A Periodic Portfolio Scheduler for Scientific Computing in the Data Center

Kefeng Deng[1,2], Ruben Verboon[2], Kaijun Ren[1], and Alexandru Iosup[2]

[1] National University of Defense Technology, Changsha, China
{dengkefeng,renkaijun}@nudt.edu.cn
[2] Delft University of Technology, Delft, the Netherlands
R.S.Verboon@student.tudelft.nl, A.Iosup@tudelft.nl

Abstract. The popularity of data centers in scientific computing has led to new architectures, new workload structures, and growing customer-bases. As a consequence, the selection of efficient scheduling algorithms for the data center is an increasingly costlier and more difficult challenge. To address this challenge, and contrasting previous work on scheduling for scientific workloads, we focus in this work on portfolio scheduling—here, the dynamic selection and use of a scheduling policy, depending on the current system and workload conditions, from a portfolio of multiple policies. We design a periodic portfolio scheduler for the workload of the entire data center, and equip it with a portfolio of resource provisioning and allocation policies. Through simulation based on real and synthetic workload traces, we show evidence that portfolio scheduling can automatically select the scheduling policy to match both user and data center objectives, and that portfolio scheduling can perform well in the data center, relative to its constituent policies.

Keywords: portfolio scheduling; data center; provisioning and allocation; scheduling policies; scientific workloads.

1 Introduction

Cluster-based data centers of all sizes are increasingly popular, a result of both increasing demand for efficient computational resources, and of several decades of technological advancement and education of administrators in distributed systems. Especially when servicing the demanding workloads typical of scientific computing [1, 2], these data centers need efficient algorithms for scheduling their users' workloads on the data center resources. Many existing scheduling algorithms have already addressed specific workload properties [3, 4] and types of applications [5–8], but data centers still rely on (expensive) human system administrators to select a scheduling algorithm and configure it appropriately. Moreover, the selection process is made significantly more difficult by changing workloads due to technology transitions (e.g., the use of virtualization and new networking architectures), and by new customers starting to use data centers as Infrastructure-as-a-Service clouds. In contrast to previous approaches, we

investigate in this work portfolio scheduling [9]—in this context, the dynamic selection and use of a scheduling policy, depending on the current system and workload conditions, from a portfolio of multiple policies—used to efficiently schedule scientific workloads for the entire data center.

Cluster-based data centers have been much employed for scientific computing workloads. For example, small data centers are commonly integrated into multi-cluster grids, such as the World Large Hedron Collider Grid (WLCG), the US Open Science Grid, the French Grid'5000, and the Dutch DAS. However, human administrators have become increasingly rare and more overloaded, as modern data centers rely increasingly on automation and allocate only 5% of the operational budgets for human administration [10]; this situation is anecdotally supported by our experience with the DAS system over the past decade.

A variety of scheduling and administrative techniques have been developed recently for the data centers, but, simultaneously, data center architectures have evolved quickly. For the former, research has focused on both sharing of networking resources [11] and time-cost-energy optimizations. For the latter, recent work has focused on new layouts of networks [12, 13] and new virtualization architectures [14, 15]. Our previous scheduling studies [8, 16, 17], which evaluate a large variety of scheduling policies for various types of scientific computing, indicate that no single policy can accommodate all workload conditions, and all user and system objective functions. Thus, a tension arises in trying to select the appropriate resource scheduling policies for the data center.

Not only the data center architecture, but also the properties of scientific workloads change over time. Long-term arrival patterns can suddenly be interrupted by bursts of arrivals [2]. As systems mature, their users may transition from loosely coupled jobs to more integrated workflows [18] and even tightly coupled parallel jobs. New approaches to computing—MapReduce and its many flavors, the graph-processing model Pregel, etc.—have appeared in the past few years. For months after data centers are launched in production and prior to their decommissioning, reduced yet system-stressful workloads with different operational patterns may appear [19]. Thus, the problem of selecting an appropriate scheduling policy remains open and increasingly in need of a solution.

In this work we investigate if a portfolio scheduler can automatically select the scheduling policy, from the set with which the portfolio is configured, such that the user and the data center's objective functions remain within their target (optimal) range. Among other differences from previous work on portfolio scheduling [9], our context rarely allows an optimal range to be computed; thus, the target range is relative to the performance of individual policies used in the portfolio. A portfolio scheduler should support many types of workload patterns and application types, yet perform similarly to the scheduling policy that has been specifically designed to support the workload pattern and application type. This has an important consequence: portfolio scheduling can then alleviate the need for human expertise in selecting scheduling policies and even configurations,

and thus become an important component in the administration of modern data centers.

A full exploration of the concept portfolio scheduling would greatly exceed the scope of this work. Among the challenging questions we do not explore are: Which policies should be selected in the portfolio? How can a portfolio support a mix of application types? Should the portfolio also configure policies, as part of its operational process? Should the portfolio select the scheduling policy periodically or continuously? In this work, we focus on exploring portfolio scheduling, with a twofold contribution:

1. We adapt the notion of periodic portfolio scheduling in the context of data centers (Section 3). We propose a periodic portfolio scheduler, and create a comprehensive portfolio for provisioning and allocation of resources.
2. We evaluate our portfolio scheduler experimentally, through synthetic and real trace-based simulation (Section 5). We compare the portfolio scheduler against its constituent policies and show evidence that portfolio scheduling can be beneficial in the context of data centers.

2 System Model

In this section we present the system model used throughout this work.

2.1 Workload and Resource Model

The workloads we consider in this work match the cluster-based traces of the Parallel Workloads Archive [20]. We further assume that jobs are CPU-bound and their runtime depends linearly on the speed of the (virtual) processor where they are executed. Because it offers a trade-off between accuracy of simulation and simplicity, this model has been much used by the parallel and grid computing communities for simulation-based work. For example, we have used it in our DGSim simulator [21].

In this work, we consider the functioning of a data center comprised of homogeneous physical resources. This model is common for the multi-cluster grids of the late-1990s up to mid-2000s—many of their clusters and even entire data centers have been initially built with homogeneous resources. This model also matches today's virtualized (homogenized) infrastructure.

Similarly to simulation-based studies in parallel and grid computing, we assume that resources can be benchmarked to quantify their speed for processing typical scientific computing workload, for example with the SPEC CPU2006 benchmark. Although this assumption may fail for other application domains or for scientific applications with irregular operational patterns [22], this approach to benchmarking has been successfully employed in the operation of several multi-cluster grids, such as the Worldwide LHC Computing Grid (WLCG), the French Grid'5000, and the Dutch DAS.

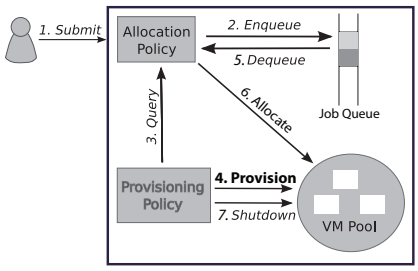

Fig. 1. Operational model of the data-center scheduler.

2.2 Operational Model

In this work, resources are provisioned exclusively from a single data center. We do not consider hybrid computing environments that span multiple data centers, because scientific and especially parallel workloads can rarely withstand co-allocation across geographically-separate data centers without significant loss of performance [7, 23].

In our resource usage model, all resources belong to the data center and are provisioned on-demand for incoming workload as virtual machines (VMs). Besides on-demand resource provisioning, we do not consider in this work advance reservation, consolidation of multiple jobs on the same resources, and other usage models [24]. The use of virtualization allows the data center to service a diverse set of scientific computing users on the same set of physical resources; the performance impact of virtualization for scientific computing has been well studied in the past and shown [25–28] to be small, for non-I/O-bound and small-scale scientific applications.

The data-center scheduling model investigated in this work is adapted from our previous work [16]. As depicted in Figure 1, users send their workloads to a system-wide scheduler, which uses an *allocation policy* to either allocate these jobs to the VMs already provisioned for the submitting user (the *VM pool*), or to enqueue the jobs in the system-wide *job queue*. VMs are provisioned, that is, leased and released, on behalf of the user by the system-wide scheduler via a *provisioning policy*. To inform proactive provisioning decisions, the provisioning policy can query the state of the allocation. From the perspective of the data center operator, the provisioning policy is in general responsible for the efficient allocation of resources to users.

Inspired by the use of data centers as IaaS cloud infrastructure, we use the billing model of Amazon EC2: VM use is charged in hourly increments.

3 A Periodic Portfolio Scheduler

In this section we adapt traditional portfolio scheduling [9] for use in the data center. We describe, in turn, our periodic portfolio scheduler, an overview of the system including the portfolio scheduler, and the set of policies used by our portfolio scheduler (later used in experimental work, in Section 5).

3.1 The Portfolio Scheduler

Portfolio schedulers follow a traditional process with four steps, creation, selection, application, and reflection. We adapt this process to data centers and design a periodic portfolio scheduler, as follows.

In the *creation* step, a set of policies is created for the portfolio scheduler, prior to the actual use of policies. The main trade-off in the creation of this set is between *capability* to schedule different workload patterns and application types, and *time* required to explore the set during the selection and application phases. The selection of policies is usually done by an expert, as we do in Section 3.3, but can also be done automatically, for example as the result of an automated comparative study of policies specific to one domain [8, 16, 17].

During *selection*, the portfolio scheduler has to select one of the scheduling policies, to be used, in the case of *continuous portfolio scheduling*, for the next scheduling decision or, for *periodic portfolio scheduling*, during the next period of taking scheduling decisions. As for the creation step, the selection step can be guided by an expert or be automated, and needs to address a trade-off between time spent in selection and *quality of selection*, which is typically a single-user utility function or a system-wide performance metric. The portfolio scheduler we propose in this work is periodic and automated; we explore various metrics for the quality of selection in Section 5. We detail in Section 3.2 a practical selection process that can be used in data center scheduling.

In the *application* step, the policy selected in the previous step takes scheduling decisions. Additionally, the portfolio scheduler collects information about the application of scheduling decisions, and may use the collected information to evaluate how the non-selected policies in its portfolio would have performed if selected. Although this step also appears in non-portfolio scheduling, for portfolio scheduling this step can be more complex. If the selected policy is complex, its application may raise non-trivial system stability issues and lead to system inefficiency. For example, the newly selected policy may be undoing some of the advanced reservations or other long-term planning decisions of previously selected policies. We see the exploration of the non-trivial interplay between the selection and application steps, including stabilization of a multi-policy system, as fertile ground for future research.

The *reflection* step analyzes the operation of the last selection and application steps, and may take the decision to change the portfolio or tune the other steps. Changing the portfolio is similar to the creation step. Tuning the other steps may, for example, lead to switching the selection step from a periodic to a continuous process, adapting the selection criterion, and setting different thresholds regarding the overturning of previous scheduling decisions when applying the newly selected policy. We leave the exploration of this step for future work.

Ideally, the portfolio scheduler has the ability to always select, for an arbitrary workload *mode* (i.e., workload pattern or application type), the best scheduling policy in the portfolio. Thus, a portfolio scheduler cannot outperform its constituent policies when confronted with a mono-modal workload. Instead,

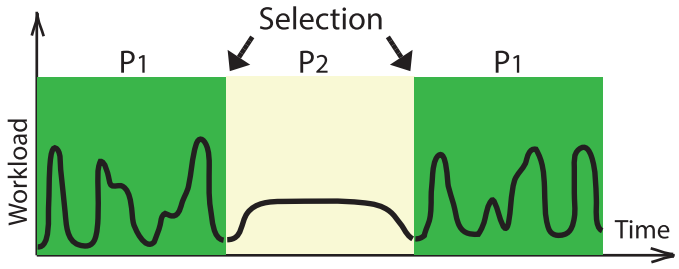

Fig. 2. Selected policy over the lifetime of a system with changing workload modes.

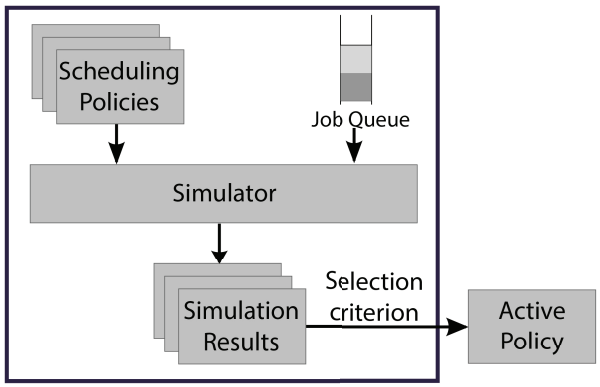

Fig. 3. The policy selection process of our periodic portfolio scheduler.

a portfolio scheduler should become useful when the workload changes modes in quick succession. For example, Figure 2 depicts a synthetic workload that alternates two arrival patterns, to be scheduled by a portfolio scheduler comprised of two policies, each adapted to one of the arrival patterns. In this constructed example, the policy is changed automatically after each selection step, in response to changes in the workload. The result in an alternation between policies P_1 and P_2. Moreover, when the succession is aperiodic or has a long period, the portfolio scheduler should become increasingly more difficult to replace with human decision-making.

3.2 System Using Portfolio Scheduling

We now extend the operational model introduced in Section 2.2 to accommodate a periodic portfolio scheduler. The main elements of this model, the system-wide scheduler, the order of operations involving the selected policies, etc., remain unchanged. Because the application of the (selected) scheduling policy remains unchanged from the initial operational model, we focus in this section on the portfolio creation and selection steps of the process introduced in Section 3.1.

For our portfolio scheduler, the creation step is executed *once* and the selection step is executed *periodically*. The selection automatically evaluates the set of policies in the scheduler's portfolio, and selects from it the policy to be

applied for the entire next period. The main choices in the design of our periodic portfolio scheduler are:

(Creation step) Which policies? The data center scheduler includes various policies, each of which can be selected through portfolio scheduling. Moreover, the combination of policies can also be selected through portfolio scheduling. Assuming that most policies have a similarity in computational demand, which matches well the simple heuristics commonly employed in data center scheduling, the time required for single- and multi-policy selection increases linearly and exponentially, respectively. We design our portfolio scheduler to cover the combination of provisioning and allocation policies, that is, the portfolio is comprised of pairs of provisioning and allocation policies. We detail the specific policies used in this work in Section 3.3.

(Selection step) How to evaluate? We design the evaluation of policies to use a *simulation*-based approach; alternatives include running selected workload parts in a reserved system partition and extrapolating results, using historical performance information and periodically risking on previously untried policies, etc. In the simulation approach (see Figure 3), each scheduling policy in the portfolio is evaluated against a simulated environment that matches the data center, subject to the currently running and queued jobs in the data center. After all the simulations are complete, a *selection criterion* is used to select the next active policy.

The **selection step** also involves several important configuration parameters:

The simulator The choice of a simulator is non-trivial, with the main trade-off in the accuracy of results (the ability to match the real environment) and execution time. Fast and accurate simulators already exist for various data center architectures [29, 21]. We select from these simulators and use in our portfolio scheduler DGSim [21], which has been used previously for data center architectures such as independent and multi-cluster system.

The interval between selections (the period of the selection step), τ, is set by the system administrator (e.g., to $\tau = 20s$). If τ is small, the selection may overload the system scheduler and may occur too frequently to allow for meaningful scheduling. It τ is large, delays are unnecessarily incurred on the execution of the workload. We leave for future work the automatic setting and tuning of this system parameter.

The maximal simulation time , T, defined as the maximal duration for each independent simulation. We have selected this single parameter from the broader trade-off between the number of parameters in the system and the ability to configure the maximal runtime for several (classes of) policies.

The selection criterion (or the utility function), U, which is used to select the next active policy after all the simulations are complete. In this work we use a selection criterion that balances the job slowdown as a proxy for user experience, and utilization of the provisioned resources as a proxy for system efficiency and cost; the metric will be detailed in Section 4.3.

3.3 Portfolio Policies

In this section, we describe the policies that we select for our portfolio scheduler. We present, in turn, the selected provisioning and allocation policies. We use six provisioning policies from our recent study of IaaS clouds [16] and two allocation policies commonly used in data centers. Our choice of provisioning policies matches the system model requirement of hourly charging per VM (see Section 2.2). Among the six provisioning policies, we use the last five policies in our portfolio:

1. (The **baseline provisioning policy**) *StartUp (STU):* This policy leases a new instance whenever there is no idle VM for the current job unless the number of rented VMs reach its maximum. Moreover, the rented VMs will not be released until the end of the workload. The advantage of STU is that it can provide user with good experience by over-provisioning VM instances. However, it cannot deal well with changing workloads such as bursty workload, since it is static and keeps VM instances alive after the flash-crowd even when there are no jobs.
2. *On-Demand Single VM (ODS):* This is a simple dynamic provisioning policy. It leases a new VM instance for each job that is waiting in the queue, whenever available instances can be provisioned in the data center (whenever there are free resources). Since instances are charged hourly, they are released when there is no job for them to run and their run time is reaching integral hours. This policy is naive: although it may lead to good user experience, it also incurs unnecessarily high cost—resources charged for an entire hour may be released after just a few minutes.
3. *On-Demand Geometric (ODG):* Because scientific workloads may include many short jobs that finish before the hourly charging of resources, it is not necessary to rent a new instance for every job. Therefore, The ODG policy is used to rent VMs gradually. ODG leases and releases VM instances in a similar way to TCP's exponential back-off mechanism [30]. A parameter $\alpha \geq 1$ is used to control the growth (shrink) of the number of VMs to be leased (released) at each provisioning step, i.e., this policy leases $\alpha^0, \alpha^1, \alpha 2, \ldots, \alpha^n$ instances, successively. We have shown in our previous work [16] evidence that this policy is helpful for bursty workloads.
4. *On-Demand ExecTime (ODE):* Job information may be helpful for taking better scheduling decisions. The ODE policy takes the execution time of the jobs into consideration for leasing VM instances. First, it estimates the run time of the queued jobs as the (historically recorded) average run time of similar jobs, for example jobs submitted by the same user. Then, it computes the number of VMs to be rented by rounding up the total execution time to hours. This policy also uses the VM release strategy of the ODS policy.
5. *On-Demand WaitTime (ODW):* Similarly to the ODE policy, and taken from previous work [16], the ODW policy uses the job wait time to decide how many instances to be rented. First, a threshold is empirically set for the maximal job wait time, to the next 5-minute increment that exceeds by 5 times the latency to acquire and boot a VM instance; we set it in this work to

20 minutes. At every provisioning point, ODW checks the wait time of each job, then leases VMs for each job having waited longer than the threshold. This policy also releases VMs near integral hours of run time.

6. *On-Demand XFactor (ODX):* This policy tries to give an upper bound for job slowdown. To this end, ODX uses both the (observed) wait time and the (estimated) run time to rent instances. ODX uses the same method as ODE to estimate the job run time. Idle VM instances are reused or, if none exists, leased whenever a job has been delayed longer than its run time (a slowdown of 2). This policy also uses the VM release strategy of the ODS policy.

We also consider two allocation policies for our portfolio, First-Come-First-Served (FCFS) and the second one is Shortest-Job-First (SJF). FCFS is the traditional allocation policy used in many data centers; it is fair but may cause fragmentations in the system. SJF is an aggressive policy: it can reach lower average job slowdown or wait time, but may also cause starvation for long jobs.

4 Experimental Goals and Setup

We evaluate in this section portfolio scheduling for scientific workload executed in the data center using an experimental approach. We compare the performance of the portfolio scheduler and of its constituent policies, when used independently. We conduct this evaluation using simulation (Section 4.1). As input to the simulator, we use synthetic and real-world traces corresponding to scientific workloads (Section 4.2). Last, as user- and data-center-oriented objective functions we use several metrics (Section 4.3).

When presenting results in this section, we use predominantly two-letter terms to denote the policy combinations in our experiments. For the six provisioning policies described in Section 3.3 we use the letters **U**, **S**, **G**, **E**, **W**, and **X**, respectively. For the FCFS and SJF allocation policies described in Section 3.3 we use the letters **F** and **S**, respectively. Thus, the combination between the provisioning policy OD**X** and the allocation policy **F**CFS is depicted as **XF**. Our portfolio scheduler is indicated through the acronym **PO**.

4.1 Simulator

In this paper, we use simulation[3] to evaluate the effectiveness of our portfolio scheduler, and to compare it with the individual pairs of provisioning and allocation policies that can be formed with the policies introduced in Section 3.3.

To this end, we extend our discrete event simulator DGSim [21] with entities such as a cloud-like resource manager and VM instances. The cloud-like resource manager implements Amazon EC2-like APIs for leasing and releasing VM instances, and implements the cost model of on-demand instances leased by

[3] The simulator used in this section should not be confused with the simulator running as part of the portfolio scheduler. Replacing the simulator used in this section, we have begun experimenting with a real-world prototype of our portfolio scheduler.

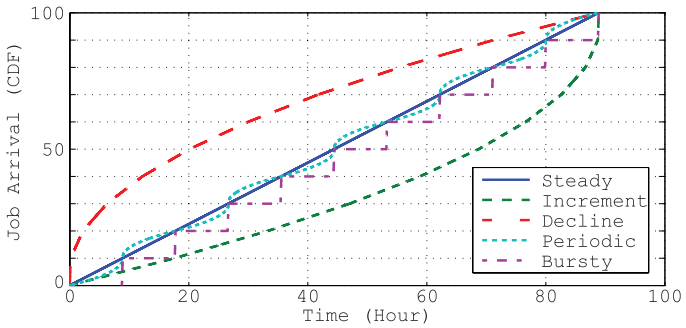

Fig. 4. The arrival of jobs for the five synthetic workloads.

Amazon EC2. To simulate a virtualized environment, we set realistically a delay for instance acquisition and booting, which is 4 minutes based on our previous research [31, 16]. To enable future comparative experiments between the environment simulated in this work and our real-world system DAS-4, which provides OpenNebula-based and Eucalyptus-based cloud interfaces, the maximum number of VMs that can be rented is set to 64.

Our simulator implements the system model introduced in Section 2. We further assume in our simulation that jobs run exclusively on their VMs and cannot be preempted or migrated. Although these assumptions are both common and do not affect the simple allocation policies investigated in this work, we intend to work on relaxing these assumptions, in future work.

4.2 Workloads

We use both synthetic workloads and real workload traces for evaluation. The synthetic workloads are short-term but with significantly different job arrival patterns, allowing us to better characterize the impact of the arrival process on portfolio scheduling. The real workload is a whole trace from the Parallel Workloads Archive [20] and allows us to gain valuable insight into the operation of our portfolio scheduler in realistic conditions.

Synthetic Workloads: In this paper, we generate five types of workloads that have different user behaviors but the same (real) job run times. We take the jobs run times from the first 1000 jobs of the ANL Intrepid 2009 workload from the Parallel Workload Archive [20]. Then, we generate the arrival time of each job such that each synthetic workload exhibits a different arrival patterns. The five arrival patterns, for which the generated workloads are compared in Figure 4, are:

1. *Steady:* the interval between two consecutive jobs is statically set to 5 minutes.
2. *Increment:* The initial interval in this workload is set to 10 minutes. After every 100 job arrivals, the interval is decreased by 70 seconds.
3. *Decline:* In contrast to Increment, Decline sets the initial interval to 5 seconds, then increased by 70 seconds for every 100 job arrivals.

4. *Periodic:* This workload exhibits a periodic pattern from an increasing arrival rate to an decreasing one. Each increasing and decreasing trend continues for 100 job arrivals. The inter-arrival times range from 10 seconds to 10 minutes.
5. *Bursty:* Real workloads often include short periods of bursty behavior. For our bursty arrivals, the submit interval during a bursty period is set to 5 seconds, and bursts include 100 jobs; there are 10 bursts in the workload.

The Real Workload Trace: To evaluate the performance of portfolio scheduling for scientific computing in realistic conditions, we use the entire ANL Intrepid 2009 workload [20] for our real trace-based experiments. The ANL Intrepid 2009 trace has a makespan of 8 months and contains a total number of 68,936 jobs.

4.3 Performance Metrics

We consider in this work various user and data-center objective functions, expressed as traditional and compound metrics. Job slowdown (S) and job wait time (W) are used as common proxies [3] for user objectives. We also measure the total run time of all the jobs (R_J) and the total run time of all rented VM instances (R_V). Because VMs are charged by the hour, the run times are rounded up to the next hour if they are not integer hours; thus, R_V also denotes the *charged cost*. The utilization of the scheduler is defined as the ratio between R_J and R_V, and indicates the efficiency of the policies. Resource utilization is an important metric for both data center administrators and users. For users, it means cost efficiency when using the virtual resources; for system operators, more efficient policies and thus higher market competitiveness.

Although a lower slowdown is to be desired, it may be the result of (much) higher cost, for example when the provisioning policy is STU (StartUp in Section 3.3). To balance these considerations, we use an extension of an utility function, which is defined elsewhere [16, 17]:

$$U = \kappa \cdot \left(\frac{R_J}{R_V}\right)^{\alpha} \cdot \left(\frac{1}{S}\right)^{\beta}$$

For this metric, κ is a scaling factor for the total score, which we set to 100 in our experiments. The metric parameters α and β are used to express different utility functions: α is used to emphasizes the efficiency of resource usage and β is used to stress the urgency of the jobs. For example, to finish jobs as soon as possible, the utility function is set such that $\beta \gg \alpha$. In this paper, similarly to previous work we set $\alpha = \beta = 1$ to balance system efficiency and user experience.

5 Experimental Results

In this section, we report our experimental results. First, we show the results of using portfolio scheduling for synthetic workloads (Section 5.1). Then, we

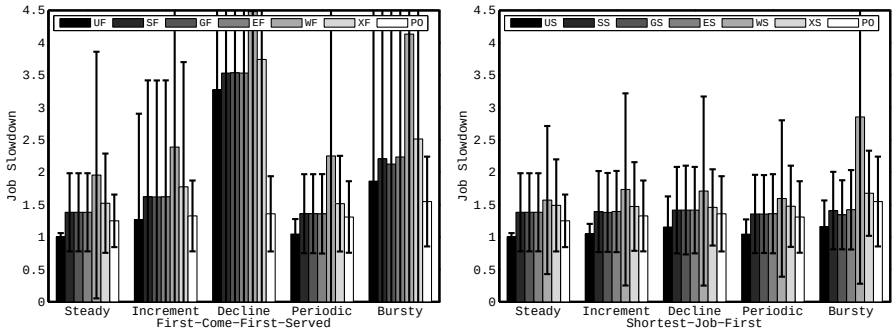

Fig. 5. Job slowdown for different synthetic workloads.

show the results for a real workload trace (Section 5.2). Finally, we analyze the operation of our portfolio scheduler during the experiments (Section 5.3). Overall, we find that portfolio scheduling is useful for data centers.

5.1 Results of Synthetic Workloads

We perform this set of experiments to evaluate our portfolio schedule for the five workload arrival patterns described in Section 4.2.

We first find that the combined policy US (STU+SJF) delivers the lowest average slowdown, but also that PORTFOLIO delivers consistently better results than the other policies. Supporting this finding, Figure 5 depicts the average job slowdown for all the policy combinations, for the five synthetic workloads. In general, and consistently with previous studies of slowdown and also job wait time [32], provisioning policies have relatively lower slowdown when combined with the SJF allocation policy,rather than with FCFS. The performance under Steady, Increment, and Periodic workloads is consistent for all the policy combinations. More pronounced variation appears for Decline and, especially, Bursty workloads. Our portfolio scheduler (PO in Figure 5) performs consistently well in all the cases—PORTFOLIO is the second-best in the first four workloads and very close to the second best in the Bursty workload.

We find that the results for job wait time are much more varied than for job slowdown; as depicted by Figure 6, PORTFOLIO behaves relatively slightly worse in the ranking of policies than for the job slowdown. For many combined policies, the wait time for Decline and Bursty workloads is larger by a factor of about two than for the other workloads. Bursty workloads introduce very challenging scheduling conditions, in which too many jobs overload the system and wait time accumulates. Decline workloads have a quick arrival of jobs in the beginning, similarly to a Bursty workload. During the rapidly varying conditions of Bursty and Periodic workloads, PORTFOLIO is relatively weaker than several other policies, but still delivers relatively good job wait time.

We now investigate the resource utilization, and depict the results in Figure 7; we also depict the charged cost in Figure 8. From these figures, we find that

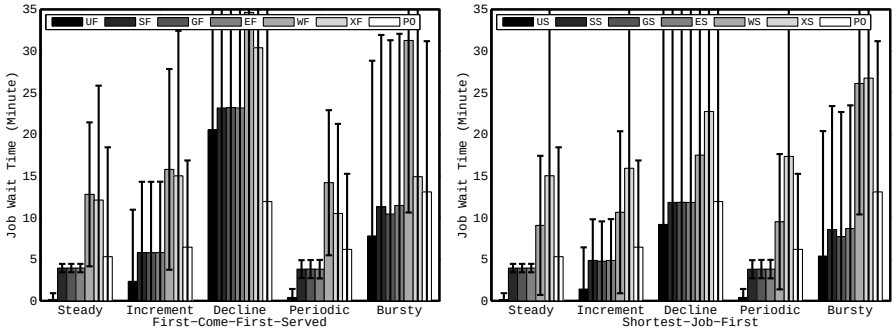

Fig. 6. Job wait time for different synthetic workloads.

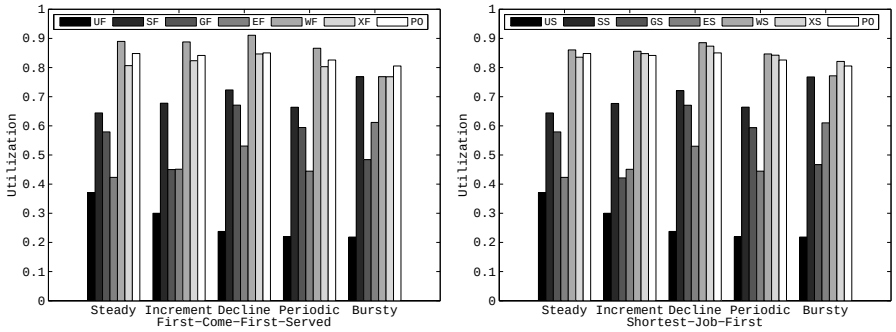

Fig. 7. Utilization for different synthetic workloads.

ODW, ODX, and our PORTFOLIO achieve the highest utilization and the lowest charged cost. As observed in various previous studies [16], the StartUp policy has the lowest utilizations, from 20% to 30%–in line to traditional provisioning policies that only look at peak workloads. As in previous studies of utilization [33], ODS, the commonly used policy in current data centers, achieves only a moderate utilization of 65% to 80%.

Our portfolio scheduler combines consistently low job slowdown and wait time, with low cost (through high utilization). Thus, our portfolio scheduler yields a gain in utility, as depicted in Figure 9. PORTFOLIO is better than its alternatives for all but the Bursty workload, where the SF (ODS+SJF) policy performs better. For the Bursty workload, jobs are submitted every 5 seconds, quickly saturating the system. Thus, it is better to provision resources as soon as possible to avoid unnecessary waiting. As our portfolio scheduler does not predict the future workload, it cannot adapt as quickly to sudden workload changes as the SF policy. However, our portfolio scheduler indeed selects the SF policy most of the time during Bursty workloads, as shown in Section 5.3.

To conclude the experiments using synthetic workloads, we have shown in this section evidence that, for a variety of workload patterns, our portfolio scheduler *can* automatically select the scheduling policy

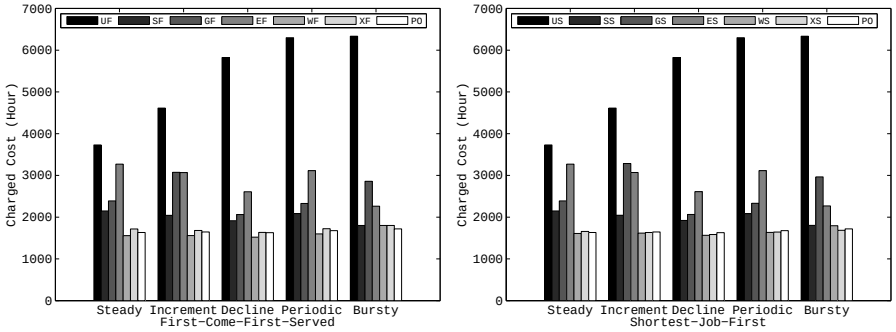

Fig. 8. Charged cost for different synthetic workloads.

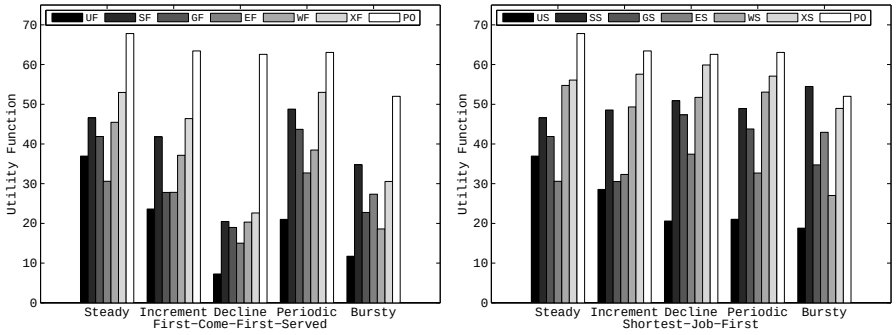

Fig. 9. Value of the utility function for different synthetic workloads.

such that it meets the user and the data center's objective functions at least similarly to, but sometimes even better than, the other scheduling policies investigated here.

5.2 Results of Real Workload Traces

We now turn our attention to the real workload trace collected from ANL Intrepid. We first study the job slowdown and wait time for the real workload trace, and depict the results in Figure 10. StartUp is the best policy and has a slowdown nearly 1–the jobs do not have to wait for execution. PORTFOLIO is among a group of second-best policies, with a slowdown of around 1.5, but has the lowest standard deviation in the group. This favorable behavior of PORTFOLIO is not repeated for the job wait time metric. We attribute this to the selection criterion used in this work, which is based on slowdown.

We further study the charged cost, utilization, and achieved utility for the various policies, when running the ANL Intrepid trace; the results are depicted in Figure 11. The charged cost of StartUp (the earlier best-performer) is about 3 times higher than the competitive policies such as ODW, ODX, and PORTFOLIO; we attribute this to the workload bursts that ANL and many

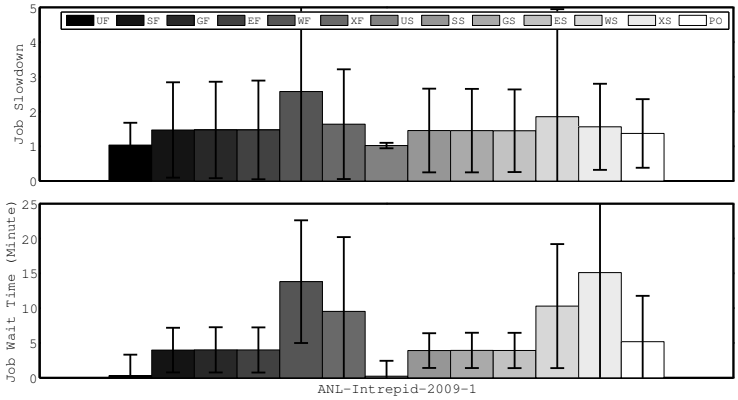

Fig. 10. Job slowdown and wait time for the ANL Intrepid Trace.

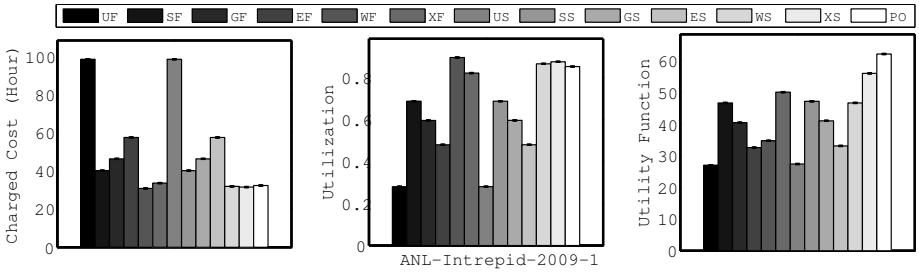

Fig. 11. Charged cost, utilization, and utility for the ANL Intrepid Trace.

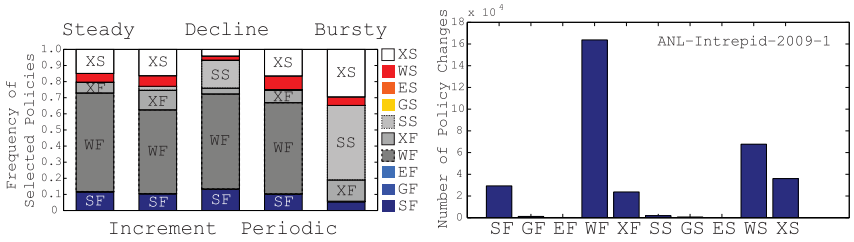

Fig. 12. Ratio and number of policy changes made by the portfolio scheduler.

other production systems exhibit [2]. PORTFOLIO achieves a good combination of utilization and slowdown, leading to overall-best achieved utility.

To conclude the experiments using real workload traces, we have shown in this section evidence that, for the compound metric that characterizes both user and data center objectives, our portfolio scheduler *can* automatically select the scheduling policy achieving better performance than its constituent scheduling policies.

5.3 Analysis of Portfolio Scheduler Operation

To explain the performance obtained in the previous experiments, we analyze the policy selection behavior of our portfolio scheduler. Figure 12 breaks-down

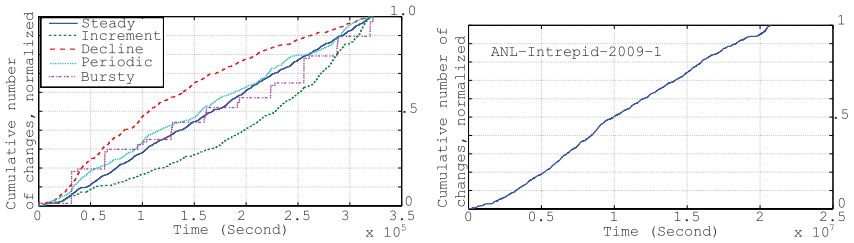

Fig. 13. Cumulative policy changes by the portfolio scheduler, normalized, over time.

the presence of selected policies over entire experiments as relative size (left side) and as absolute counts (right side). Two main conclusions can be made from the figure: (1) although the portfolio scheduler does choose one policy often, several other policies account for a significant fraction of the selections; (2) no single policy is dominant for all the workloads.

We also observe the cumulative number of policy changes over time, and depict this in normalized form in Figure 13. The policy change patterns match very well with the job arrival patterns, indicating that our portfolio scheduler is adaptive and explaining its good performance.

6 Related Work

In this section we survey a large body of related work, related to the concept of computational portfolio design [9], to the modern portfolio theory in finance [34], and to general scheduling in data centers and IaaS clouds. In contrast to these related studies, ours is the first to apply portfolio scheduling to data centers and scientific workloads. Our adaptations of the seminal idea of Huberman [9] and Markowitz [34] to data centers are non-trivial: designing a portfolio around scheduling policies typical to the data center, selecting of utility functions related to both users and data center operators, and designing an operational process that includes simulation-based scheduling.

Closest to our work, Huberman [9] designs a portfolio of search instruments for hard computational problems. This seminal work has led to the creation of a broad field in satisfiability and algorithm portfolio design [35]. Since then, extensive work has focused on improving the selection by and use of heuristics in the portfolio. Streeter et al. [36] consider the duration of heuristics when selected one of them, in the context of dynamic allocation of CPU time. Bougeret [37] and Goldman et al. [38] study the concurrent execution of different heuristics on parallel resources. Besides scheduling of constituent heuristics, Streeter et al. [39] simultaneously address predicting the runtime of heuristics. Gagliolo et al. [40, 41] study the allocation of CPU time based on performance models of constituent algorithms, in the broader context of bandit problems. Our work differs significantly from this body of previous work: previous use of portfolio scheduling tries to find the fastest heuristic for a given set of problem instances, whereas we seek through our scheduler to find the policy that maximizes the

performance objectives given by users and system administrators; the heuristics in previous portfolios generate the same result, whereas those in our portfolio have different solution properties; and the aforementioned work solves a given set of problem instances, whereas our work addresses scheduling of many kinds of unknown and continuous workload patterns.

The portfolio creation and reflection steps, as defined in our work, are important mechanisms in finance. Markowitz [34] introduced a seminal algorithm and set of assumptions for the creation of a portfolio, later refined by Merton [42]. We share with financial portfolios the costs for adding new policies, the risk that the added policy would not perform, the transition costs [43] in changing the portfolio to adapt to expected future conditions in the market or scheduling problem, and the reflection step which is typical in hedging derivatives [44]. Important differences between our work and financial portfolios are that the policies can be infinitely and freely shared among data centers, whereas financial portfolio elements are owned by a single entity at any given time; and that the return of our portfolio is the result of a single, selected policy, whereas in financial portfolios it is the combined return of all the individual assets in the portfolio.

The study of policies for data centers and IaaS clouds has already resulted in a large body of related work. Closest to our work, our own [8, 16, 17] and related [45–47, 5] studies of multiple scheduling policies have emphasized the inability of any single policy to perform well under a wide yet realistic variety of scientific workloads.

The concept of portfolio scheduling may also follow from historical simulation of policies. Historical simulation to adopt the scheduling policy has been done via genetic algorithms in cloud [48] and grid [49] environments. Workloads that have changing properties over time perform better with an adaptive provisioning policy [50], especially one which predicts well the future [51].

7 Conclusion and Future Work

Because data centers around face a growing user base, and an increasingly set of user and data center objectives, the selection of efficient scheduling algorithms is ever costlier and more difficult. Addressing this challenge, we have focused in this work on portfolio scheduling, that is, the dynamic selection and use of a scheduling policy, depending on the current system and workload conditions, from a portfolio of multiple policies.

We have designed in this work a periodic portfolio scheduler for the entire data center. Our portfolio scheduler combines provisioning and allocation policies, and periodically selects from them a pair that optimizes a user-defined or data center-wide utility function. The selection process is simulation-based, that is, our portfolio scheduler simulates at each decision point each of the policies included in its portfolio. Our approach contrasts with previous work on scheduling for scientific workloads, where individual scheduling policies are designed for specific workload patterns and application types but may perform poorly for the dynamic workloads typical of scientific computing. Intuitively,

our portfolio scheduling approach holds the promise of exploiting the collective strengths of its constituent policies, and thus alleviate any of their individual weaknesses.

We have evaluated the behavior of our portfolio scheduler through simulations, based on real and synthetic workload traces. By comparing the statistically meaningful results obtained for our scheduler and for each of its individual policies, independently, we have shown evidence that our portfolio scheduler can perform well in the data center, and better than the alternatives we have considered. We have also shown evidence that our portfolio scheduler can automatically select the scheduling policy to match various user and data center objectives that are common in scientific computing, such as low job slowdown, high resource utilization, and a runtime-efficiency-based utility function. Thus, portfolio scheduling can alleviate the need for human expertise in selecting scheduling policies, and become an important component in the administration of modern data centers.

Extending this work, we are currently conducting a comprehensive sensitivity analysis that covers all the configuration parameters of our portfolio scheduler, such as the interval between selections and the maximal simulation time. Reducing the maximal simulation time may require prioritizing policies for evaluation, to use the spare time remaining from already evaluated policies.

Acknowledgments

Supported by the STW/NWO Veni grant 11881, the National Natural Science Foundation of China (Grant No. 60903042 and 61272483), and the R&D Special Fund for Public Welfare Industry (Meteorology) GYHY201306003.

References

1. Lublin, U., Feitelson, D.G.: The workload on parallel supercomputers: modeling the characteristics of rigid jobs. J. Parallel Distrib. Comput. **63**(11) (2003)
2. Iosup, A., Dumitrescu, C., Epema, D.H.J., Li, H., Wolters, L.: How are real grids used? the analysis of four grid traces and its implications. In: GRID. (2006)
3. Feitelson, D.G., Rudolph, L., Schwiegelshohn, U.: Parallel job scheduling - a status report. In: JSSPP. (2004) 1–16
4. Klusácek, D., Rudová, H.: Performance and fairness for users in parallel job scheduling. In: JSSPP. (2012) 235–252
5. Zhao, H., Sakellariou, R.: Advance reservation policies for workflows. In: JSSPP. (2006) 47–67
6. Sabin, G., Lang, M., Sadayappan, P.: Moldable parallel job scheduling using job efficiency: An iterative approach. In: JSSPP. (2006) 94–114
7. Bucur, A.I.D., Epema, D.H.J.: Scheduling policies for processor coallocation in multicluster systems. IEEE Trans. Parallel Distrib. Syst. **18**(7) (2007) 958–972
8. Iosup, A., Sonmez, O.O., Anoep, S., Epema, D.H.J.: The performance of bags-of-tasks in large-scale distributed systems. In: HPDC. (2008) 97–108
9. Huberman, B.A., Lukose, R.M., Hogg, T.: An economics approach to hard computational problems. Science **27**(5296) (1997) 51–53

10. Greenberg, A.G., Hamilton, J.R., Maltz, D.A., Patel, P.: The cost of a cloud: research problems in data center networks. Comp. Comm. Rev. **39**(1) (2009)
11. Popa, L., Kumar, G., Chowdhury, M., Krishnamurthy, A., Ratnasamy, S., Stoica, I.: Faircloud: sharing the network in cloud computing. In: SIGCOMM. (2012)
12. Greenberg, A.G., Hamilton, J.R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D.A., Patel, P., Sengupta, S.: Vl2: a scalable and flexible data center network. Commun. ACM **54**(3) (2011) 95–104
13. Farrington, N., Porter, G., Sun, P.C., Forencich, A., Ford, J., Fainman, Y., Papen, G., Vahdat, A.: A demonstration of ultra-low-latency data center optical circuit switching. In: SIGCOMM. (2012) 95–96
14. Gordon, A., Amit, N., Har'El, N., Ben-Yehuda, M., Landau, A., Schuster, A., Tsafrir, D.: ELI: bare-metal performance for I/O virtualization. In: ASPLOS. (2012)
15. Ben-Yehuda, M., Day, M.D., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O., Yassour, B.A.: The turtles project: Design and implementation of nested virtualization. In: OSDI. (2010) 423–436
16. Villegas, D., Antoniou, A., Sadjadi, S.M., Iosup, A.: An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds. In: CCGRID. (2012) 612–619
17. Agmon Ben-Yehuda, O., Schuster, A., Sharov, A., Silberstein, M., Iosup, A.: Expert: Pareto-efficient task replication on grids and a cloud. In: IPDPS. (2012)
18. Iosup, A., Epema, D.H.J.: Grid computing workloads. IEEE Internet Computing **15**(2) (2011) 19–26
19. Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., Epema, D.H.J.: The grid workloads archive. Future Generation Comp. Syst. **24**(7) (2008) 672–686
20. Feitelson, D.: Parallel workloads archive. `http://www.cs.huji.ac.il/labs/parallel/workload/`
21. Iosup, A., Sonmez, O.O., Epema, D.H.J.: Dgsim: Comparing grid resource management architectures through trace-based simulation. In: Euro-Par. (2008) 13–25
22. Petrini, F., Fossum, G., Fernández, J., Varbanescu, A.L., Kistler, M., Perrone, M.: Multicore surprises: Lessons learned from optimizing sweep3d on the cell broadband engine. In: IPDPS. (2007) 1–10
23. Sonmez, O.O., Mohamed, H.H., Epema, D.H.J.: On the benefit of processor coallocation in multicluster grid systems. IEEE Trans. Parallel Distrib. Syst. **21**(6) (2010) 778–789
24. Shen, S., Deng, K., Iosup, A., Epema, D.: Scheduling jobs in the cloud using on-demand and reserved instances. In: Euro-Par. (2013) To Appear
25. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T.L., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP. (2003)
26. Clark, B., Deshane, T., Dow, E., Evanchik, S., Finlayson, M., Herne, J., Matthews, J.N.: Xen and the art of repeated research. In: USENIX ATC. (2004) 135–144
27. Menon, A., Santos, J.R., Turner, Y., Janakiraman, G.J., Zwaenepoel, W.: Diagnosing performance overheads in the Xen virtual machine environment. In: VEE. (2005) 13–23
28. Youseff, L., Seymour, K., You, H., Dongarra, J., Wolski, R.: The impact of paravirtualized memory hierarchy on linear algebra computational kernels and software. In: HPDC, ACM (2008) 141–152
29. Donassolo, B., Casanova, H., Legrand, A., Velho, P.: Fast and scalable simulation of volunteer computing systems using simgrid. In: HPDC. (2010) 605–612

30. Jacobson, V.: Congestion avoidance and control. In: SIGCOMM. (1988) 314–329
31. Iosup, A., Ostermann, S., Yigitbasi, N., Prodan, R., Fahringer, T., Epema, D.H.J.: Performance analysis of cloud computing services for many-tasks scientific computing. IEEE Trans. Parallel Distrib. Syst. **22**(6) (2011) 931–945
32. Feitelson, D.G.: Experimental analysis of the root causes of performance evaluation results: A backfilling case study. IEEE Trans. Parallel Distrib. Syst. **16**(2) (2005) 175–182
33. Jones, J.P., Nitzberg, B.: Scheduling for parallel supercomputing: A historical perspective of achievable utilization. In: JSSPP. (1999) 1–16
34. Markowitz, H.: Portfolio selection. The Journal of Finance **7**(1) (1952) 77–91
35. Gomes, C.P., Selman, B.: Algorithm portfolios. Artif. Intell. **126**(1-2) (2001) 43–62
36. Streeter, M.J., Golovin, D., Smith, S.F.: Combining multiple heuristics online. In: AAAI. (2007) 1197–1203
37. Bougeret, M., Dutot, P.F., Goldman, A., Ngoko, Y., Trystram, D.: Combining multiple heuristics on discrete resources. In: IPDPS. (2009) 1–8
38. Goldman, A., Ngoko, Y., Trystram, D.: Malleable resource sharing algorithms for cooperative resolution of problems. In: IEEE Congress on Evolutionary Computation. (2012) 1–8
39. Streeter, M.J., Smith, S.F.: New techniques for algorithm portfolio design. CoRR **abs/1206.3286** (2012)
40. Gagliolo, M., Schmidhuber, J.: Learning dynamic algorithm portfolios. Ann. Math. Artif. Intell. **47**(3-4) (2006) 295–328
41. Gagliolo, M., Schmidhuber, J.: Algorithm portfolio selection as a bandit problem with unbounded losses. Ann. Math. Artif. Intell. **61**(2) (2011) 49–86
42. Merton, R.C.: Optimum consumption and portfolio rules in a continuous-time model. MIT (1970)
43. Magill, M.J., Constantinides, G.M.: Portfolio selection with transaction costs. Journal of Economic Theory **13**(2) (1976) 245–263
44. Black, F., Scholes, M.: The pricing of options and corporate liabilities. The journal of political economy (1973) 637–654
45. Marshall, P., Keahey, K., Freeman, T.: Elastic site: Using clouds to elastically extend site resources. In: CCGRID. (2010) 43–52
46. den Bossche, R.V., Vanmechelen, K., Broeckhove, J.: Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In: IEEE CLOUD. (2010) 228–235
47. Palankar, M.R., Iamnitchi, A., Ripeanu, M., Garfinkel, S.: Amazon s3 for science grids: a viable solution? In: Proceedings of the 2008 international workshop on Data-aware distributed computing, ACM (2008) 55–64
48. Hu, J., Gu, J., Sun, G., Zhao, T.: A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In: PAAP. (2010) 89–96
49. Gao, Y., Rong, H., Huang, J.Z.: Adaptive grid job scheduling with genetic algorithms. Future Generation Comp. Syst. **21**(1) (2005) 151–161
50. Calheiros, R.N., Ranjan, R., Buyya, R.: Virtual machine provisioning based on analytical performance and qos in cloud computing environments. In: ICPP. (2011) 295–304
51. Ali-Eldin, A., Kihl, M., Tordsson, J., Elmroth, E.: Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control. In: ScienceCloud. (2012) 31–40