to more or less aggressive, such that QoS is preserved. In the same spirit, Bartolini et al. [5] propose a self-configurable overload control policy that adjusts the rate of admitted sessions to preserve SLAs and improve utilization. Liu et al. [28] propose an adaptive scheme based on queueing theory to control the performance of multi-tier web applications. Carlstrom et al. [9] also design a session-based admission control protocol for web servers that leverages generalized processor sharing (GPS) [31] to maximize a reward function that corresponds to the rate of completed jobs. Similarly, Salehi et al. [35] propose a preemption-aware admission control system for virtualized systems, where the system services both internal and external requests, with the internal requests having preemptive priority over external requests. The scheme maximizes the rate of admitted requests, subject to preserving per-application SLAs. Cheng et al. [11] also divide the application space to high and low-priority workloads and partition the server's capacity to service workloads with different priorities. The authors propose a threshold-based admission control algorithm where thresholds depend on the application's priority, and rewards are higher for critical versus non-critical applications. Finally, Guitart et al. [22] consider the problem of admission control in the context of a secure web application and propose an adaptive overload control strategy based on SSL connection differentiation.

Techniques such as predictive admission control [12, 28], protection against DoS attacks, or schemes that additionally account for application security at admission control [22], are orthogonal to the design of ARQ, and can be incorporated in the scheme if the corresponding functionality is desired.

Multi-class queueing networks: Multiclass queueing networks have applications in a wide spectrum of systems ranging from banks, to product lines and network systems. Miller [29] analyzes a multi-class queueing network that optimizes the rewards obtained by accepting or rejecting customers in a system with multiple customer classes. Bertsimas et al. [7] study the distribution of steady-state queue lengths for a multi-class markovian queueing network and propose a methodology based on Lyapunov functions for the performance analysis of MCs with infinite states, including multi-class queueing networks. Kulkarni et al. [27] examine an admission control protocol for multi-class traffic with service priorities in high-speed networks. They assign different size buffers to each class and derive policies to guarantee per-class QoS. Stolyar [38] discusses the stability of multi-class queueing networks, whose stochastic process is a continuous time MC. He shows that the sequence of underlying stochastic processes converges to a fluid process with sample paths defined as fixed points of a special operator and defines the conditions under which the network is stable. In the same context, Chen [10] studies the fluid approximation and stability of a multi-class queueing network.

Gurvich [23] provides an overview of the design and control of multi-class queueing networks (M/M/N queues with multiple types of customers and many servers). He analyzes the V-Model of skills-based routing, and examines how different customer classes are scheduled to servers and how many servers are required to minimize staffing and waiting costs. Sethuraman et al. [37] propose that globally optimal scheduling for a multi-class system with parallel queues reduces to finding the optimal routing matrix under the assumption that the optimal sequencing strategy for each server is a simple static priority policy. Atar et al. [1] also consider asymptotic optimality in a multi-class queueing system with many exponential servers, under the presence of heavy traffic.

In the context of computer systems, Gemikonakli et al. [20] model the performance of a virtualized server using a multi-class M/M/1 queueing model, where applications of different rates arrive in each queue. They analyze the stability, backlog and throughput of the system using an MC model. In a system that resembles a multi-class queue, Yolken et al. [46] propose a game-based capacity allocation system, where each client receives service rate proportional to the bid on resources he submitted to the system operator. Each client has a flow of jobs and although applications are serviced in a FCFS manner, service rates vary across jobs.

7. Conclusions

We have presented ARQ, a QoS-aware admission control protocol for heterogeneous datacenters that complements Paragon. ARQ divides applications to classes based on their resource quality requirements and queues them separately in a multi-class queueing network. ARQ is derived from validated queueing models, and it improves the system's throughput by allowing easy-to-satisfy workloads to be serviced before demanding applications. It preserves each application's QoS requirements by limiting waiting time, and diverging workloads to other queues when necessary. ARQ can also be enhanced with optimizations, such as SJF and priority-aware scheduling. We have evaluated Paragon with ARQ in both small and large-scale experiments and have shown that for 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.

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] N. Bartolini, G. Bongiovanni, S. Silvestri. "Self-overload Control for Fistributed Web Systems". *In Proc. of the International Workshop on Quality of Service (IWQoS), Enschede, 2008*.

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

[7] 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*.

[8] 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*.

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

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

[11] 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*.

[12] L. Cherkasova, P. Phaal. "Predictive Admission Control Strategy for Overloaded Commercial Web Server". *In Proc. of the international symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), San Francisco, CA, 2000*.

[13] L. Cherkasova, P. Phaal. "Session-Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites". *In IEEE Transactions on Computers, Vol. 51, No. 6, 2002*.

[14] 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*.

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

[16] 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*.

[17] 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*.

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

[19] D. Gamarnik, D. Katz. "On Deciding Stability of Multiclass Queueing Networks Under Buffer Priority Scheduling Policies". '*In Annals of Applied Probability, 5:2008-2037, 2009*.

[20] O. Gemikonakli, E. Ever, E. Gemikonakli. "Performance Modelling of Virtualized Servers". *In Proc. of Computer Modelling and Simulation Conference (UKSim), UK, 2010*.

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

[22] J. Guitart, D. Carrera, V. Beltran, J. Torres, E. Ayguade. "Designing an overload control strategy for secure e-commerce applications". *In Computer Networks 51, 2007*.

[23] I. Gurvich. "Design and Control of the M/M/N Queue with Multi-Type Customers and Many Servers". *M.S. Thesis, Technion Israel Institute of Technology, 2004*.

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

[25] 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*.

[26] 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*.

[27] 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*.

[28] X. Liu, J. Heo, L. Sha, X. Zhu. "Adaptive Control of Multi-Tiered Web Applications Using Queueing Predictor". *In IEEE Transactions on Network and Service Management, Sept. 2008*.

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

[30] R. Narayanan, B. Ozisikyilmaz, et al. "MineBench: A Benchmark Suite for DataMining Workloads". *In Proc. of the IEEE Int'l Symposium on Workload Characterization (IISWC), SJ, 2006*.

[31] A. Parekh, R. Gallager. "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case". *In IEEE Transactions on Networks, Vol. 1, No. 3, June 1993*.

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

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

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

[35] M. A. Salehi, B. Javadi, R. Buyya. "Preemption-aware Admission Control in a Virtualized Grid Federation". *In Proc. of the international conference on Advanced Information Networking and Applications (AINA), Japan, 2012*.

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

[37] J. Sethuraman, M. Squillante. "Optimal Stochastic Scheduling in Multiclass Parallel Queues". *In Proc. of SIGMETRICS, 1999*.

[38] A. Stolyar. "On the Stability of Multiclass Queueing Networks: A Relaxed Sufficient Condition via Limiting Fluid Processes". Tech. Report.

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

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

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

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

[43] 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*.

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

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

[46] B. Yolken, N. Bambos. "Game Based Capacity Allocation for Utility Computing Environments". *In Proc. of Gamecomm, 2008*.