Quality-of-Service-Aware Scheduling in Heterogeneous Datacenters with Paragon

Paragon, an online, scalable datacenter scheduler, enables better cluster utilization and per-application quality-of-service guarantees by leveraging data mining techniques that find similarities between known and new applications. For a 2,500-workload scenario, Paragon preserves performance constraints for 91 percent of applications, while significantly improving utilization. In comparison, a baseline least-loaded scheduler only provides similar guarantees for 3 percent of workloads.

Christina Delimitrou
Christos Kozyrakis
Stanford University

Efficiency is a first-class requirement and the main source of scalability concerns both for small and large systems.[1,2] Achieving high efficiency is not only a matter of sensible design, but also a function of how the system is managed, which becomes essential as the hardware grows progressively heterogeneous and parallel and applications get dynamic and diverse. Architecture has traditionally been about efficient system design. As efficiency increases in importance, architecture should be about both design and management for systems of any scale.

In this article, we focus on improving efficiency while guaranteeing high performance in large-scale systems. Although an increasing amount of computing now happens in public and private clouds, such as Amazon Elastic Compute Cloud (EC2; see http://aws.amazon.com/ec2) or vSphere (www.vmware.com/products/vsphere), datacenters continue to operate at utilizations in the single digits.[1,3] This lessens the two main advantages of cloud computing—flexibility and cost efficiency both for cloud operators and end users—because not only are the machines underutilized, they are also operating in a non-energy-proportional region.[1,4]

There can be several reasons why machines are underutilized. Two of the most prominent obstacles are interference between coscheduled applications and heterogeneity in server platforms. For more information, see the "Interference and Heterogeneity" sidebar.

In our paper presented at the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2013),[5] we introduced Paragon, an online and scalable

Published by the IEEE Computer Society

Interference and Heterogeneity

Interference occurs as coscheduled applications contend in shared resources. Coscheduled applications may interfere negatively even if they run on different processor cores because they share caches, memory channels, storage, and networking devices.[1,2] If unmanaged, interference can result in performance degradations of integer factors,[2] especially when the application must meet tail latency guarantees apart from average performance.[3] Figure A shows that an interference-oblivious scheduler will slow workloads down by 34 percent on average, with some running more than two times slower. This is undesirable for both users and operators.

Heterogeneity is the natural result of the infrastructure's evolution, as servers are gradually provisioned and replaced over the typical 15-year lifetime of a datacenter.[4-7] At any point in time, a datacenter may host three to five server generations with a few hardware configurations per generation, in terms of the processor speed, memory, storage, and networking subsystems. Managing the different hardware incorrectly not only causes significant performance degradations to applications sensitive to server configuration, but also wastes resources as workloads occupy servers for significantly longer, and gives a low-quality signal to hardware vendors for the design of future platforms. Figure A shows that a heterogeneity-oblivious scheduler will slow applications down by 22 percent on average, with some running nearly 2 times slower (see the "Methodology" section in the main article).

Finally, a baseline scheduler that is oblivious to both interference and heterogeneity and which schedules applications to least-loaded servers is even worse (48 percent average slowdown), causing some workloads to crash due to resource exhaustion on the server. Unless interference and heterogeneity are managed in a coordinated fashion, the system loses both its efficiency and predictability guarantees. Previous research has identified the issues of heterogeneity[6] and interference,[2] but while most cloud management systems—such as Mesos[8] or vSphere (www.vmware.com/products/vsphere)—have some notion of contention or interference awareness, they either use empirical rules for interference management or assume long-running workloads (for example, online services), whose repeated behavior can be progressively modeled. In this article, we target both heterogeneity and interference and assume no a priori analysis of the application. Instead, we leverage information the system already has about the large number of applications it has previously seen.

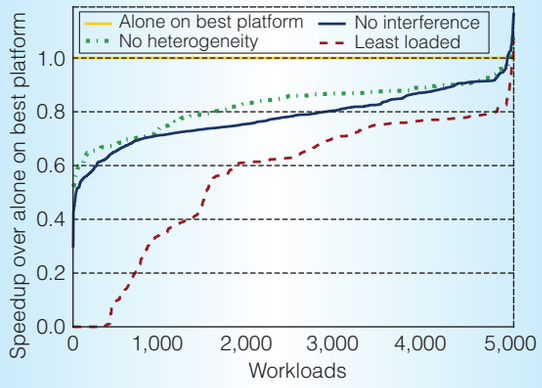

Figure A. Performance degradation for 5,000 applications on 1,000 Amazon Elastic Compute Cloud (EC2) servers with heterogeneity-oblivious, interference-oblivious, and baseline least-loaded schedulers compared to ideal scheduling (application runs alone on best platform). Results are ordered from worst to best-performing workload.

References

1. S. Govindan et al., "Cuanta: Quantifying Effects of Shared On-Chip Resource Interference for Consolidated Virtual Machines," *Proc. 2nd ACM Symp. Cloud Computing*, 2011, article no. 22.
2. J. Mars et al., "Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations," *Proc. 44th Ann. IEEE/ACM Int'l Symp. Microarchitecture*, 2011, pp. 248-259.
3. D. Meisner et al., "Power Management of Online Data-Intensive Services," *Proc. 38th Ann. Int'l Symp. Computer Architecture* (ISCA 11), 2011, pp. 319-330.
4. L.A. Barroso and U. Holzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, Morgan and Claypool Publishers, 2009.
5. C. Kozyrakis et al., "Server Engineering Insights for Large-Scale Online Services," *IEEE Micro*, vol. 30, no. 4, 2010, pp. 8-19.
6. J. Mars, L. Tang, and R. Hundt, "Heterogeneity in 'Homogeneous' Warehouse-Scale Computers: A Performance Opportunity," *IEEE Computer Architecture Letters*, vol. 10, no. 2, 2011, pp. 29-32.
7. R. Nathuji, C. Isci, and E. Gorbatov, "Exploiting Platform Heterogeneity for Power Efficient Data Centers," *Proc. 4th Int'l Conf. Autonomic Computing* (ICAC 07), 2007, doi:10.1109/ICAC.2007.16.
8. B. Hindman et al., "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center," *Proc. 8th USENIX Conf. Networked Systems Design and Implementation*, 2011, article no. 22.

datacenter scheduler that accounts for heterogeneity and interference. The key feature of Paragon is its ability to quickly and accurately classify an unknown application with respect to heterogeneity (which server configurations it will perform best on) and interference (how much interference it will cause to coscheduled applications and how much interference it can tolerate itself in multiple shared resources). Unlike previous techniques that require detailed profiling of each incoming application, Paragon's classification engine exploits existing data from previously scheduled workloads and requires only a minimal signal about a new workload. Specifically, it is organized as a low-overhead recommendation system similar to the one deployed for the Netflix Challenge,[6] but instead of discovering similarities in users' movie preferences, it finds similarities in applications' preferences with respect to heterogeneity and interference. It uses singular value decomposition (SVD) to perform collaborative filtering and identify similarities between incoming and previously scheduled workloads.

Once an incoming application is classified, a greedy scheduler assigns it to the server that is the best possible match in terms of platform and minimum negative interference between all coscheduled workloads. Even though the final step is greedy, the high accuracy of classification leads to schedules that achieve both fast execution time and efficient resource usage. Paragon scales to systems with tens of thousands of servers and tens of configurations, running large numbers of previously unknown workloads. We implemented Paragon and showed that it significantly improves cluster utilization, while preserving per-application quality-of-service (QoS) guarantees both for small- and large-scale systems. For more information on related work, see the "Research Related to Paragon" sidebar.

Fast and accurate classification

The key requirement for heterogeneity and interference-aware scheduling is to quickly and accurately classify incoming applications. First, we need to know how fast an application will run on each of the tens of server configurations (SCs) available. Second, we need to know how much interference it can tolerate from other workloads in each of several shared resources without significant performance loss and how much interference it will generate itself. Our goal is to perform online scheduling for large-scale systems without any a priori knowledge about incoming applications. Most previous schemes address this issue with detailed but offline application characterization or long-term monitoring and modeling.[7-9] Paragon takes a different approach. Its core idea is that, instead of learning each new workload in detail, the system leverages information it already has about applications it has seen to express the new workload as a combination of known applications. For this purpose, we use collaborative filtering techniques that combine a minimal profiling signal about the new application with the large amount of data available from previously scheduled workloads. The result is fast and accurate classification of incoming applications with respect to heterogeneity and interference. Within a minute of its arrival, an incoming workload is scheduled on a large-scale cluster.

Background on collaborative filtering

Collaborative filtering techniques are frequently used in recommendation systems. We use one of their most publicized applications, the Netflix Challenge,[6] to provide a quick overview of the two analytical methods we rely on, SVD and PQ reconstruction.[10] In this case, the goal is to provide valid movie recommendations for Netflix users given the ratings they have provided for various other movies.

The input to the analytical framework is a sparse matrix A, the *utility matrix*, with one row per user and one column per movie. The elements of A are the ratings that users have assigned to movies. Each user has rated only a small subset of movies; this is especially true for new users, who might only have a handful of ratings, or even none. Although techniques exist that address the cold-start problem (that is, providing recommendations to a completely fresh user with no ratings), we focus here on users for whom the system has some minimal input. If we can estimate the values of the missing ratings in the sparse matrix A,

Research Related to Paragon

We discuss work relevant to Paragon in the areas of datacenter scheduling, virtual machine (VM) management, workload rightsizing, and scheduling for heterogeneous multicore chips.

Datacenter scheduling

Recent work on datacenter scheduling has highlighted the importance of platform heterogeneity and workload interference. Mars et al. showed that the performance of Google workloads can vary by up to 40 percent because of heterogeneity, even when considering only two server configurations, and by up to 2 times because of interference, even when considering only two colocated applications.[1,2] Govindan et al. also present a scheme to quantify the effects of cache interference between consolidated workloads.[3] In Paragon, we extend the concepts of heterogeneity- and interference-aware scheduling by providing an online, scalable, and low-overhead methodology that accurately classifies applications for both heterogeneity and interference across multiple resources.

VM management

Systems such as vSphere (http://www.vmware.com/products/vsphere) or the VM platforms on public cloud providers can schedule diverse workloads submitted by users on the available servers. In general, these platforms account for application resource requirements that they expect the user to express or they learn over time by monitoring workload execution. Paragon can complement such systems by making scheduling decisions on the basis of heterogeneity and interference and detecting when an application should be considered for rescheduling.

Resource management and rightsizing

There has been significant work on resource allocation in virtualized and nonvirtualized large-scale datacenters. Mesos performs resource allocation between distributed computing frameworks such as Hadoop or Spark.[4] Rightscale (http://www.rightscale.com) automatically scales out three-tier applications to react to changes in the load in Amazon's cloud service. DejaVu serves a similar goal by identifying a few workload classes and, based on them, reusing previous resource allocations to minimize reallocation overheads.[5] In general, Paragon is complementary to rightsizing systems. Once such a system determines the amount of resources needed by an application, Paragon can classify and schedule it on the proper hardware platform in a way that minimizes interference.

Scheduling for heterogeneous multicore chips

Scheduling in heterogeneous CMPs shares some concepts and challenges with scheduling in heterogeneous datacenters; thus, some of the ideas in Paragon can be applied in heterogeneous CMP scheduling as well. Shelepov et al. present a scheduler for heterogeneous CMPs that is simple and scalable,[6] whereas Craeynest et al. use performance statistics to estimate which workload-to-core mapping is likely to provide the best performance.[7] Given the increasing number of cores per chip and coscheduled tasks, techniques similar to the ones used in Paragon can be applicable when deciding how to schedule applications in heterogeneous CMPs as well.

References

1. J. Mars, L. Tang, and R. Hundt, "Heterogeneity in 'Homogeneous' Warehouse-Scale Computers: A Performance Opportunity," *IEEE Computer Architecture Letters*, vol. 10, no. 2, 2011, pp. 29-32.
2. J. Mars et al., "Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations," *Proc. 44th Ann. IEEE/ACM Int'l Symp. Microarchitecture*, 2011, pp. 248-259.
3. S. Govindan et al., "Cuanta: Quantifying Effects of Shared On-Chip Resource Interference for Consolidated Virtual Machines," *Proc. 2nd ACM Symp. Cloud Computing*, 2011, article no. 22.
4. B. Hindman et al., "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center," *Proc. 8th USENIX Conf. Networked Systems Design and Implementation*, 2011, article no. 22.
5. N. Vasic et al., "DejaVu: Accelerating Resource Allocation in Virtualized Environments," *Proc. 17th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 2012, pp. 423-436.
6. D. Shelepov et al., "HASS: A Scheduler for Heterogeneous Multicore Systems," *ACM SIGOPS Operating Systems Rev.*, vol. 43, no. 2, 2009, pp. 66-75.
7. K. Craeynest et al., "Scheduling Heterogeneous Multi-Cores through Performance Impact Estimation (PIE)," *Proc. 39th Ann. Int'l Symp. Computer Architecture* (ISCA 12), 2012, pp. 213-224.

we can make movie recommendations; that is, we can suggest that users watch the movies for which the recommendation system estimates they will give high ratings to with high confidence.

The first step is to apply SVD, a matrix factorization method used for dimensionality reduction and similarity identification. Factoring A produces the decomposition to the following matrices of left (U) and right (V)

singular vectors and the diagonal matrix of singular values (Σ):

$$A_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$
$$= U \cdot \Sigma \cdot V^T \text{where}$$
$$U_{m\times r} = \begin{pmatrix} u_{1,1} & \cdots & u_{1,r} \\ \vdots & \ddots & \vdots \\ u_{m,1} & \cdots & u_{m,r} \end{pmatrix},$$
$$V_{n\times r} = \begin{pmatrix} v_{1,1} & \cdots & v_{1,r} \\ \vdots & \ddots & \vdots \\ v_{n,1} & \cdots & v_{n,r} \end{pmatrix},$$
$$\Sigma_{r\times r} = \begin{pmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_r \end{pmatrix}$$

Dimension r is the rank of matrix A, and it represents the number of similarity concepts identified by SVD. For instance, one similarity concept might be that certain movies belong to the drama category, while another might be that most users who liked the movie *The Lord of the Rings: The Fellowship of the Ring* also liked *The Lord of the Rings: The Two Towers*. Similarity concepts are represented by singular values (σ_i) in matrix Σ and the confidence in a similarity concept by the magnitude of the corresponding singular value. Singular values in Σ are ordered by decreasing magnitude. Matrix U captures the strength of the correlation between a row of A and a similarity concept. In other words, it expresses how users relate to similarity concepts such as the one about liking drama movies. Matrix V captures the strength of the correlation of a column of A to a similarity concept. In other words, to what extent does a movie fall in the drama category? The complexity of performing SVD on a $m \cdot n$ matrix is $\min(n^2m,\ m^2n)$. SVD is robust to missing entries and imposes relaxed sparsity constraints to provide accuracy guarantees.

Before we can make accurate score estimations using SVD, we need the full utility matrix A. To recover the missing entries in A, we use PQ reconstruction. Building from the decomposition of the initial sparse A matrix, we have $Q_{m\times r} = U$ and $P^T_{r\times n} = \sum \cdot V^T$. The product of Q and P^T gives matrix R, which is an approximation of A with the missing entries. To improve R, we use stochastic gradient descent (SGD), a scalable and lightweight latent-factor model that iteratively recreates A:

$\forall r_{ui}$, where r_{ui} is an element of the reconstructed matrix R

$$\in_{ui} = r_{ui} - q_i \cdot p_u^T$$
$$q_i \leftarrow q_i + \eta(\in_{ui} p_u - \lambda q_i)$$
$$p_u \leftarrow p_u + \eta(\in_{ui} q_i - \lambda p_u)$$

until $|\in|_{L_2} = \sqrt{\sum_{u,i} |\in_{ui}|^2}$ becomes marginal.

In this process, η is the learning rate and λ is the regularization factor. The complexity of PQ is linear with the number of r_{ui} and in practice takes up to a few milliseconds for matrices whose m and n equal about 1,000. Once the dense utility matrix R is recovered, we can make movie recommendations. This involves applying SVD to R to identify which of the reconstructed entries reflect strong similarities that enable making accurate recommendations with high confidence.

Classification for heterogeneity

We use collaborative filtering to identify how well a previously unknown workload will run on different hardware platforms. The rows in matrix A represent applications, the columns represent server configurations (SCs), and the ratings represent normalized application performance on each SC. As part of an offline step, we select a small number of applications and profile them on all the different SCs. This provides some initial information to the classification engine to address the cold-start problem that would otherwise occur. It only needs to happen once in the system.

During regular operation, when an application arrives, we profile it for 1 minute on any two SCs, insert it as a new row in matrix A, and use the process described previously to derive the missing ratings for the other server configurations. In this case, Σ represents similarity concepts such as the fact that applications that benefit from SC1 will also benefit

from SC3. *U* captures how an application correlates to the different similarity concepts, and *V* shows how an SC correlates to them. Collaborative filtering identifies similarities between new and known applications. Two applications can be similar in one characteristic (for instance, they both benefit from high clock frequency) but different in others (for example, only one benefits from a large L3 cache). This is especially common when scaling to large application spaces and hardware configurations. SVD addresses this issue by uncovering hidden similarities and filtering out the ones less likely to have an impact on the application's behavior.

As incoming applications are added in *A*, the density of the matrix increases and the recommendation accuracy improves. Note that online training is performed only on two SCs. This reduces the training overhead and the number of servers needed for it compared to exhaustive search. In contrast, if we attempted an exhaustive application profiling, the number of profiling runs would equal the number of SCs. For a cloud service with high workload arrival rates, this would be infeasible to support. On a production-class Xeon server, classification takes 10 to 30 milliseconds for thousands of applications and tens of SCs. We can perform classification for one application at a time or for small groups of incoming applications (batching) if the arrival rate is high without impacting accuracy or speed.

Performance scores. We use the following performance metrics according to the application type:

- *Single-threaded workloads*: We use instructions committed per second (IPS) as the initial performance metric. Using execution time would require running applications to completion during profiling, increasing overheads. We have verified that IPS leads to similar classification accuracy as using time to completion. For multiprogrammed workloads, we use aggregate IPS.
- *Multithreaded workloads*: In the presence of spinlocks or other synchronization schemes, IPS can be deceptive. We address this by detecting active waiting and weight such execution segments out of the IPS computation. We verified that using this "useful" IPS leads to similar classification accuracy as using the full execution time.

The choice of IPS is influenced by our current evaluation, which focuses on single-node CPU-, memory-, and I/O-intensive programs. The same methodology can be extended to higher-level metrics, such as queries per second (QPS), which cover complex multitier workloads as well.

Validation. We evaluate the accuracy of heterogeneity classification on a 40-server cluster with 10 SCs with a large set of diverse applications. The offline training set includes 20 randomly selected applications. Using the classification output for scheduling improves performance by 24 percent for single-threaded workloads, 20 percent for multithreaded workloads, 38 percent for multiprogrammed workloads, and 40 percent for I/O workloads, on average, while some applications have a 2× performance difference. Table 1 summarizes key statistics on the validation study. It is important to note that the accuracy does not depend on the SCs selected for training, which matched the top-performing configuration only for 20 percent of workloads. We also compare performance predicted by the recommendation system to performance obtained through experimentation. The deviation is 3.8 percent on average.

Classification for interference

We are interested in two types of interference: that which an application can tolerate from preexisting load on a server, and that which the application will cause on that load. We detect interference due to contention and assign a score to the sensitivity of an application to a type of interference. To derive sensitivity scores, we develop several microbenchmarks (sources of interference, or SoIs), each stressing a specific shared resource with tunable intensity.[11] SoIs span the core, memory, and cache hierarchy and network and storage bandwidth. We run an application concurrently with a microbenchmark

Table 1. Validation of heterogeneity classification.

Metric	Applications			
	Single threaded (%)	Multithreaded (%)	Multiprogrammed (%)	I/O bound (%)
Selected best platform	86	86	83	89
Selected platform within 5% of best	91	90	89	92
Correct platform ranking (best to worst)	67	62	59	43
90% correct platform ranking	78	71	63	58
Training and best selected platform match	28	24	18	22

Table 2. Validation of interference classification.

Metric	Percentage (%)
Average estimation error of sensitivity across all examined resources	5.3
Average estimation error for sensitivities > 60%	3.4
Applications with < 5% estimation error	59.0
Resource with highest estimation error: L1 instruction cache	15.8
Frequency L1 instruction cache used for training	14.6
Resource with lowest estimation error: Storage bandwidth	0.9

and progressively tune up its intensity until the application violates its QoS. Applications with high tolerance to interference (for example, a sensitivity score over 60 percent) are easier to coschedule than applications with low tolerance. Similarly, we detect the sensitivity of a microbenchmark to the interference the application causes by tuning up its intensity and recording when the microbenchmark's performance degrades by 5 percent compared to its performance in isolation. In this case, high-sensitivity scores correspond to applications that cause a lot of interference in the specific shared resource.

Collaborative filtering for interference. We classify applications for interference tolerated and caused, using twice the process described earlier. The two utility matrices have applications as rows and SoIs as columns. The elements of the matrices are the sensitivity scores of an application to the corresponding microbenchmark. Similarly to classification for heterogeneity, we profile a few applications offline against all SoIs and insert them as dense rows in the utility matrices. In the online mode, each new application is profiled against two randomly chosen microbenchmarks for one minute, and its sensitivity scores are added in a new row in each of the matrices. Then, we use SVD and PQ reconstruction to derive the missing entries and the confidence in each similarity concept.

Validation. We evaluated the accuracy of interference classification using the same workloads and systems as before. Table 2 summarizes key statistics on the classification quality. The average error in estimating both tolerated and caused interference across SoIs is 5.3 percent. For high values of sensitivity (that is, applications that tolerate and cause a lot of interference), the error is even lower (3.4 percent).

Putting it all together

Overall, Paragon requires two short runs (approximately 1 minute) on two SCs to classify incoming applications for heterogeneity. Another two short runs against two microbenchmarks on a high-end SC are needed for interference classification. Running for 1 minute provides some signal on the new workload without introducing significant

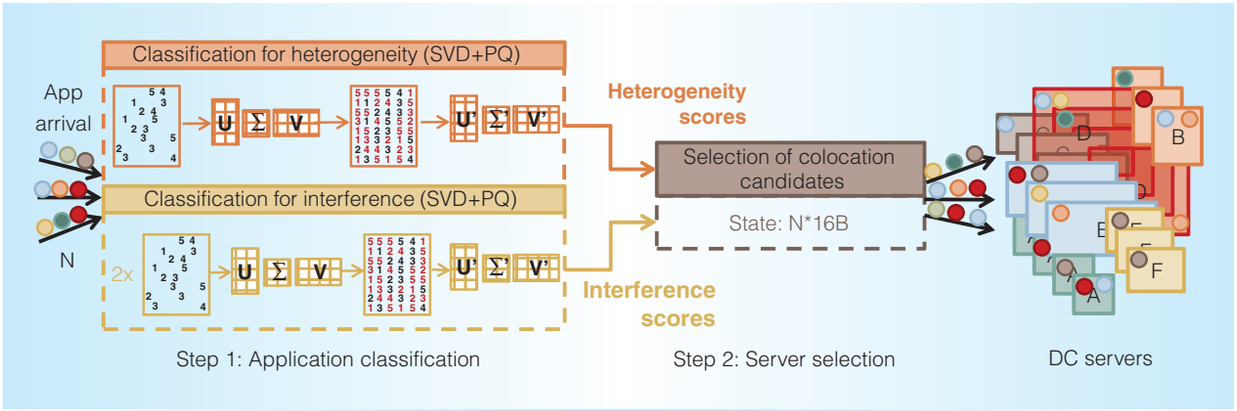

Figure 1. The components of Paragon and the state maintained by each component. Overall, the state requirements are marginal and scale linearly or logarithmically with the number of applications (*N*), servers (*M*), and configurations. (PQ: PQ reconstruction; SVD: singular value decomposition; DC: datacenter.)

profiling overheads. In our full paper,[5] we discuss the issue of workload phases (that is, transient effects that do not appear in the 1-minute profiling period). Next, we use collaborative filtering to classify the application in terms of heterogeneity and interference. This requires a few milliseconds even when considering thousands of applications and several tens of SCs or SoIs. Classification for heterogeneity and interference is performed in parallel. For the applications we considered, the overall profiling and classification overheads are 1.2 and 0.09 percent on average.

Using analytical methods for classification has two benefits. First, we have strong analytical guarantees on the quality of the information used for scheduling, instead of relying mainly on empirical observation. The analytical framework provides low and tight error bounds on the accuracy of classification, statistical guarantees on the quality of colocation candidates, and detailed characterization of system behavior. Moreover, the scheduler design is workload independent, which means that the properties the scheme provides hold for any workload. Second, these methods are computationally efficient, scale well with the number of applications and SCs, and do not introduce significant scheduling overheads.

Paragon

Once an incoming application is classified with respect to heterogeneity and interference, Paragon schedules it on one of the available servers. The scheduler attempts to assign each workload to the server of the best SC and colocate it with applications so that interference is minimized for workloads running on the same server.

Scheduler design

Figure 1 presents an overview of Paragon's components and operation. The scheduler maintains per-application and per-server state. The per-application state includes the classification information; for a datacenter with 10 SCs and 10 SoIs, it is 64 bytes per application. The per-server state records the IDs of applications running on a server and the cumulative sensitivity to interference (roughly 64 bytes per server). The per-server state is updated as applications are scheduled and, later on, completed. Overall, state overheads are marginal and scale logarithmically or linearly with the number of applications (*N*) and servers (*M*). In our experiments with thousands of applications and servers, a single server could handle all processing and storage requirements of scheduling, although additional servers can be used for fault tolerance.

Greedy server selection

In examining candidates, the scheduler considers two factors: first, which assignments minimize negative interference between the new application and existing load, and second, which servers have the best SC for this workload.

The scheduler evaluates two metrics, $D_1 = t_{server} - c_{newapp}$ and $D_2 = t_{newapp} - c_{server}$, where t is the sensitivity score for tolerated and c for caused interference for a specific SoI. The cumulative sensitivity of a server to caused interference is the sum of sensitivities of individual applications running on it, whereas the sensitivity to tolerated interference is the minimum of these values. The optimal candidate is a server for which D_1 and D_2 are exactly zero for all SoIs, which implies no negative impact from interference and perfect resource usage. In practice, a good selection is one where D_1 and D_2 are positive and small for all SoIs. Large, positive values for D_1 and D_2 indicate suboptimal resource utilization. Negative values for D_1 or D_2 imply violation of QoS.

We examine candidate servers for an application in the following way. The process is explained for interference tolerated by the server and caused by the new workload (D_1) and is exactly the same for D_2. We start from the resource the new application is most sensitive to. We select the server set for which D_1 is non-negative for this SoI. Next, we examine the second SoI in order of decreasing sensitivity scores, filtering out any servers for which D_1 is negative, until all SoIs have been examined. Then, we take the intersection of server sets for D_1 and D_2 and select the machine with the best SC and with $min(\|D_1 + D_2\|_{L_1})$.

As we filter out servers, at some point the set of candidate servers might become empty. This implies that there is no single server for which D_1 and D_2 are non-negative for some SoI. Although unlikely, we support this event with backtracking and QoS relaxation. Given M servers, the worst-case complexity is $O(M \cdot SoI^2)$, because, theoretically, backtracking might extend all the way to the first SoI. In practice, however, we observe that for a 1,000-server system, 89 percent of applications were scheduled without any backtracking. For 8 percent of the remaining applications, backtracking led to negative D_1 or D_2 for a single SoI (and for 3 percent for multiple SoIs). Additionally, we bound the runtime of the greedy search using a timeout mechanism, after which the best server from the ones already examined is selected.

Our full paper includes a discussion on workload phases and applicability to multitier latency-critical applications.[5]

Evaluation methodology

In the following paragraphs, we describe the server systems, alternative schedulers, applications, and workload scenarios used in our evaluation.

We evaluated Paragon on a 1,000-server cluster on Amazon EC2 with 14 instance types from small to extra large.[12] All instances were exclusive (reserved)—that is, no other users had access to the servers. There were no external scheduling decisions or actions such as auto-scaling or workload migration during the course of the experiments.

We compared Paragon to three schedulers. The first is a baseline scheduler that assigns applications to least-loaded (LL) machines, accounting for their core and memory requirements but ignoring their heterogeneity and interference profiles. The second is a heterogeneity-oblivious (NH) scheme that uses the interference classification in Paragon to assign applications to servers without visibility in their SCs. The third is an interference-oblivious (NI) scheme that uses the heterogeneity classification but has no insight on workload interference.

We used 400 single-threaded (ST), multithreaded (MT), and multiprogrammed (MP) applications from SPEC CPU2006, several multithreaded benchmark suites,[5] and SPECjbb. For multiprogrammed workloads, we created 350 mixes of four SPEC applications. We also used 26 I/O-bound workloads in Hadoop and Matlab running on a single node. Workload durations range from minutes to hours. For workload scenarios with more than 426 applications, we replicated these workloads with equal likelihoods (1/4 ST, 1/4 MT, 1/4 MP, and 1/4 I/O) and randomized their interleaving.

We used the applications listed in this section to examine the following scenarios: a low-load scenario with 2,500 randomly chosen applications submitted with 1-second intervals, a high-load scenario with 5,000 applications submitted with 1-second intervals, and an oversubscribed scenario where 7,500 workloads are submitted with 1-second intervals and an additional 1,000 applications arrive in

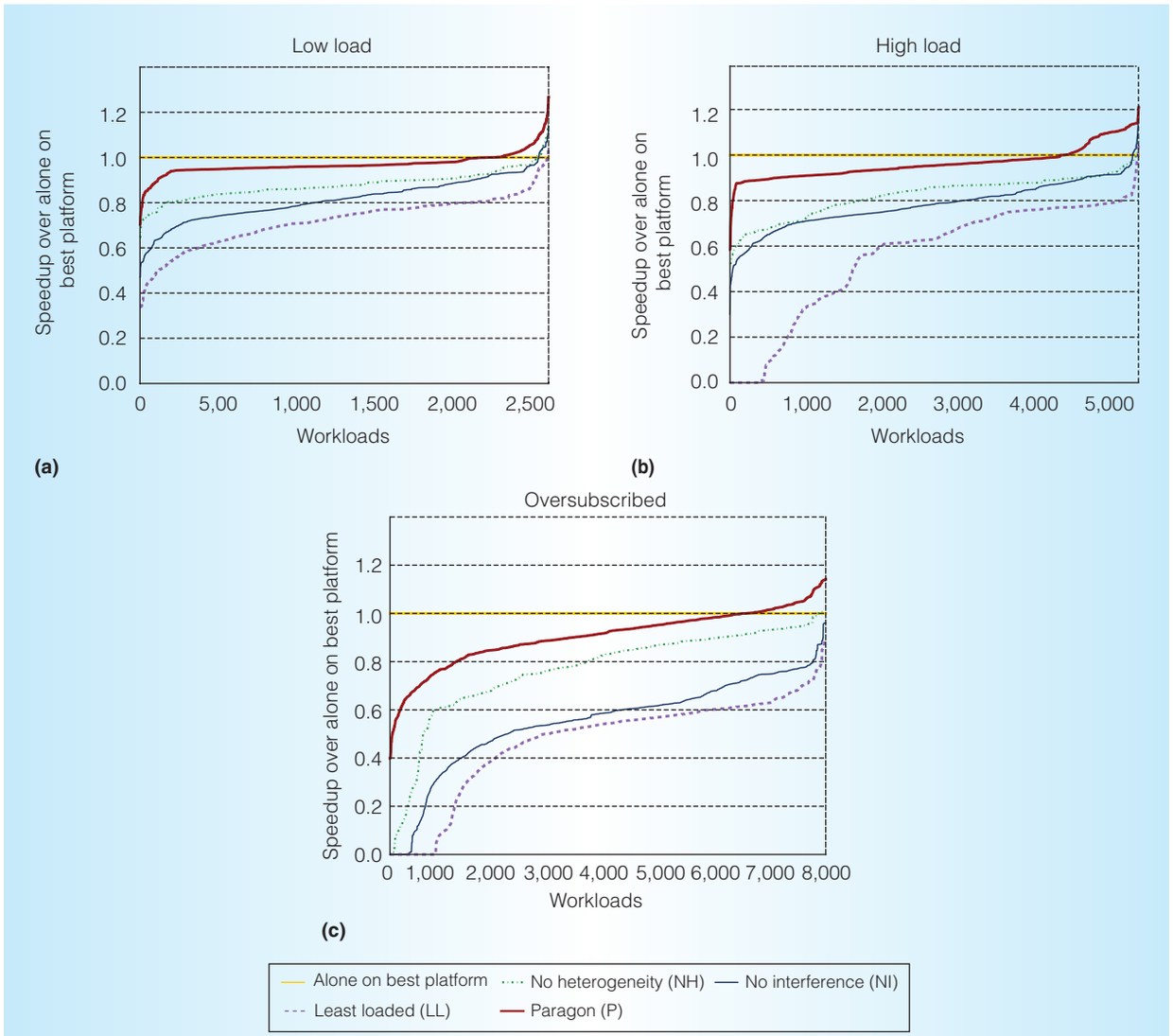

Figure 2. Performance comparison between the four schedulers for three workload scenarios on 1,000 Amazon Elastic Compute Cloud (EC2) servers. Performance is normalized to optimal performance in isolation, and applications are ordered from worst to best performing.

burst (less than 0.1-second intervals) after the first 3,750 workloads.

Evaluation

We evaluated the Paragon scheduler against the LL, NH, and NI schedulers, with respect to performance, decision quality, resource allocation, and cluster utilization.

Performance impact

Figure 2 shows the performance for the three workload scenarios on the 1,000-server EC2 cluster. The low-load scenario, in general, does not create significant performance challenges. Nevertheless, Paragon outperforms the other three schemes; it preserves QoS for 91 percent of workloads and achieves on average 96 percent of the performance of a workload running in isolation in the best SC. When moving to the high-load scenario, the difference between schedulers becomes more obvious. Although the heterogeneity and interference-oblivious schemes degrade performance by an average

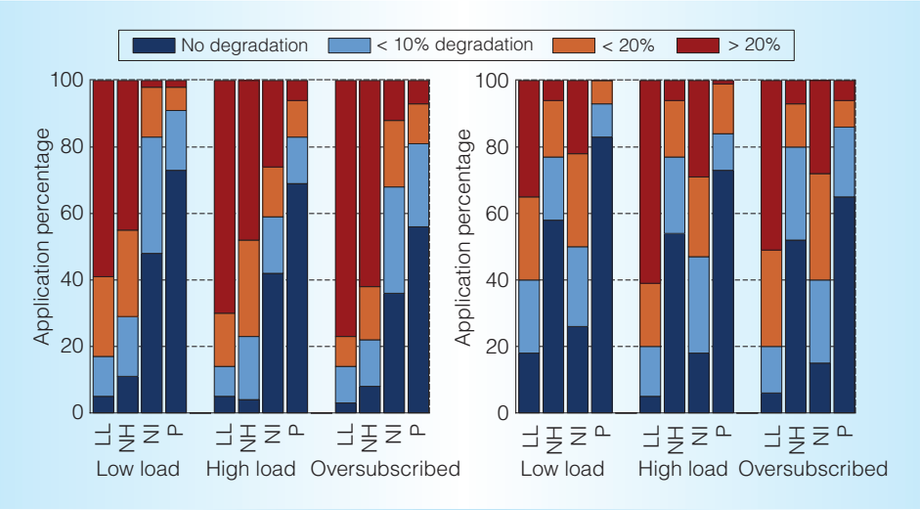

Figure 3. Breakdown of decision quality for the four schedulers across the three EC2 scenarios. Different colors correspond to different impacts in application performance in terms of heterogeneity (left) and interference.

of 22 and 34 percent and violate QoS for 96 and 97 percent of workloads, respectively, Paragon degrades performance by only 4 percent and guarantees QoS for 61 percent of workloads. The least-loaded scheduler degrades performance by 48 percent on average, with some applications not terminating successfully. The differences in performance are larger for workloads submitted when the system is heavily loaded.

Finally, for the oversubscribed case, NH, NI, and LL dramatically degrade performance for most workloads, while the number of applications that do not terminate successfully increases to 10.4 percent for LL. Paragon, on the other hand, preserves QoS guarantees for 52 percent of workloads, while the other schedulers provide similar guarantees only for 5, 1, and 0.09 percent of workloads, respectively. Additionally, it limits degradation to less than 10 percent for an additional 33 percent of applications and maintains moderate performance degradation (no cliffs in performance similar to NH for applications 1 through 1,000).

Decision quality

Figure 3 shows a breakdown of the decision quality of the different schedulers for heterogeneity (left) and interference (right) across the three scenarios. LL induces more than 20 percent performance degradation to most applications, both due to heterogeneity and interference. NH has low decision quality in terms of platform selection, whereas NI causes performance degradation by colocating unsuitable applications. The errors increase as we move to scenarios of higher load. Paragon decides optimally for 65 percent of applications for heterogeneity and 75 percent for interference, on average, significantly higher than the other schedulers. It also constrains decisions that lead to larger than 20 percent degradation to less than 8 percent of workloads.

Resource allocation

Figure 4 shows why this deviation exists. The solid black line in each graph represents the required core count based on the applications running at a snapshot of the system, while the other lines show the allocated cores by each of the schedulers. Because Paragon optimizes for increased utilization within QoS constraints, it follows the application requirements closely. It only deviates when the required core count exceeds the resources available in the system (oversubscribed case). NH has mediocre accuracy, whereas NI and LL either significantly overprovision the number of allocated cores, or oversubscribe certain servers. There are two important points in

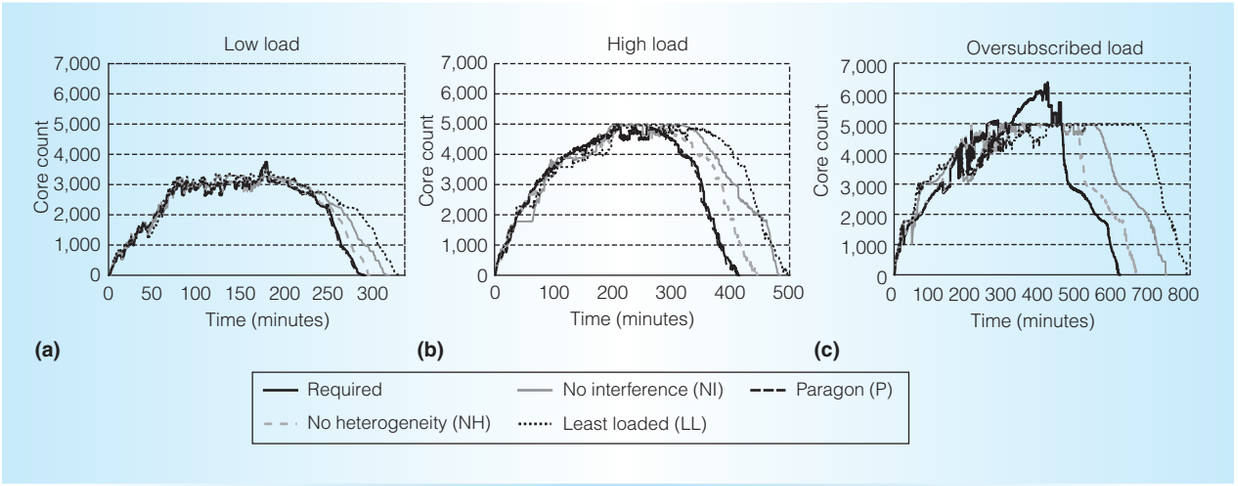

Figure 4. Resource allocation for the three workload scenarios. Each line corresponds to the number of allocated computing cores at each point during the execution of the scenario. Although the heterogeneity-oblivious (NH), interference-oblivious (NI), and least-loaded (LL) schedulers under- or overestimate the required resources, Paragon closely follows the application resource requirements.

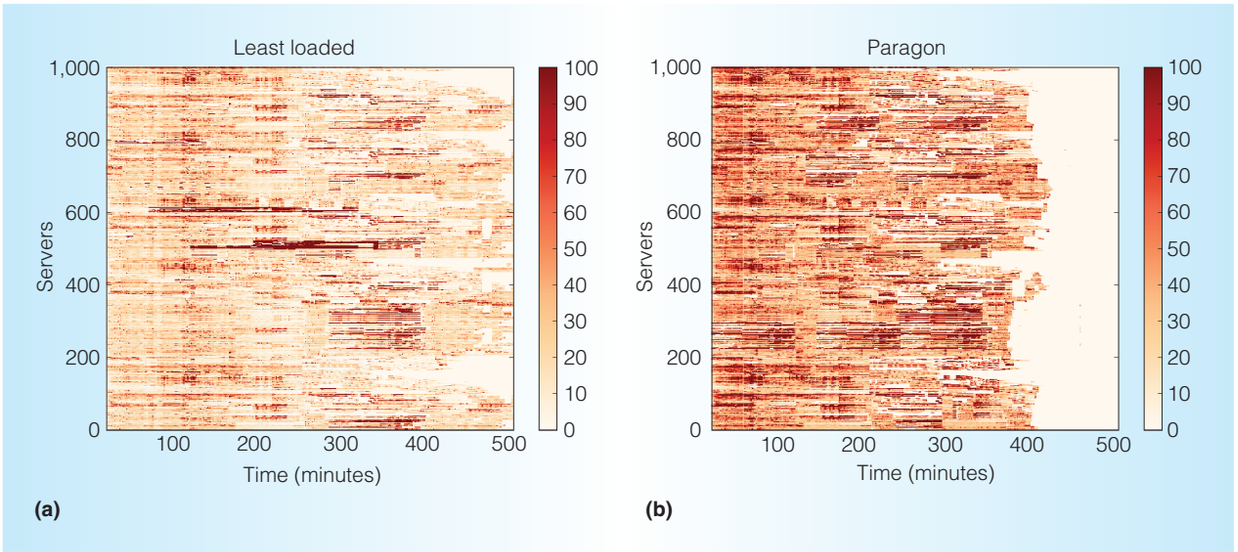

Figure 5. CPU utilization heat maps for the high-load scenario for the least-loaded system and Paragon. Utilization is averaged across the cores of a server and is sampled every 5 seconds. Darker colors correspond to higher CPU utilization in the heatmaps.

these graphs. First, as the load increases, the deviation of execution time from optimal increases for NH, NI, and LL, whereas Paragon approximates it closely. Second, for high loads, the errors in core allocation increase dramatically for the other three schedulers, whereas for Paragon the average deviation remains approximately constant, excluding the part where the system is oversubscribed.

Cluster utilization

Figure 5 shows the cluster utilization in the high-load scenario for LL and Paragon in the form of heat maps. Utilization is shown

for each individual server throughout the duration of the experiment and is averaged across the server's cores every 5 seconds. Whereas with LL utilization does not exceed 20 percent for the majority of time, Paragon achieves an average utilization of 52 percent. Additionally, as workloads run closer to their QoS requirements, the scenario completes in 19 percent less time.

The Paragon scheduler moves away from the traditional empirical design approach in computer architecture and systems and adopts a more data-driven approach. In the past few years, we have entered an era where data has become so vast and rich that it can provide much better (and faster) insight on design decisions than the traditional trial-and-error approach can. Applying such techniques in datacenter scheduling with significant gains is proof of the value of using data to drive system design and management decisions. There are other highly dimensional problems where similar techniques can be proven effective, such as the large space-design explorations for either processors[13] or memory systems or the more general cluster management problem in cloud providers. The latter becomes increasingly challenging because many cloud applications are multitier workloads with complex dependencies and they must satisfy strict tail latency guarantees. Additionally, issues like heterogeneity and interference are not relevant only to datacenters. Systems of all scales, from low-power mobile to traditional CMPs and large-scale cloud computing facilities, face similar challenges, which makes employing techniques that work online, fast and can handle huge spaces a pressing need.

Determining which data can offer valuable insights in system decisions and designing efficient techniques to collect and mine it in a way that leverages their nature and characteristics is a significant challenge moving forward.

MICRO

Acknowledgments

We sincerely thank John Ousterhout, Mendel Rosenblum, Byung-Gon Chun, Daniel Sanchez, Jacob Leverich, David Lo, and the anonymous reviewers for their feedback on earlier versions of this manuscript. This work was partially supported by a Google-directed research grant on energy-proportional computing. Christina Delimitrou was supported by a Stanford Graduate Fellowship.

References

1. L.A. Barroso and U. Holzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, Morgan and Claypool, 2009.
2. J. Rabaey et al., "Beyond the Horizon: The Next 10x Reduction in Power—Challenges and Solutions," *Proc. IEEE Int'l Solid-State Circuits Conf.*, 2011, doi:10.1109/ISSCC.2011.5746206.
3. L. Barroso, "Warehouse-Scale Computing: Entering the Teenage Decade," *Proc. 38th Ann. Int'l Symp. Computer Architecture* (ISCA 11), 2011.
4. D. Meisner et al., "Power Management of Online Data-Intensive Services," *Proc. 38th Ann. Int'l Symp. Computer Architecture* (ISCA 11), 2011, pp. 319-330.
5. C. Delimitrou and C. Kozyrakis, "Paragon: QoS-Aware Scheduling in Heterogeneous Datacenters," *Proc. 18th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 13), 2013, pp. 77-88.
6. R.M. Bell, Y. Koren, and C. Volinsky, *The BellKor 2008 Solution to the Netflix Prize*, tech. report, AT&T Labs, Oct. 2007.
7. J. Mars et al., "Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations," *Proc. 44th Ann. IEEE/ACM Int'l Symp. Microarchitecture*, 2011, pp. 248-259.
8. R. Nathuji, C. Isci, and E. Gorbatov, "Exploiting Platform Heterogeneity for Power Efficient Data Centers," *Proc. 4th Int'l Conf. Autonomic Computing* (ICAC 07), 2007, doi:10.1109/ICAC.2007.16.
9. N. Vasic et al., "DejaVu: Accelerating Resource Allocation in Virtualized Environments," *Proc. 17th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 2012, pp. 423-436.

10. A. Rajaraman and J.D. Ullman, *Mining of Massive Datasets*, Cambridge Univ. Press, 2011.
11. C. Delimitrou and C. Kozyrakis, "iBench: Quantifying Interference for Datacenter Workloads," *Proc. IEEE Int'l Symp. Workload Characterization*, 2013, pp. 23-33.
12. C. Delimitrou and C. Kozyrakis, "QoS-Aware Scheduling in Heterogeneous Datacenters with Paragon," *ACM Trans. Computer Systems*, vol. 31, no. 4, 2013, article no. 12.
13. O. Azizi et al., "Energy Performance Tradeoffs in Processor Architecture and Circuit Design: A Marginal Cost Analysis," *Proc. 37th Ann. Int'l Symp. Computer Architecture* (ISCA 10), 2010, pp. 26-36.

Christina Delimitrou is a PhD student in the Department of Electrical Engineering at Stanford University. Her research focuses on large-scale datacenters, specifically on scheduling and resource allocation techniques with quality-of-service guarantees, practical cluster management systems that improve resource efficiency, and datacenter application analysis and modeling. Delimitrou has an MS in electrical engineering from Stanford University. She is a student member of IEEE and the ACM.

Christos Kozyrakis is an associate professor in the Departments of Electrical Engineering and Computer Science at Stanford University, where he investigates hardware architectures, system software, and programming models for systems ranging from cell phones to warehouse-scale datacenters. His research focuses on resource-efficient cloud computing, energy-efficient multicore systems, and architectural support for security. Kozyrakis has a PhD in computer science from the University of California, Berkeley. He is a senior member of IEEE and the ACM.

Direct questions and comments about this article to Christina Delimitrou, Gates Hall, 353 Serra Mall, Room 316, Stanford, CA 94305; cdel@stanford.edu.