Exploring Plan-Based Scheduling for Large-Scale Computing Systems

Xingwu Zheng*, Zhou Zhou†, Xu Yang†, Zhiling Lan† and Jia Wang*

*Department of Electrical and Computer Engineering
xzheng18@hawk.iit.edu, jwang@ece.iit.edu
†Department of Computer Science
{xyang56,zzhou1}@hawk.iit.edu, lan@iit.edu
Illinois Institute of Technology, Chicago, Illinois, USA 60616

***Abstract*—As HPC systems scale toward exascale, it becomes critical to manage the underlying resource more effectively. While almost all existing resource management systems schedule jobs in a queuing fashion and have drawbacks of making isolated scheduling decisions that would compromise system performance even with backfilling, plan-based schedulers have the potential to generate better job schedules by producing an execution plan of all waiting jobs but do not receive enough attention.**

In this paper, we present a novel plan-based scheduling system that utilizes simulated annealing as the optimization engine to support effective resource management on HPC systems. As demonstrated by extensive trace-based simulations with workload traces collected from a wide range of production supercomputers, in comparison with the queue-based scheduling system using FCFS with EASY backfilling, our plan-based scheduling system can reduce the job wait time by 40%, reduce the job response time by 30%, while slightly improving system utilization at the same time. Moreover, our plan-based system is able to run online by solving the scheduling problem at each scheduling iteration within one second, making it practical for production HPC systems.

***Index Terms*—Plan-based scheduling, Simulated Annealing algorithm, Optimization**

I. Introduction

The ever increasing demand for computational resources, which is a fundamental requirement in both science and industry, is always driving the development of new generation of computing systems. We have seen in recent years a growing deployment of such large-scale computing systems consisting of massive number of tightly- or loosely-connected nodes [1]. To operate such large-scale systems, a resource management system is essential that handles large numbers of computing nodes, provides load balancing, and even manages heterogeneity for nodes of different architectures. One key responsibility of the resource management system is to provide efficient resource isolation and sharing across multiple parallelized or distributed applications. A common scenario is that the resource management system decides when and how to dedicate computing nodes to specific applications, which is often called "job scheduling" [2].

Most of the resource management systems available today can be classified as queue-based systems, including SLURM [3], PBS [4], Cobalt [5], Platform LSF [6], and IBM Loadleveler [7]. Job schedulers are used by these systems to operate multiple waiting queues, which have different priorities, properties or constraints (e.g., high priority queue, short job queue). Each incoming job request/submission is assigned to one of these queues. When there is available resource, the job scheduler makes the scheduling decision by selecting jobs from these queues in a certain order and allocating resource to these jobs until all available resource is consumed up. The most simple and commonly used strategy is FCFS (First Come, First Served), which allocates resource to jobs in chronological order according to their submission times in the system. Some modern resource management systems also use priority queues to do more fine-grained job categorization, reflecting the priority of a certain type of jobs [2]. Nevertheless, these strategies always start to allocate resource to jobs from the queue head. If the available resource cannot satisfy the job in the queue head, the job scheduler may use a strategy called "backfilling" to improve system utilization by allowing out-of-order executions of jobs without delaying the current job. First, the job scheduler reserves a possible start time in future for the first job in the queue given its resource requirement and the expected completion times of all running jobs. Then suitable jobs are selected from the rest of the queue to get allocated, whose expected runtime is shorter than the length of the reservation. Two well-known backfilling strategies are *conservative* and *EASY*. In particular, conservative backfilling [8] attempts to schedule as many jobs as possible following their ordering in the queue while EASY backfilling [9] [10] [11] stops at the first such job for simplicity. FCFS with EASY backfilling is a widely-used policy adopted by production systems.

Although a queue-based approach is easy to understand and implement, it inevitably has drawbacks resulting from its decision process that implicitly or explicitly depends on the job ordering defined by the queue. The scheduling decision is isolated in that every time the job scheduler only selects the first job in the queue, disregarding information that is not captured by the ordering, e.g. current system states, possible future scheduling choices, and complex job constraints such as deadlines [12] or SLA (Service Level Agreement) [13]. The same concern extends to backfilling strategies in that to

2168-9253/16 $31.00 © 2016 IEEE
DOI 10.1109/CLUSTER.2016.43

backfill the jobs following the queue ordering at a particular time point does not necessarily lead to the best overall system utilization and application performance. Moreover, queue-based approaches lack the ability to evaluate scheduling decisions beyond the immediate future. For example, in order to satisfy deadline bounded jobs, the scheduling system must have predictability to assess the effect of its scheduling decisions, which is not well solved by queue-based systems.

As an alternative, a plan-based scheduling system produces *an execution plan* by assigning a start time and location for each job in the waiting queue, considering not only currently available resource but also predictions on job execution time. As shown in Figure 1, in this manner, a "big picture" of the transition of system states encompassing all running and waiting jobs is generated, including which jobs are scheduled to execute now and how the following jobs are coordinated, leading to several major advantages of plan-based systems over queue-based systems. First, the execution plan enables the job scheduler to evaluate scheduling decisions and to find the best one not only for now but also for future since we have predictions of each job's start time, expected completion time, and the target nodes. We can have a clear and holistic view of how the system state transits based on the current scheduling decision. Second, since the execution plan holds very detailed scheduling information for each job, the scheduler is able to evaluate scheduling decisions using multiple criteria, e.g. wait time, response time and system utilization. It is impossible to do this in a queue-based system as a result of lack of such information known in advance. Third, a plan-based system can handle job requests with more complex requirements beyond what are provided by aforementioned queue-based scheduling policies, such as deadline or reservation, which are commonly seen on Grid computing platforms.

Nevertheless, as a plan-based system would need to explore a larger solution space of schedules than a queue-based system within the same short period of time that such scheduling decisions has to be made, most plan-based systems are based on not-so-powerful local search [14] or tabu search [15] [16] due to the lack of computational power to support more advanced optimization algorithms, leading to schedules that are only minimal locally. As a result, plan-based systems do not receive enough attention, and thus only a few solutions are available: some are out of date and not operational anymore; some are commercial software and not free [17]. Current plan-based systems are more used on the computational Grids, and are less popular on general clusters or supercomputers [18] [19].

In this paper, we present a new plan-based scheduling system to support effective resource management on HPC systems. Leveraging more powerful processors that become available recently, our plan-based system utilizes simulated annealing (SA) as the optimization engine to perform scheduling decisions online [20] [21]. SA gives our scheduler a great flexibility in choosing performance metric to be optimized, e.g. job wait time or system utilization, which is quite preferable for plan-based systems when complex requirements are presented. In comparison to evolutionary algorithms used in other plan-based systems, whose effectiveness heavily depends on how one encodes the solution space, by using SA our scheduler can use the most straightforward way to encode valid job schedules. Moreover, in comparison to local search or tabu search those are usually trapped at local optimums, the stochastic nature of SA allows our scheduler to move out of local optima toward the global optimum.

We demonstrate that our plan-based system can be incorporated into queue-based systems as an alternative scheduling scheme by implementing it inside the queue-based HPC simulator CQSim [22] [23]. We further evaluate the effectiveness of our proposed plan-based system by performing extensive trace-based simulations. Our simulations use publicly available [24] real workload traces collected from a wide range of production supercomputers including IBM BG/P from Argonne National Lab (ANL), Altas from Lawrence Livermore National Lab (LLNL), and Blue Horizon from San-Diego Supercomputer Center. In comparison to the queue-based system using FCFS with EASY backfilling, the simulation results show that our plan-based system can reduce the job wait time by 40%, reduce the job response time by 30%, while slightly improve system utilization at the same time. Moreover, the simulations confirm that our plan-based system can indeed work online by completing each scheduling optimization within one second.

The rest of the paper is organized as follows. We first discuss background on scheduling in Section II. Then, an overview of our plan-based system is presented in Section III, followed by its details in Section IV. Our evaluation methodology is introduced in Section V. The experimental results are presented in Section VI. Finally, Section VII concludes the paper.

II. Background

Most job scheduling systems available today are queue-based systems. In such systems, one or more queues are created with different priorities, properties, or constraints. The objective of a queue-based scheduling system is to assign available resource to waiting jobs following the queue ordering. A very simple strategy is FCFS (First Come, First Served), which assigns resources to the jobs according to their submission times [25]. A more comprehensive strategy may assign priorities to jobs, e.g. in Shortest/Longest Job First scheduling [4], or according to certain fair-sharing policy [26], in order to prioritize particular jobs ahead of its ordering by submission time. However, these approaches usually suffers from fragmentation, where the next job to be scheduled at the queue head may block all following jobs if there is not enough resource for it. The resource that is currently free has to remain idle and wasted, leading to low system utilization [10].

To resolve the issue of fragmentation, backfilling [9] were introduced to allow the small jobs from the back of the queue to execute before the larger jobs that arrived earlier. Two backfilling variants are commonly used, named EASY [9] [10] [11] and conservative [8] backfilling. In particular, EASY, which is widely adopted by production systems for its simplicity, allows

any job to be backfilled providing that it does not delay the first job in the queue. On the other hand, for conservative backfilling, backfilling of a job arrived later is subject to that it does not delay *any* previous job in the queue. The algorithm goes through the rest of the waiting queue to consider each job for "backfilling" following their ordering in the queue. In each iteration, the algorithm attempts to schedule the job to start as early as possible, usually by performing a search for the resource from present to the reserved start time.

Nevertheless, due to their greedy nature where the decisions are made for the rest of the waiting jobs one-by-one following the queue ordering, both EASY and conservative backfilling algorithms only explore a small portion of the solution space consisting of all valid schedules of the waiting jobs. More sophisticated policies were proposed to explore a larger portion of the solutions space by relaxing the strict queue ordering. Lookahead Optimizing Scheduler(LOS) was proposed in Shmueli et al. [27] to improve system performance by maximizing system utilization at present for every scheduling step. To achieve so, a 0-1 knapsack problem to maximize the resource usage is formulated at each scheduling step by considering the waiting jobs that can be started immediately, and is solved via dynamic programming. The authors further indicated that empirically the scheduler running time can be reduced without impacting system performance by only considering 50 waiting jobs. In our previous work [28] [29] [22], window-based scheduling was proposed to allow the scheduler to make decisions upon a "window" of waiting jobs instead of individual ones, making it possible to optimize system utilization and job performance simultaneously at present by solving 0-1 multiple knapsack problems.

Although the aforementioned approaches did explore a larger portion of the solution space than the classical backfilling policies, to maximize the *present* system performance in a greedy manner may miss certain opportunity to optimize the *overall* system performance. For example, consider the jobs in Fig. 1 (a) that are waiting to be scheduled onto an 8-node system. The width of a job represents its expected runtime and the height represents the number of nodes that it requires. The job schedule in Fig. 1 (b) achieves 100% system utilization at present, though the overall utilization and the makespan are worse than the schedule in Fig. 1 (c), which does not achieve 100% system utilization for any instant.

The schedule in Fig. 1 (c) that maximizes the overall system utilization and minimizes the makespan for the jobs in Fig. 1 (a) could be obtained by a *plan-based* scheduling system [18] [19]. Different from a *queue-based* scheduling system that may prefer to schedule {J1, J2, J3, J4} to run immediately (as depicted in Fig. 1(b)) in order to fully utilize the available nodes at the moment, a plan-based scheduling system may discover that better overall system utilization can be obtained by not utilizing all the available nodes for now. Such discoveries are usually driven by a global optimization technique in the scheduler that assigns every job in the waiting queue a start time considering resource availability hereafter. Furthermore, as the scheduler is able to predict the resource

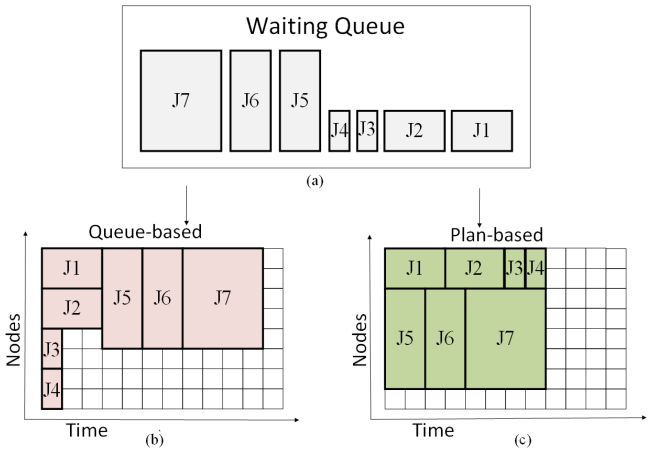


Fig. 1: Comparison of queue-based scheduling and plan-based scheduling. (a) Jobs in the waiting queue; (b) possible result from a queue-based scheduling system; (c) possible result from a plan-based scheduling system. The height and width of a job represent its required nodes and runtime respectively.

usage at any specified time, it is also possible for plan-based scheduling systems to consider other system performance metrics. For example, the mean wait time of the schedule in Fig. 1 (c) is slightly smaller (better) than that in Fig. 1 (b). Note that when a new job is submitted and there are jobs whose previously planned start times have not arrived yet, a plan-based scheduler may perform *replanning* [19] to reassign start times for existing jobs and to assign start time for the new job.

A few plan-based scheduling systems have been reported in the literature. Computing Center Software [19] (CCS) combined FCFS, Shortest Job First, and Longest Job First strategies with a backfilling-like policy to generate a complete schedule by assigning start times to each resource request. While the emphasis was to demonstrate the unique features that plan-based resource management systems may implement but queue-based ones cannot, no quantitative comparison to alternatives was provided. For grid systems, Global Optimising Resource Broker and Allocator (GORBA) [18] used certain simple policies for schedule creation and relied on an evolutionary algorithm for its optimization, though the impact to system performance was not reported. On the other hand, Dalibor et al. [30] focused on the plan-based methods that allow to handle both Quality of Service requirements from Grid users and system performance metrics. Starting from an initial scheduling, when a new job is submitted, Tabu search was applied to optimize each successive scheduling incrementally in order to reduce the running time required for replanning. More recently, the TORQUE [17] Resource Manager was implemented within a production system to support plan-based scheduling. The initial scheduling was constructed by conservative backfilling [8]. Still concerning by the running time for replanning, the authors applied a

local search based method instead of advanced optimization to obtain successive schedules when new jobs are submitted.

Our work on plan-based systems differs from aforementioned works as we have adopted Simulated Annealing (SA) [20] [31] as the optimization engine. SA gives our scheduler a great flexibility in choosing performance metric to be optimized, while being efficient enough to perform scheduling decisions online. Unlike evolutionary algorithms whose effectiveness heavily depends on how one encodes the schedules [18], by using SA our scheduler can use the most straightforward way to encode valid job schedules. In comparison to local search or tabu search that are usually trapped at local optimums, the stochastic nature of SA allows our scheduler to move out of local optima toward the global optimum. As we will show later on in Section VI, extensive trace-based simulation confirms that our plan-based scheduler can improve both system utilization and job performances.

Note that all schedulers will need to make decisions upon the expected job running times reported by the users at the time of job submissions, which are not necessarily accurate. While the impact of such inaccuracy on queue-based systems was studied [28] [11] [32] [33] [34], we do not consider this issue here and would leave it to a future work.

III. System Overview

The architecture of our proposed plan-based scheduling system is shown in Fig. 2. We consider a HPC system managed by a centralized scheduler that has complete control over all jobs and system resources. We define a scheduling iteration as the instant that a certain event takes place on the HPC system, including job submission, job start, normal job end, and job termination if the job exceeds its expected runtime. Upon each scheduling iteration, all jobs that are waiting to be executed are held in the waiting queue. Different from queue-based schedulers that will process the jobs according to their order in the waiting queue, our plan-based scheduler will generate *an execution plan of all the waiting jobs* consisting of their planned start times. The execution plan will consider the present and expected future resource availability, as well as be optimized for certain performance metric. The waiting jobs are extracted from the queue to the execution queue sorted by their planned start times, allowing the HPC system to start a job at the head of the executing queue when its planned start time arrives. Note that at each scheduling iteration, our scheduler will re-plan all the jobs that are not started yet based on current system state, and thus may rewrite the whole execution queue to update the execution plan.

IV. Plan-Based Scheduling

The details of our plan-based scheduler are presented in this section. We formulate the plan-based scheduling problem as an optimization problem, provide insights into the choice of performance metrics as optimization objectives, and design a heuristic algorithm based on Simulated Annealing (SA) to solve the problem.

A. Problem Formulation

As mentioned in Section III, our plan-based scheduler considers all the jobs in the waiting queue for each scheduling iteration. For each job, our scheduler needs to know its resource usage, measured by the number of nodes, and its expected runtime, measured by any unit of time, both reported by the user at the time of job submission. Moreover, our scheduler also needs to know what resource is available currently and in the future, which can be estimated from the start times of the jobs currently running on the HPC system and their resource usages and expected runtimes. With such information, our scheduler generates an execution plan of all the waiting jobs, optimized for certain performance metric. The output of our scheduler, denoted as the *execution plan*, consists of all the waiting jobs ordered by their planned start times.

Hence, the *plan-based scheduling problem* is formulated as an optimization problem as follows.

Plan-Based Scheduling Problem: Given the current system state consisting of the start time, the expected runtime, and the resource usage for each running job, the scheduler should generate an execution plan for the jobs in the waiting queue that assigns each waiting job a start time in order to minimize certain performance metric.

Note that since our plan-based scheduler works in an online mode, it is critical for the above plan-based scheduling problem to be solved efficiently in a short time period (e.g., in seconds).

B. Performance Metrics for Optimization

The choice of the performance metric to be optimized by our plan-based scheduler at each scheduling iteration will impact the overall performance of the HPC system. As job wait time and system utilization are among the most important scheduling performance metrics for HPC systems, we propose to consider the following three optimization metrics. Three plan-based scheduling policies, each optimizing one of the three metrics, are denoted by PLAN 1, PLAN 2 and PLAN 3 respectively for ease of presentation.

- *Mean Job Wait Time (PLAN 1)*. For each job, its wait time refers to the time elapsed between the moment when it is submitted and the moment when it starts to run. Intuitively, the mean job wait time that is calculated as the average across all the jobs in the waiting queue can be minimized by our scheduler for each execution plan in order to minimized the overall mean job wait time.
- *Mean Squared Job Wait Time (PLAN 2)*. The mean job wait time as a metric in PLAN 1 to optimize job schedules cannot prevent one job to be postponed arbitrarily by many other jobs as long as their total wait time remains the same. Such pathetic case may cause job starvation that is quite unfriendly for users. Therefore, we propose to consider mean squared job wait time as a metric, which is calculated as the average of the job wait time timed by itself for all the jobs in the waiting queue. Intuitively, the mean squared job wait time could be treated as a

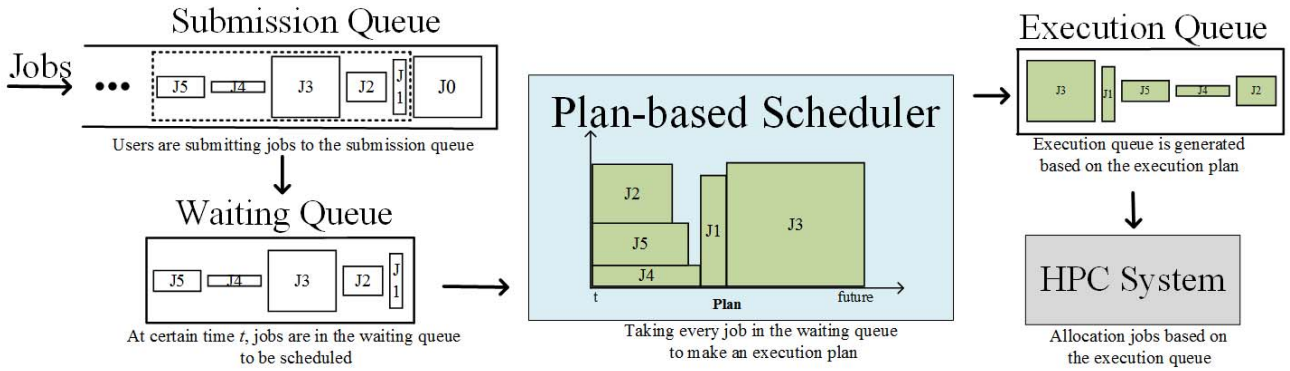


Fig. 2: Our proposed plan-based scheduling system.

weighted average of the job wait times, where the weights change dynamically to prioritize jobs that have already been waiting for a long time, allowing our scheduler to schedule them to start earlier.

- *System Utilization (PLAN 3).* For each execution plan, we define its system utilization to be the ratio of the total node-hours used by the jobs to the total elapsed system node-hours from the present to when all the running and queued jobs complete. Clearly, since the total node-hours used by the jobs is a constant for all job schedules at a particular scheduling iteration, to maximize the system utilization is equivalent to minimize the finish time of all the jobs in our plan-based scheduling problem.

C. Scheduling Optimization by Simulated Annealing

We solve the plan-based scheduling problem using simulated annealing (SA), which is a stochastic metaheuristic originated from the physical heating and annealing process that removes defects from a material [20]. As an iterative metaheuristic for optimization, SA depends on the choice of a neighbourhood structure defined by a strategy to move from one solution to another solution in the solution space, and on the choice of a cost function that maps a solution to a positive real number that should be minimized. Unlike mathematical optimizations that are effective for continuous solution spaces with special properties like convexity, SA is usually more effective when the solution space has no such properties, e.g. for the plan-based scheduling problem as we will discuss below. On the other hand, unlike other metaheuristics that may also optimize with the neighbourhood structure and the cost function, e.g. greedy algorithm and tabu search, SA is less likely to be trapped into local minima due to its stochastic nature. While moving from solutions with higher costs to those with lower costs are always accepted, SA introduces a variable named "temperature" to control the probability to accept a move from a solution with a lower cost to a solution with a higher cost, which is inspired by the annealing process in heat treatment that uses temperature to control changes in material. SA starts with a high temperature where such moves are more likely to be accepted so it will not be trapped to local minima, and gradually decreases the temperature so the effort is more focused on improving solutions in a greedy manner by the end of the optimization.

We choose the cost function to be the same as the performance metric that need to be minimized in our plan-based scheduling problem. To define the neighbourhood structure that can be explored effectively and efficiently by SA, we would need to first consider the solution space of our plan-based scheduling problem. Obviously we should only consider those *valid* scheduling solutions where the resource usage limit is observed for any instant. Since from the user's perspective one would prefer to start a job as soon as possible, we further restrict the solution space to the scheduling solutions where it is impossible to start any job earlier without delaying any other jobs. Such restriction is essential since there are only *finite* number of such solutions, and we may *represent* each such solution by a permutation of jobs in the waiting queue.

More specifically, for each job schedule, we obtain a permutation P of the jobs in the waiting queue by ordering them according to their starting times (where ties are broken arbitrarily), and no two solutions may lead to the same permutation. On the other hand, once a permutation P of the jobs in the waiting queue is given, we may obtain a job schedule by scheduling them one at a time as early as allowed by the resource usage following the ordering of P. Therefore, if there are n jobs in the waiting queue, there will be at most $n!$ such scheduling solutions. The neighbourhood structure of the solution space for SA is thus defined upon *the permutation representation*: for a solution represented by a permutation P, we define its neighboring solutions to be the permutations obtained by removing an arbitrary job and then inserting it back to an arbitrary position in P.

Nevertheless, we cannot evaluate the cost function directly upon the permutations since the cost function depends on the actual times that the jobs are scheduled to run. It is therefore necessary to actually compute an execution plan from a permutation P by assigning each job in the waiting queue a start time. Moreover, this computation has to be done very efficiently since its running time dominates the running time of the SA optimization. To support this computation, We use a

linked list of tuples to represent the available resource, where each tuple (t, a) denotes that there are a nodes available from time t, until the time specified by the next tuple if (t, a) is not the last in the list. We use the following procedure to compute an execution plan from P. We first initializes the list using the expected finish time and the resource usage of each running job. The size of the list is at most $m + 1$ if there are m running jobs. Then we schedule each job in the waiting queue following the ordering of P by scanning the list from the front to the end in order to find the earliest time that the job may start at. The list will need to be updated reflecting the newly scheduled job, possibly with one more tuple created at its expected finish time. Therefore, if there are n jobs in the waiting queue and m running jobs, the size of the list is bounded by $m+n+1$, and to find the start time of each waiting job takes $O(m+n)$ time. Overall, the space and the time complexities of each computation are $O(m+n)$ and $O(n(m+n))$ respectively.

Algorithm 1 Plan-Based Scheduling by Simulated Annealing

Input: *Running and waiting jobs, cost function, SA parameters.*
Output: *An optimized execution plan.*

```
1: Generate an initial permutation P of waiting jobs.
2: Memorize P as P_best.
3: Initialize temperature T as T_0.
4: while T > T_th do
5:     for i = 1 to N do
6:         Randomly generate P' from P.
7:         if P' is better than P then
8:             P ← P'.
9:         else
10:            P ← P' probabilistically.
11:        end if
12:        Memorize P as P_best if P is better than P_best.
13:    end for
14:    T ← rT.
15: end while
16: Compute execution plan from P_best and return.
```

The pseudocode to solve our plan-based scheduling problem by SA is shown in Algorithm 1. We generate the initial permutation following the ordering of the jobs in the waiting queue. The progress of simulated annealing is controlled by four SA parameters T_0, T_{th}, N, and r: T_0 is the initial temperature; the outer loop continues as long as the temperature T is higher than T_{th} and it reduces T each iteration by $0 < r < 1$; the inner loop attempts N moves at each temperature, accepting those that improve the solution P, and those that degrade P using a probability function [20]. The best permutation seen so far is memorized in P_{best} throughout and is used to compute the result execution plan. If a sufficiently long time is allowed for the SA run, selecting appropriate parameter values may not be an issue, since an SA algorithm with a high initial temperate and a slow cooling scheme performs well. However, our plan solution has to be found by an SA algorithm within the typical 30-second [29] time frame that the HPC scheduler has to make a decision. An experimental design has often been used for selecting parameter value [20] [31], we omit here for simplicity. In our experiments, we set the parameters (T_0, T_{th}, N, r) to $(1, 0.0001, 100, 0.9)$. This parameter setting allows us to solve the plan-based scheduling problem at each schedule decision-making in one second and at the same time results in job schedules with good quality as discussed in Section VI.

V. EVALUATION METHODOLOGY

A. Trace-Based Scheduling Simulation Using CQSim

We implement our plan-based scheduling system in the event-driven HPC scheduling simulator CQSim [23]. and evaluate its performance under a variety of workloads using extensive trace-based simulations. CQSim reads job events like job submission, job start, and job complete from a trace file in order to emulate job scheduling on HPC systems based on specific scheduling and allocation policies [22]. In particular, CQSim supports scheduling by FCFS with EASY backfilling, which will be compared to our proposed plan-based scheduling policies PLAN 1, PLAN 2, and PLAN 3.

B. Production Job Traces

Trace	#nodes	#jobs	Duration
ANL	40,960	68,936	Jan/09-Sep/09
LLNL	9,216	60,332	Nov/06-Jun/07
SDSC	1,152	67,155	May/00-Dec/01

TABLE I: Production workload traces used for evaluation.

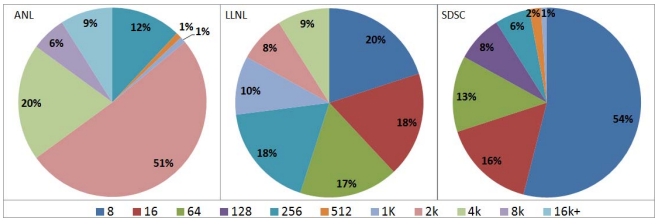


Fig. 3: Distribution of job sizes.

We use three publicly available real workload traces collected from a wide range of production supercomputers including Intrepid from Argonne National Lab (ANL), Altas from Lawrence Livermore National Lab (LLNL), and Blue Horizon from San-Diego Supercomputer Center (SDSC) [24]. Each trace contains a list of jobs, each consisting of a record of its size, arrival time, actual and estimated runtime, and other descriptive fields. The overall statistics of the three traces are given in Table I.

Fig. 3 further illustrates the distribution of job sizes in these traces. It can be seen from the figure that these traces represent the variety of job scheduling problems a scheduler would need to solve. In particular, ANL represents capability computing to solve large problems, whereas LLNL and SDSC represent capacity computing to solve a large number of middle-sized

and small-sized problems respectively. In the ANL trace, half of the jobs require 2K nodes. On the other hand, while the number of large-sized jobs that require more than 16K nodes is relatively low, these jobs consume a considerable amount of node-hours due to their sizes. In the LLNL trace, job sizes spread more evenly with jobs requiring 1K nodes being the majority as 20%. In the SDSC trace, more than half of the jobs require 8 nodes only. We will discuss the impact of job sizes on different scheduling policies in Section VI.

C. Evaluation Metrics

Although our plan-based scheduling policies PLAN 1, PLAN 2 and PLAN 3 use different metrics to optimize job schedules, we compare them with FCFS with EASY backfilling using a fixed set of metrics measuring system and individual job performances, including *job wait time* and *system utilization* as discussed in Section IV-B, as well as *job response time*, which is defined as the amount of time from when a job is submitted until the job completes, i.e. its wait time plus its runtime.

VI. Experimental Results

A. System Scheduling Performance

Trace	FCFS	PLAN 1	PLAN 2	PLAN 3
ANL	0.631	0.641	0.640	0.642
LLNL	0.744	0.750	0.751	0.758
SDSC	0.780	0.785	0.782	0.793

TABLE II: Comparison of system utilization under different scheduling policies.

The system utilizations from our trace-based simulations for all 4 scheduling policies including our plan-based PLAN 1, PLAN 2, and PLAN 3, as well as the queue-based FCFS with EASY backfilling, which will be denoted as FCFS hereafter for ease of presentation, are compared in Table II for all three traces. It is clear that our plan-based policies always lead to better system utilization than FCFS. This is even true for PLAN 1 and PLAN 2 where the system utilization is *not* the objective of optimization. Nevertheless, PLAN 3 achieves the highest system utilization among the 4 policies as it explicitly optimizes for system utilization.

The mean job wait times and mean job response times for all policies and all traces are compared in Fig. 4 and Fig. 5 respectively for each month. It can be seen that our plan-based policy PLAN 2 always lead to better mean job wait time and mean job response time than FCFS. PLAN 1 and PLAN 3, on the other hand, outperform FCFS in all but two months 3 and 4 for LLNL. Interestingly, the figures show that our choice to optimize for mean squared job wait time in PLAN 2, though sometimes may lead to worse results than PLAN 1, allows PLAN 2 to reduce job wait time and job response time in comparison to FCFS more consistently for cases where PLAN 1 cannot.

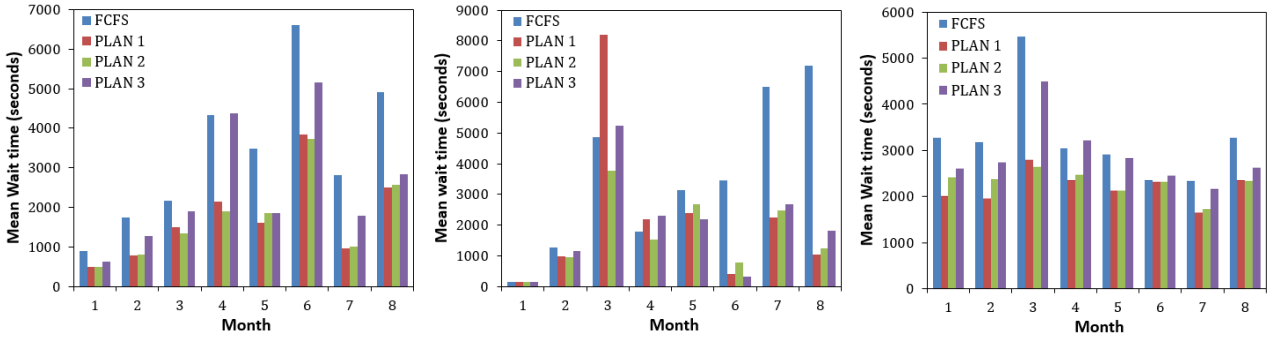
For our plan-based policies, the relative reductions in mean job wait time, and similarly in mean job response time, differ for different traces in comparison to FCFS. For LLNL in Fig. 4 (b), we see that in month 6-8, the mean job waiting times by our plan-based policies are much smaller than that of FCFS. This is most likely due to the fact that plan-based schedulers are allowed to pick any job from the job waiting queue to start freely and thus may avoid the pathetic case for FCFS where a large job is stuck in the head of waiting queue, blocking all smaller jobs submitted later. For ANL in Fig. 4 (a), the improvements by PLAN 1 and PLAN 2 over FCFS are more consistent across all the months, achieving 40% of reduction on average, which is quite significant considering that ANL has the largest system size among the three traces. On the other hand, although the improvements by PLAN 3 over FCFS can also be observed for most of the months in Fig. 4 (a), the reductions are not as large as PLAN 1 and PLAN 2 as the job wait time is not the objective of optimization for PLAN 3.

B. Individual Job Performance

As we can see from the previous subsection, our plan-based scheduling system is able to improve system performance in comparison to FCFS for system utilization, mean job wait time, and mean job response time. However, it is not possible to measure individual job performance using these metrics as they may vary significantly for different jobs. Since the wait time for any job is affected by its size and runtime, we therefore measure individual job performance by considering mean wait time for jobs with similar sizes and runtimes. We present here the results for PLAN 2 and FCFS for the sake of brevity and note that the results for PLAN 1 and PLAN 3 show similar trends.

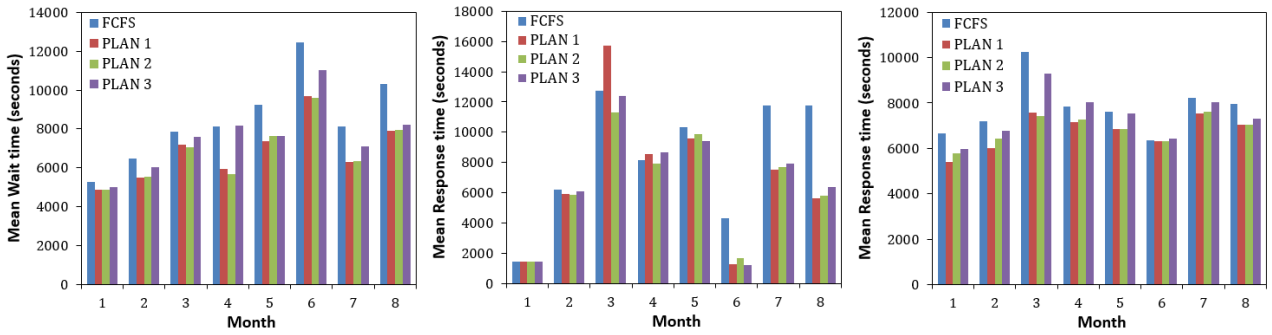
The baseline mean wait times for jobs binned with their sizes and runtimes obtained by FCFS are visualized in Fig. 6 (a)(d)(g) for the three traces respectively. These figures are created by dividing the job size range and the job runtime range of each job traces into 15 logarithmic bins respectively. Each point in the figure is colored according to the mean job wait time of the bin. The areas in the figures that are uncolored are empty bins without job. Generally speaking, the jobs in the blue bins are with less wait time, while those in the red bins are stuck in the waiting queue for longer time. The mean wait times for jobs obtained by our PLAN 2 policy are visualized in Fig. 6 (b)(e)(h) for the three traces respectively, using the same bin setting as that of FCFS. Overall, for all three traces, we can observe from Fig. 6 that for both FCFS and PLAN 2, jobs requiring less nodes and shorter runtime have smaller wait time, and jobs requiring more nodes and longer runtime would have to stay in the waiting queue longer. This is the case for FCFS because smaller jobs will have a better chance for backfilling. For PLAN 2, this is the case because it is more effective to optimize the overall job wait time by reducing wait times for smaller and shorter jobs, and larger and longer jobs may still need to wait longer due to lack of system resource.

To provide a comparative view of the two policies FCFS and PLAN 2, we further calculate the ratios of the mean job wait times by PLAN 2 to those by FCFS, and visualize them in Fig. 6 (c)(f)(i), where blue means our PLAN 2 policy



(a) ANL (b) LLNL (c) SDSC

Fig. 4: Comparison of mean job wait time under different scheduling policies.



(a) ANL (b) LLNL (c) SDSC

Fig. 5: Comparison of mean job response time under different scheduling policies.

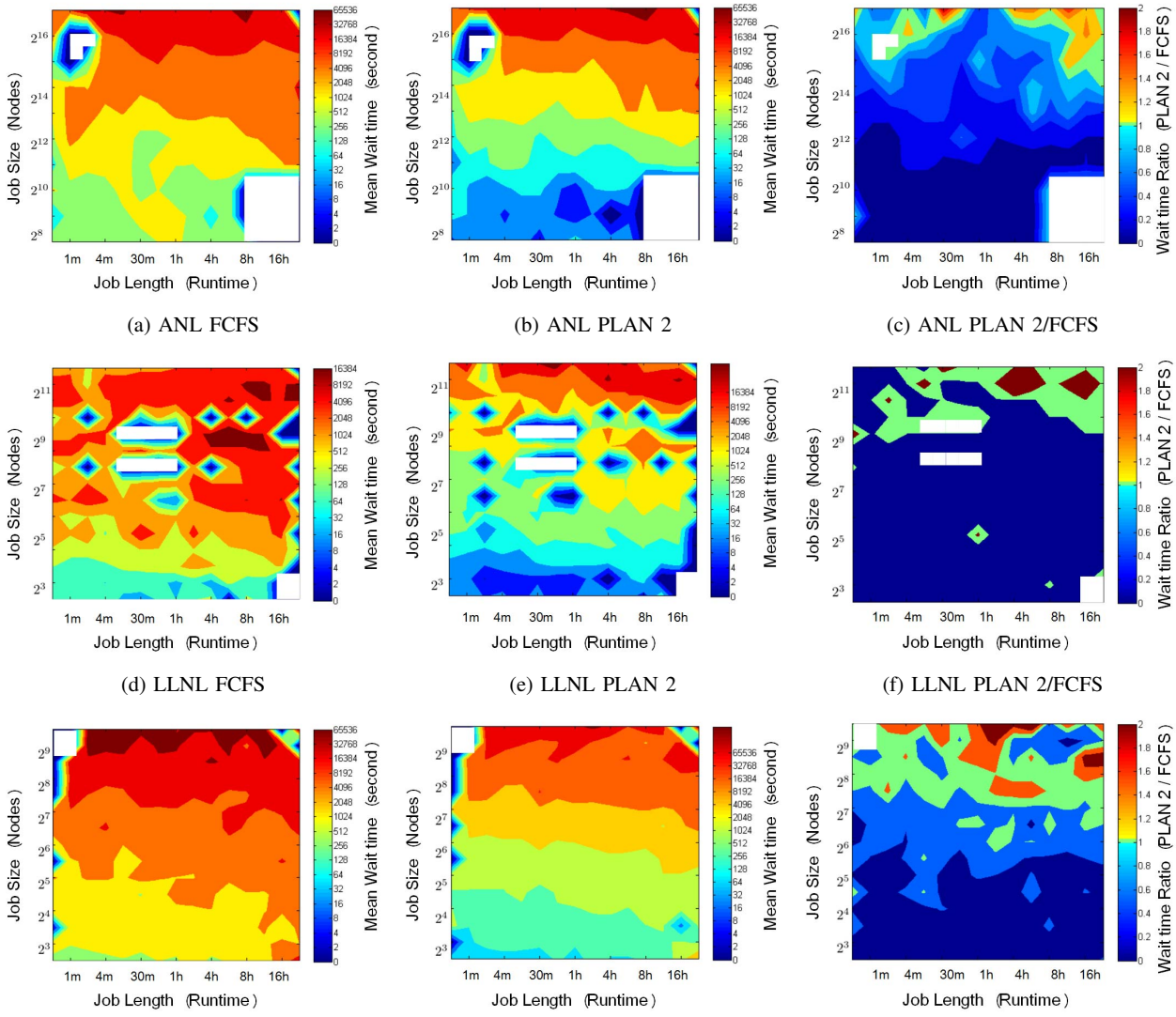
outperforms FCFS with better wait time and red means FCFS has better wait time than PLAN 2. These figures show that for most of the bins, our PLAN 2 policy outperforms FCFS. We find that PLAN 2 almost always achieves much smaller wait times than FCFS for smaller and shorter jobs. One possible reason is that when the system is actually saturated, which is a state characterized by a continuously growing length of waiting queue, our plan-based scheduler would prefer to run smaller and shorter jobs first instead of to run a larger and longer job first, as the later may block the former for a long waiting time.

On the other hand, for larger and longer jobs, it can be seen that PLAN 2 also achieves significant reduction of wait times for ANL and SDSC, indicating PLAN 2 is also helpful for these kind of jobs. For the LLNL in Fig. 6 (f), we observe that the color turns into dark red close to the top right corner, indicating that FCFS achieves better wait times than PLAN 2 for jobs that are both large and long. This generally shows how PLAN 2 makes trade-off wait times among different jobs in order to optimize for the overall system performance – as only 9% of the jobs in LLNL are in those bins requiring 4K nodes, PLAN 2 makes them to wait at most 2 times as long as that of FCFS while allowing smaller and shorter jobs to start much sooner so that the overall system performance can be greatly improved. Nevertheless, if one would prefer to explore different trade-offs among smaller and larger, as well as shorter and longer jobs, it should be possible to adjust the performance metric for optimization as discussed in Section IV-B for our plan-based scheduler, which would be leaved as a future work.

C. Result Summary

Overall, the following key observations can be made from the above trace-based simulations:

- Compared to traditional queue-based scheduling, plan-based scheduling can achieve much better system performance even under high loads in terms of mean job wait time, mean job response time, and system utilization. Our plan-based scheduler can deliver over 40% reduction on average job wait time, 30% reduction on average job response time, and 1% improvement on system utilization, varying depending on the different choices of optimization metrics.
- In addition to the improvements of system performance, plan-based scheduling can improve performance of certain kinds of jobs depending the choice of optimization metrics. In particular, if overall system performance is optimized, our plan-based scheduler can reduce job wait time for smaller and shorter jobs significantly without delaying larger and longer jobs. Further, our scheduler often manages to start the small-size jobs immediately



(a) ANL FCFS (b) ANL PLAN 2 (c) ANL PLAN 2/FCFS

(d) LLNL FCFS (e) LLNL PLAN 2 (f) LLNL PLAN 2/FCFS

(g) SDSC FCFS (h) SDSC PLAN 2 (i) SDSC PLAN 2/FCFS

Fig. 6: **(a)(b)(d)(e)(g)(h) Mean job wait time, binned by size and length of job: blue for jobs with less wait time, red for jobs stuck in the waiting queue for longer time. (c)(f)(i) Wait time ratio between PLAN 2 and FCFS, binned by size and length of job: each point is colored according the mean wait job time of PLAN 2 in each bin divided by that of FCFS, blue for PLAN 2 to outperform FCFS, red for FCFS to have less wait time. Similar results are observed for PLAN 1 and PLAN 3 (not shown).**

when their wait times are longer by using FCFS with EASY backfilling.

- When the job requests contain huge amount of small-size jobs, we encourage the use of PLAN 2 in order to improve individual job performance, and if system utilization is more important, PLAN 3 is a better choice.

VII. Conclusion and Future Work

In this paper, we have presented a novel plan-based scheduling system to support effective resource management on HPC systems. Our design was motivated by the observation that existing queue-based schedulers have drawbacks of making isolated decision that would compromise the system performance even with backfilling. Using a plan-based scheduler, users can better plan their work as the system's behavior becomes more predictable. Three performance metrics, namely mean job wait time, mean square job wait time, and system utilization have been investigated for our plan-based scheduler that used simulated annealing for optimization. Extensive trace-based simulations with traces from production HPC

systems demonstrate that in comparison to the queue-based scheduling system using FCFS with EASY backfilling, our plan-based system can reduce the job wait time by 40%, reduce the job response time by 30%, while slightly improving system utilization at the same time. Moreover, our plan-based system can be run online by solving the scheduling problem at each scheduling iteration within one second, which makes it practical for using on production HPC systems.

Several avenues are open for future work. One is to further extend the capabilities of our design so that it can meet additional requirements, e.g. job priorities. In addition, we would like to investigate the impact of inaccurate expected job runtimes as provided by the users on our plan-based scheduling system. Furthermore, it would be very useful to explore various methods to further reduce the algorithm running time in order to explore a larger portion of the solutions space for a better job schedule.

ACKNOWLEDGMENT

We appreciate the valuable comments and suggestions from the anonymous reviewers. This work is supported in part by US National Science Foundation grant CNS-1320125 and CCF-1422009.

REFERENCES

[1] TOP500, "Top500 supercomputing web site." available online http://www.top500.org.

[2] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *Job scheduling strategies for parallel processing*. Springer, 1997, pp. 1–34.

[3] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*. Springer, 2003, pp. 44–60.

[4] "PBS Works." available online http://www.pbsworks.com.

[5] "Cobalt Resource Manager." available online https://trac.mcs.anl.gov/projects/cobalt.

[6] "Platform LSF." available online http://www.platform.com/workload-management/high-performance-computing.

[7] "IBM Load Leveler." available online http://www.redbooks.ibm.com/abstracts/sg246038.html.

[8] D. G. Feitelson *et al.*, "Utilization and predictability in scheduling the IBM SP2 with backfilling," in *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International... and Symposium on Parallel and Distributed Processing 1998*. IEEE, 1998, pp. 542–546.

[9] D. Lifka, M. Henderson, and K. Rayl, "Users guide to the Argonne SP scheduling system, mathematics and computer science division technical memorandum no," *ANL/MCS- TM*, vol. 201, 1995.

[10] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, "Parallel job scheduling—a status report," in *Job Scheduling Strategies for Parallel Processing*. Springer, 2004, pp. 1–16.

[11] A. W. M. Alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 12, no. 6, pp. 529–543, 2001.

[12] W. Smith, I. Foster, and V. Taylor, "Scheduling with advanced reservations," in *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*. IEEE, 2000, pp. 127–132.

[13] R. Kübert and S. Wesner, "Service level agreements for job control in high-performance computing," in *Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on*. IEEE, 2010, pp. 655–661.

[14] D. Klusácek, L. Matyska, H. Rudová, R. Baraglia, and G. Capannini, "Local search for grid scheduling," in *Doctoral Consortium at the International Conference on Automated Planning and Scheduling (ICAPS 2007), Providence, RI, USA*, 2007.

[15] E. Alba, J. Carretero, B. Dorronsoro, and F. Xhafa, "Tabu search algorithm for scheduling independent jobs in computational grids," 2009.

[16] D. Klusáček, H. Rudová, R. Baraglia, M. Pasquali, and G. Capannini, "Comparison of multi-criteria scheduling techniques," in *Grid Computing*. Springer, 2008, pp. 173–184.

[17] D. Klusáček, V. Chlumskỳ, and H. Rudová, "Planning and optimization in TORQUE resource manager," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2015, pp. 203–206.

[18] W. Süß, W. Jakob, A. Quinte, and K.-U. Stucky, "GORBA: A global optimising resource broker embedded in a grid resource management system." in *IASTED PDCS*, 2005, pp. 19–24.

[19] M. Hovestadt, O. Kao, A. Keller, and A. Streit, "Scheduling in HPC resource management systems: Queuing vs. Planning," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2003, pp. 1–20.

[20] P. Salamon, P. Sibani, and R. Frost, *Facts, conjectures, and improvements for simulated annealing*. Siam, 2002.

[21] A. YarKhan and J. J. Dongarra, "Experiments with scheduling using simulated annealing in a grid environment," in *International Workshop on Grid Computing*. Springer, 2002, pp. 232–242.

[22] X. Yang, Z. Zhou, S. Wallace, Z. Lan, W. Tang, S. Coghlan, and M. E. Papka, "Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 60.

[23] "CQSim: An event-driven simultor." available online http://bluesky.cs.iit.edu/cqsim.

[24] "Parallel Workloads Archive." available online http://www.cs.huji.ac.il/labs/parallel/workload.

[25] K. Aida, "Effect of job size characteristics on job scheduling performance," in *Job Scheduling Strategies for Parallel Processing*. Springer, 2000, pp. 1–17.

[26] N. D. Doulamis, A. D. Doulamis, E. A. Varvarigos, and T. A. Varvarigou, "Fair scheduling algorithms in grids," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, no. 11, pp. 1630–1648, 2007.

[27] E. Shmueli and D. G. Feitelson, "Backfilling with lookahead to optimize the packing of parallel jobs," *Journal of Parallel and Distributed Computing*, vol. 9, no. 65, pp. 1090–1107, 2005.

[28] W. Tang, N. Desai, D. Buettner, and Z. Lan, "Analyzing and adjusting user runtime estimates to improve job scheduling on the Blue Gene/P," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–11.

[29] X. Yang, Z. Zhou, W. Tang, X. Zheng, J. Wang, and Z. Lan, "Balancing job performance with system performance via locality-aware scheduling on torus-connected systems," in *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*. IEEE, 2014, pp. 140–148.

[30] D. Klusacek and H. Rudova, "Improving QoS in computational grids through schedule-based approach," in *Scheduling and Planning Applications Workshop at the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008), Sydney, Australia*, 2008.

[31] M.-W. Park and Y.-D. Kim, "A systematic procedure for setting parameters in simulated annealing algorithms," *Computers & Operations Research*, vol. 25, no. 3, pp. 207–217, 1998.

[32] W. Smith, I. Foster, and V. Taylor, "Predicting application run times with historical information," *J. Parallel Distrib. Comput.*, vol. 64, no. 9, pp. 1007–1016, Sep. 2004. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2004.06.008

[33] D. Zotkin and P. J. Keleher, "Job-length estimation and performance in backfilling schedulers," in *High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on*, 1999, pp. 236–243.

[34] D. Tsafrir and D. G. Feitelson, "The dynamics of backfilling: Solving the mystery of why increased inaccuracy may help," in *2006 IEEE International Symposium on Workload Characterization*, Oct 2006, pp. 131–141.