Design and Potential Performance of Goal-Oriented Job Scheduling Policies for Parallel Computer Workloads

Su-Hui Chiang and Sangsuree Vasupongayya

Abstract—To balance multiple scheduling performance requirements on parallel computer systems, traditional job schedulers are configured with many parameters for defining job or queue priorities. Using many parameters seems flexible, but in reality, tuning their values is highly challenging. To simplify resource management, we propose goal-oriented policies, which allow system administrators to specify high-level performance goals rather than tuning low-level scheduling parameters. We study the design of goal-oriented policies, including 1) multiobjective models for specifying trade-offs between objectives; 2) efficient search algorithms for searching the best schedule; and 3) performance measures for optimization in the objectives with respect to two common performance requirements: preventing starvation and favoring shorter jobs. We compare goal-oriented policies with widely used backfill policies. Policies are evaluated by simulation using real job traces from several computing systems. Our results show that by automatically optimizing performance according to the given objectives through search, goal-oriented policies can improve the performance of backfill policies.

Index Terms—Parallel job scheduling, multiobjective models, goal-oriented policies, backfilling, search algorithms.

1 Introduction

On general-purpose parallel computer systems, there are typically multiple conflicting scheduling performance goals. Current production job schedulers have limitations in dealing with such multiobjective problems. To address the problem, we propose goal-oriented scheduling policies. We study several design and implementation questions and evaluate potential performance of the policies.

Most production job schedulers use a variety of nonpreemptive policies, based on queue or job priorities. Using *queue-based* priority (e.g., PBS [1] and LSF [2]), jobs in the highest priority queue are scheduled first. As a simple example, to favor short jobs, we can assign the highest priority to short-job queues; at the same time, we may need to place a limit on the number of short jobs that can be simultaneously executed, to prevent starving other queues. Using *job-based* priority (e.g., Maui Scheduler [3]), jobs are prioritized using a weighted sum of job measures. For example, to favor short jobs, we can assign a large priority weight to the job *expansion factor* (i.e., current slowdown), which is the sum of job runtime and job current waiting time divided by the job runtime. To reduce starvation, we can use some weight for the job current waiting time, to boost the priority of jobs as they wait; we might also assign some weight to the job requested resources since large-resource jobs often incur a long wait under heavy loads. Configuring these parameters is highly challenging. Furthermore, their values need to be continually tuned to adapt to the workload changes.

To simplify the administrative task, we propose goal-oriented policies, which allow decision-makers to specify high-level performance goals rather than tuning low-level scheduling parameters. High-level goals can be as abstract as "favoring shorter jobs," in which case the scheduler determines what measures will be optimized. Alternatively, the administrators can explicitly specify the objectives, such as "minimizing the average slowdown." In either case, the schedulers, and not the administrators, figure out how to prioritize the jobs.

To design goal-oriented policies in the presence of conflicting objectives, several questions need to be studied. First, since no single solution optimizes all objectives, how should we select *the* schedule from all possible schedules at each scheduling decision point? The concept of multiobjective problems is certainly not new. In the parallel and grid resource management field, the common approach is to aggregate objectives into a scalarizing function, thus converting a multiobjective problem to a single-objective problem (e.g., [4] and [5]). The problem is that determining appropriate weights for the scalarizing function is difficult. In the case it is computationally feasible and human interaction is possible, one can find a set of optimal solutions in the sense of Pareto optimality and let the decision-makers select the final solution. Note that a Pareto optimal solution is one that is better than any other possible solution for at least one objective. However, for the problem studied here, finding even one optimal solution is not computationally feasible. In addition, it is not practical to require feedback from system administrators at each

• *S.-H. Chiang is with the Department of Computer Science, Portland State University, PO Box 751-CMPS, Portland, OR 97207-0751. E-mail: suhui@cs.pdx.edu.*
• *S. Vasupongayya is with the Department of Computer Engineering, Prince of Songkla University, Room R400, Robot Building, Hadyai, Songkla 90112, Thailand. E-mail: vsangsur@eng.psu.ac.th.*

Manuscript received 6 Sept. 2007; revised 8 Feb. 2008; accepted 25 Feb. 2008; published online 14 Mar. 2008.
Recommended for acceptance by D. Trystram.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2007-09-0307.
Digital Object Identifier no. 10.1109/TPDS.2008.48.

TABLE 1
Notation

	Symbol	Definition
Per-job metric	N	Number of job requested nodes
	T	Actual job runtime
	X	Job slowdown = job turnaround time / T
Other metrics	ρ	Offered load in simulation
	L	Max. # of visits to tree nodes during search

scheduling decision point, since the scheduling events are frequent. In this study, we will investigate and design multiobjective models that make trade-off among objectives in an intuitive manner and require no manual configuration for low-level parameters.

The second question concerns the efficiency of searching the schedule at each scheduling decision point. Since the number of possible schedules grows faster than exponentially with the problem size (to be discussed in Section 2.4), it is not practical to evaluate all schedules. The question is in what order schedules should be evaluated so that a *good enough* schedule can be found in a reasonable amount of time.

Another question concerns the measures to be optimized in the objectives. The problem arises because for a given high-level performance goal, such as preventing starvation, more than one measure may be used for optimization. To define appropriate objectives, we will study the performance impact of optimizing alternative measures.

In this paper, we consider two performance goals, commonly placed on general-purpose parallel computer systems. They are preventing starvation and favoring shorter jobs. We investigate the performance impact of alternative objective models and measures for optimization. Our goal-oriented policies are compared with backfill policies, which are used on many production parallel systems because of their high performance. Policies are evaluated using event-driven simulation of real job traces from three parallel computer systems. Results are reported for the 10 monthly job traces from a 128-node Intel Itanium (IA-64) Linux cluster at NCSA. Results for two IBM-SP2 traces from SDSC and KTH [6] are qualitatively similar and omitted to conserve space. Part of our results were reported in two previous papers [7], [8].

The main contributions of this study include the following:

- To deal with conflicting scheduling objectives in parallel computer systems, we propose goal-oriented job scheduling policies, which not only greatly simplify the administrative task but also have the potential to significantly improve the performance of traditional priority policies.
- We study the impact of using alternative objective models and optimizing alternative measures, with respect to two common scheduling performance goals: preventing starvation and favoring shorter jobs.
- We design fairly intuitive Eq-Tradeoff objective model, which can be used to simultaneously optimize multiple objectives.

TABLE 2
Capacity and Job Limit on NCSA/IA-64

Capacity (#Nodes)	Max. Job Size		Period
	N	T	
128	128	12h	6/03 – 11/03
		24h	12/03 – 3/04

- To prevent starvation, we propose to minimize the total excessive wait, which may be more appropriate than minimizing the maximum wait time, especially in the presence of other objectives that optimize average performance measures.

Section 2 provides background information for this study. Section 3 defines objective models and measures to be optimized. Section 4 discusses our evaluation method. Section 5 studies several policy design issues. Section 6 evaluates performance of goal-oriented policies using alternative objectives. Section 7 summarizes our results.

2 Background

In this section, we provide information of the workloads studied and briefly review relevant work, including previous parallel job scheduling policies, representative approaches for dealing with multiobjective problems, and search algorithms for exploring the solution space. For convenience, Table 1 defines some of the notations used.

2.1 Workloads

In this section, we provide some information of the NCSA/IA-64 monthly workloads during June 2003-March 2004, used for evaluating policies.

Table 2 summarizes the system capacity and job limits. The system contains 128 dual-processor nodes; the smallest allocation unit is one node. The job runtime limit was 12 hours in the first six months and increased to 24 hours in December 2003.

Table 3 summarizes the workload characteristics of each month, including the number of submitted jobs, processor demand (defined as $N \times T$ of the submitted jobs, expressed as a fraction of the total nodes-time available on the system in a month), average job size (N, T, and $N \times T$), and some information of the distributions of N and T. The monthly processor demand is typically 70-80 percent, except in July 2003 where it is close to 90 percent. Typically in each month, 3,000-4,000 jobs were submitted, 80-90 percent of jobs requested no more than eight nodes, and 80-90 percent of jobs had a runtime of under 5 hours. Other than that, workload characteristics vary widely from month to month. For example, the average job runtime ranges from 1 to 4.5 hours, and average nodes ranges from 5 to 23.5.

Two monthly workloads stand out: 1) in July 2003, 18 percent of the jobs requested over 32 nodes, compared to only under 5 percent in most other months, and 2) in January 2004, over 30 percent of the jobs had a runtime of at least 5 hours, compared to 15 percent in most other months. In addition, January 2004 has a larger average job requested nodes than in most of the other months (except June and July 2003), due to an almost 30 percent of jobs in

TABLE 3
Summary of Monthly Load and Workload Characteristics on NCSA/IA-64

Month	# Jobs	Proc. demand	Avg. job size			Fraction of jobs in each class					
			N	T	N×T	$N \leq 8$	$9 \leq N \leq 32$	$32 < N$	$T < 5h$	$5h \leq T < 10h$	$10h \leq T$
Jun 03	2191	82%	12.1	1.4h	34.5h	74.2%	19.0%	6.8%	90.3%	4.0%	5.7%
Jul 03	1399	89%	23.5	1.9h	60.7h	60.7%	21.1%	18.2%	84.8%	7.8%	7.4%
Aug 03	3220	79%	7.3	1.1h	23.5h	86.3%	9.5%	4.2%	91.0%	3.7%	5.3%
Sep 03	3056	72%	9.1	1.4h	21.7h	80.7%	15.0%	4.4%	90.3%	4.6%	5.0%
Oct 03	4149	71%	5.0	2.0h	16.3h	88.8%	9.2%	2.0%	86.6%	7.2%	6.3%
Nov 03	3446	73%	6.0	2.5h	19.5h	87.8%	7.7%	4.5%	82.0%	10.7%	7.3%
Dec 03	3517	74%	5.6	3.6h	20.2h	86.9%	9.6%	3.6%	76.2%	12.7%	11.1%
Jan 04	3154	73%	10.7	4.5h	22.1h	69.8%	27.3%	2.9%	67.2%	19.9%	12.9%
Feb 04	3969	74%	5.0	3.1h	16.6h	90.4%	7.1%	2.4%	84.2%	4.6%	11.2%
Mar 04	3468	75%	5.8	2.4h	20.6h	88.5%	8.2%	3.4%	88.0%	2.1%	9.9%

January 2004 requesting a medium size, i.e., 9-32 nodes. The distinct features of July 2003 and January 2004 present a great challenge for scheduling policies. As can be seen, the 10 monthly workloads provide a variety of workloads for evaluating scheduling policies.

2.2 Priority Backfill Scheduling Policies

The first implementation of first-come-first-served (FCFS)-backfill [9] was reported for an IBM SP1 in 1995. Since then, backfill policies have been extensively studied, and the backfill feature has been implemented in many widely used production schedulers, including the Maui Scheduler [3], LSF [2], PBS [1], and LoadLeveler [10]. Because of the high performance and popularity of backfill policies, we use them as baseline policies for evaluating our goal-oriented policies.

Under the original FCFS-backfill, jobs are considered for scheduling in their arriving order, and the first waiting job that cannot be started due to insufficient free resources is given a reservation; jobs in the back of the queue can be *backfilled* on idle resources as long as they will not delay the reservation.

Many papers proposed priority functions to improve FCFS-backfill (e.g., [11], [12], [13], and [14]), and some papers studied the performance impact of giving more than one reservation [15], [16]. The key results are summarized below. First, there is a trade-off between improving the maximum job wait time and improving average-performance measures. In particular, largest slowdown first (LXF)-backfill, which gives priority to the job with the largest expansion factor, significantly improves the average slowdown and average wait but gives a worse maximum wait, compared with FCFS-backfill. In the extreme case, SJF-backfill, which gives priority to the shortest job, may starve long jobs and, thus, is not a practical policy. Second, a few reservations can improve the maximum wait of one reservation without significantly degrading the average performance measures, but too many reservations can degrade the average performance without further improving the maximum wait.

In addition, many other variations of backfill policies have been proposed. Two papers that proposed adaptive policies [17], [18] are most relevant to our work. Under these adaptive policies, different backfill policies may be used in different periods of time. To choose the best policy for the next period of time, the schedulers conduct an online simulation of particular backfill policies, using workloads that recently ran on the system. These two approaches differ in when and how online simulations are performed and what backfill policies are simulated. Although the idea of choosing the best policy is somewhat similar to choosing the best schedule in our goal-oriented policies, their policies are fundamentally different from ours. First, they are concerned with only a single performance measure, while we are concerned with multiple objectives. Second, they can only choose a policy from a set of particular policies, while we do not have such a limitation.

2.3 Dealing with Conflicting Objectives

As discussed in Section 1, finding the set of Pareto optimal solutions is not practical for the problem studied. Thus, we focus on methods that convert multiobjective optimization problems into some forms such that a single solution can be obtained. Three representative methods are reviewed below. For a more comprehensive survey of multiobjective combinatorial optimization, we refer to [19].

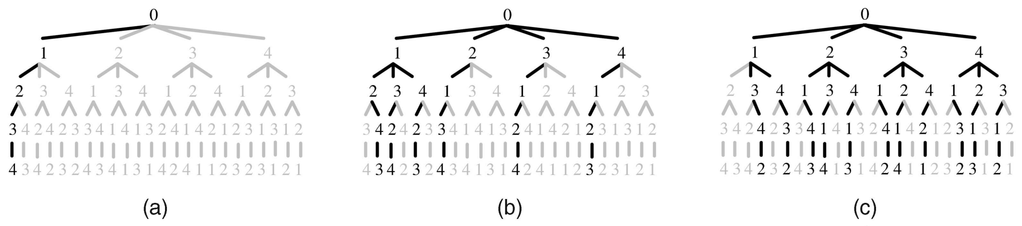
A *weighted objective* combines multiple objectives into a weighted sum of measures. While simple, choosing the weights can be difficult. Thus, we do not consider this approach. However, interestingly, one of the objective models (C-Tradeoff) studied in this paper turns out to use a weighted objective with the weights dynamically and automatically determined at each scheduling decision point.

In *lexicographical ordering*, objectives are ranked by their importance and optimized one by one according to their importance. Although we consider equally important objectives, this simple approach may still be applied if appropriate measures are optimized. We adopt this approach to define the Lexical model, which is discussed in Section 3.2.

In *goal programming*, a target value is specified for each objective; the aim is to minimize the deviation from specified goals. We do not consider this approach because it is difficult to determine the target values and compute deviation.

2.4 Search Algorithms

At each scheduling decision point, the scheduler evaluates possible schedules to search for the final schedule, according to the given objectives. Since each permutation of waiting jobs is an eligible schedule, the number of possible schedules grows faster than exponentially with the number of waiting jobs. For example, as shown in Fig. 1d, with only



# Wait jobs	# Paths	# Nodes
4	24	64
8	40K	110K
10	3,629K	9,864K
15	1,307,674M	3,554,627M

(Note: 'K' = 1000; 'M' = 1000,000)

(d)

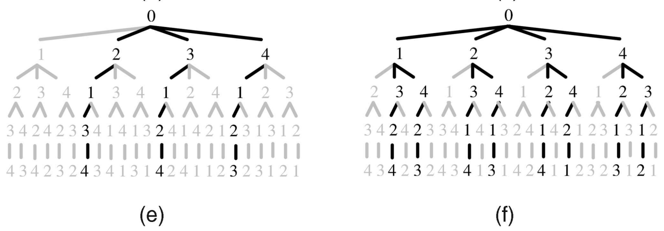


Fig. 1. Illustration of the LDS and DDS algorithms using a sample search tree of four jobs. (a) Zeroth iteration. (b) LDS: first iteration. (c) LDS: second iteration. (d) Tree size as a function of waiting jobs. (e) DDS: first iteration. (f) DDS: second iteration.

eight waiting jobs, there are over 40,000 possible schedules; with 10 waiting jobs, it is over 3.5 million. As evaluating all schedules is not computationally practical, the best we can do is to find a "good enough" solution; the key to the success of search algorithms is to explore promising solutions as soon as possible.

Roughly speaking, there are two classes of search algorithms: systematic search (or the so-called tree search) and local search. In this study, we adopt systematic search algorithms for their simplicity. Our future work will consider combining systematic search with local search, as suggested in [20], to possibly improve the search efficiency. Below, we discuss how schedules are evaluated using systematic search algorithms.

Each *schedule* defines the order in which the currently waiting jobs are *considered* for scheduling. When a job of a given schedule is considered, the earliest start time of the job is determined, and the resources are *tentatively* allocated for the duration of the job runtime. The earliest start time of the job is computed such that the job will not take away resources from any currently executing jobs during their execution, nor can it delay the start time tentatively scheduled for any waiting job (considered before this job in the same schedule). After the last job of a schedule is considered, the performance measures of the schedule are computed. If they are better than that of the best schedule so far, this schedule replaces the best schedule.

The role of the search algorithm is to determine the order in which candidate schedules are evaluated. Under systematic search algorithms, the candidate schedules are organized into a tree according to some *branching heuristic*, and the tree is *traversed* in a particular order, independent of the branching heuristic. The search algorithm typically refers to the tree traversing algorithm, but we also use the term to refer to the combination of the tree traversing and branching heuristic when there is no confusion.

In this study, we consider two commonly used discrepancy-based systematic search algorithms for traversing the tree: limited discrepancy search (LDS) [21], [22], and depth-bound discrepancy search (DDS) [23]. To illustrate the structure of the search tree and search algorithms, we use an example of four waiting jobs, numbered 1 to 4, in their arriving order. Clearly, there are 24 permutations of the jobs and, thus, 24 schedules. Note that different schedules may result in the same solution, i.e., they assign the same start time to each waiting job, but this is not known until they are evaluated.

Fig. 1a shows an example of organizing the 24 schedules into a tree. The root is a dummy node, labeled "0"; every other node is labeled by the job number. Each *path*, from the root to a leaf, constitutes a schedule. Thus, the number of leaves is equal to the number of possible schedules. For example, the leftmost path, 0-1-2-3-4, shown in bold in Fig. 1a, is the schedule in which jobs are considered in the arriving order.

The root is said to have a *depth* of zero, each child of the root has a depth of one, and each child of a depth-one node has a depth of two, and so forth. At the root, there are four branches, representing the four choices of the first job in the schedule. Each depth-one node has three branches, representing the three choices of the second job in the schedule. In general, in a tree of n jobs, each $\text{depth-}i$ node has $n-i$ branches, and the tree contains $n!$ paths and $O(n!)$ nodes. Fig. 1d shows several sample tree sizes to give an idea of how fast the tree size grows with the number of jobs. To be realistic, only the path being evaluated, and not the entire tree, is stored.

This sample tree uses the FCFS branching heuristic, in that the branches, from left to right, at each node are in the job arriving order. At any $\text{depth-}i$ node, only the leftmost branch follows the heuristic; any other branch is called a *discrepancy* at depth $i+1$, by convention. For example, the leftmost path, 0-1-2-3-4, is the only path that contains no discrepancy; the path 0-2-1-3-4 contains exactly one discrepancy at depth one (i.e., job 2).

The assumption of discrepancy-based search algorithms is that a good branching heuristic is likely to make only a few mistakes. Thus, to find promising solutions soon, the idea is to visit paths with fewer discrepancies first. Both LDS and DDS proceed in iterations. In each iteration, one or more paths (i.e., schedules) are evaluated. As many paths are compared as the time permits. For the four-job example, Fig. 1 shows the paths in bold, evaluated in the first three iterations (zeroth, first, and second) by LDS and DDS. The paths are visited in the order from left to right as they appear in the tree, which is further explained below.

LDS visits the paths that contain the fewest discrepancies first. On the zeroth iteration, LDS always branches left with the branching heuristic, i.e., 0-1-2-3-4 highlighted in Fig. 1a. On the first iteration, LDS visits the six paths containing exactly *one* discrepancy at any depth, highlighted in Fig. 1b. Similarly, on the second iteration, LDS visits the 11 paths containing exactly *two* discrepancies, highlighted in Fig. 1c.

DDS biases the search to the discrepancies high in the tree, by iteratively increasing the depth at which discrepancies occur. The motivation is if there are mistakes in the branching heuristic, the mistakes are more likely to occur at the top of the tree than at the bottom. Specifically, on the ith iteration, DDS visits each path containing a discrepancy at depth i, any number of discrepancies above depth i, and no discrepancy below depth i. On the zeroth iteration, DDS visits the only path containing no discrepancy, same as in LDS. On the first iteration, shown in Fig. 1e, DDS visits three paths, each containing a discrepancy at *depth one* and no discrepancy below. On the second iteration, shown in Fig. 1f, DDS visits eight paths containing zero or one discrepancy at depth one and one discrepancy at *depth two*, but no discrepancy below depth two.

By biasing search to discrepancies high in the tree, DDS explores early in time the schedules that differ in their first few jobs, which have the greatest effect on the start times of all waiting jobs. For example, as shown in Fig. 1, the first four schedules evaluated under DDS have different first job, while that under LDS have the same first job (i.e., job 1). Thus, if the branching heuristic is not good enough, it is likely that DDS explores promising solutions sooner than LDS does for the problem studied.

To reduce the search space, we use a simple pruning technique for both LDS and DDS, when applicable. The idea is to abandon a subtree when the subtree does not contain better solutions than that of the best schedule found so far. For example, suppose the objective is to minimize the maximum wait time. Then, as soon as we find that the wait time of the job at a nonleaf node, say x, of the path under evaluation is already worse than that of the best schedule found so far, we abandon the subtree rooted at x. Pruning requires comparing the partial solution at each node with that of the best schedule found so far. Nevertheless, the overhead is minimal compared with other computation overhead of evaluating each node.

3 Measures, Objective Models, and Policies

In this paper, we consider two goals: preventing starvation and favoring shorter jobs. Although they are common requirements, they are not specific objectives. Below, we will first discuss the measures used for optimization in the objectives, with respect to the two goals studied. We then define the multiobjective models. Using these measures and objective models, we define a set of scheduling policies to be studied.

TABLE 4
Performance Measures for Optimization

Type of Measures	Notation	Meaning
Starvation	T_w	Total excessive wait (using the dynamic threshold)
	$maxW$	Maximum job wait time
Average performance	$avgW$	Average job wait time
	$avgX$	Average job slowdown

3.1 Measures for Optimization

To prevent starvation, a simple but perhaps too strict an objective is to minimize the maximum wait time ($maxW$). As an alternative, we define a new measure, called total excessive wait (T_w), to be discussed later. To favor shorter jobs, we consider minimizing either the average slowdown ($avgX$) or average wait ($avgW$), which are commonly used for evaluating scheduling performance in previous papers. Table 4 summarizes the measures and notation. Note that as mentioned in Section 1, the measures to be optimized can be either defined for each goal within the schedulers, or directly specified by system administrators.

The *excessive wait time* of a job is defined to be the wait time of the job in excess of a given threshold, ω, which can be thought of as a target upper bound on job wait time. Note that if a job has waited no more than ω, the job does not have an excessive wait. The *total excessive wait* is the sum of the excessive wait times of all jobs. To compute T_w at a given point of time, the scheduler considers only the jobs that are still waiting at that time and computes their *current* excessive waits. The threshold ω can be *fixed* (e.g., 50 hours), which is simple but may not perform well for all load conditions. To automatically adapt to the load condition, we consider a *dynamic* threshold, defined to be the current waiting time of the oldest job in the waiting queue. By default, T_w uses the dynamic threshold. For clarity, we use T_w^ω to denote the total excessive wait using a fixed threshold.

3.2 Multiobjective Models

Two objective models are studied: Lexical and Eq-Tradeoff. The Lexical model uses the lexicographical ordering approach, in which objectives are ranked in their importance, as discussed in Section 2.3. To more naturally model equally important objectives, we design the Eq-Tradeoff model. Three variations of Eq-Tradeoff will be studied in this paper. Below, we define these objective models.

An objective model with the objectives define a binary relation, $\prec$, over the set of candidate schedules as follows (informally): $\Phi \prec \Gamma$ if the schedule Φ is considered better than Γ, for any schedule Φ, Γ. If $\prec$ is a total order,[1] then the best schedule is uniquely defined (provided that all schedules have been evaluated), i.e., the best schedule is independent of the order in which the schedules are evaluated. As it can be seen later, the relations studied here are not total order because they lack the totality property, i.e., not every pair of schedules can be compared

1. A relation $\prec$ is a total order over the set Π, if for any $a \neq b \neq c \in \Pi$, it has the following three properties: 1) asymmetric: $a \prec b \Longrightarrow b \not\prec a$; 2) transitive: $a \prec b$ and $b \prec c \Longrightarrow a \prec c$; and 3) totality: $a \prec b$ or $b \prec a$.

TABLE 5
Definition of Objective Models that Minimize Two Measures x and y

Model	Notation	Condition for which $\Phi \prec \Gamma$ (i.e., Φ is better than Γ)
Lexical	Lexical($x \rightarrow y$)	(1) $\triangle_x(\Gamma, \Phi) > 0$, or (2) $\triangle_x(\Gamma, \Phi) = 0$ and $\triangle_y(\Gamma, \Phi) > 0$
Equal-Tradeoff	Tradeoff(x:y)	$\frac{\triangle_x(\Gamma,\Phi)}{\hat{x}} + \frac{\triangle_y(\Gamma,\Phi)}{\hat{y}} > 0$

Notation and definition: for any schedules Γ and Φ and any measure x,
(1) Γ_x: denotes the value of x of the schedule Γ.
(2) $\triangle_x(\Gamma, \Phi) = \Gamma_x - \Phi_x$: denotes the difference between Γ and Φ on the measure x.

For Equal-Tradeoff model, three versions of $\hat{x}$ and $\hat{y}$, with respect to the schedules Γ and Φ, are defined:
Version A: $\hat{x} = \Gamma_x$; $\hat{y} = \Gamma_y$;
Version B: $\hat{x} = \text{Minimum}\{\Gamma_x, \Phi_x\}$; $\hat{y} = \text{Minimum}\{\Gamma_y, \Phi_y\}$;
Version C: $\hat{x}$ and $\hat{y}$ are *approximately* lower bounds of x and y, respectively, at the time when Γ and Φ are compared ($\hat{x}$ and $\hat{y}$ are derived based on backfill policies, explained in the text)

(for they have the same quality according to the given model). However, a relation that has the asymmetric and transitive properties still guarantees that the set of best schedules is uniquely defined, which will be referred to as the *uniqueness* property, for convenience. Among the best schedules, the one evaluated first will be chosen, as it more closely follows the search heuristics. Having the uniqueness property is ideal but perhaps not critical, since typically not all schedules can be evaluated in time. We will comment on the property of each model later.

For the purpose of explaining the models, we assume that our objectives are to *minimize* x and y. Our future work will extend the models to deal with more than two measures. Further assume that Γ is the best schedule found so far in the given scheduling epoch, and Φ is the next schedule to be evaluated. Table 5 summarizes the condition under which $\Phi \prec \Gamma$ for each model, which will be further discussed below. For convenience, in Table 5, we also define two variables: Γ_x and $\Delta_x(\Gamma, \Phi)$, for any measure x and any schedule Γ and Φ. Note that if $\Delta_x(\Gamma, \Phi) > 0$, it is the amount by which Φ_x *improves* Γ_x; if $\Delta_x(\Gamma, \Phi) < 0$, then Φ_x *degrades* Γ_x by $-\Delta_x(\Gamma, \Phi)$. The difference as defined is *strict* in that two values are considered different, albeit a tiny difference. Later, in Section 3.3, we will loosen the definition to tolerate marginal differences.

Lexical model. Denoted by $\text{Lexical}(x \rightarrow y)$: In this model, x dominates y. That is, among the candidate schedules evaluated, the one with the best x value is chosen; the y value is only used to break a tie, if needed. This model is simple and has the uniqueness property. Although designed for the cases where the objectives can be ranked, it may be used to model roughly equally important goals, if the dominating measure x is chosen such that there is some room for optimizing y. Note that this is the model studied in our previous paper [7] and was called the hierarchical model.

Eq-Tradeoff model. Denoted by $\text{Tradeoff}(x : y)$: In this model, x and y are equally important. Thus, $\text{Tradeoff}(x : y)$ is equivalent to $\text{Tradeoff}(y : x)$. Under this model, for $\Phi \prec \Gamma$, one of two conditions has to hold: 1) Φ improves at least one measure and does not degrade the other measure, or 2) Φ improves one measure and degrades the other measure, but the improvement is more than the degradation, computed as the *ratios* of the actual improvement and degradation to some *reference values* of the corresponding measures. The above two conditions can be combined into a single condition: $\Delta_x(\Gamma, \Phi)/\hat{x} + \Delta_y(\Gamma, \Phi)/\hat{y} > 0$, where $\hat{x}$ and $\hat{y}$ are the reference values of x and y, respectively. To see this, let us assume, without loss of generality, that Φ_x improves Γ_x. Then, for $\Phi \prec \Gamma$, either Condition 1 is true, which means $\Delta_x(\Gamma, \Phi) > 0$ and $\Delta_y(\Gamma, \Phi) \geq 0$; or Condition 2 is true, which means $\Delta_x(\Gamma, \Phi)/\hat{x} > -\Delta_y(\Gamma, \Phi)/\hat{y}$. In either case, we have $\Delta_x(\Gamma, \Phi)/\hat{x} + \Delta_y(\Gamma, \Phi)/\hat{y} > 0$.

Three versions of reference values for the Eq-Tradeoff model are studied. Note that the version of reference values is not intended to be a configurable parameter, rather we will compare different versions and use the best one to define the model. For convenience, the binary relations defined by the three versions A, B, and C are denoted by $\prec_A$, $\prec_B$, and $\prec_C$, respectively. The relations $\prec_B$ and $\prec_C$ have the uniqueness property, but $\prec_A$ does not, which will be discussed below (without proof). Note that version A was used in our previous paper [8].

In *version A*, the reference values are defined by the best schedule found so far, say Γ. That is, $\hat{x} = \Gamma_x$ and $\hat{y} = \Gamma_y$. As an example, let $\Gamma_x = 100$ and $\Phi_x = 20$, then Φ_x improves Γ_x by $(100 - 20)/100$, i.e., 80 percent. Conversely, if $\Gamma_x = 20$ and $\Phi_x = 100$, then Φ_x degrades Γ_x by $-(20 - 100)/20$, i.e., 400 percent. Clearly, an improvement to any measure is always less than 100 percent in version A. Thus, a degradation of over 100 percent to any measure of Γ will not be accepted to replace Γ in any case. Version A is intuitive in that the changes to Γ are computed against the quality of Γ. Unfortunately, since the reference values used for comparing two schedules depend on which schedule is evaluated first, $\prec_A$ lacks the asymmetric property. Furthermore, it also lacks the transitive property. That is, $\Phi_1 \prec_A \Phi_2, \Phi_2 \prec_A \Phi_3, \ldots,$ and $\Phi_{n-1} \prec_A \Phi_n$ do not imply $\Phi_1 \prec_A \Phi_n$. Nevertheless, under the above conditions, we still have $\Phi_1 \prec_B \Phi_n$, which will become clear later.

In *version B*, the reference value of measure x is the smaller of Φ_x and Γ_x. We design this version to have the uniqueness property. We note that a similar idea was used as the acceptance criteria in [24] in a different context. Using the above example again, if $\Gamma_x = 100$ and $\Phi_x = 20$, then the ratio improvement made by Φ_x is 400 percent, rather than 80 percent as computed in version A. On the other hand, if $\Gamma_x = 20$ and $\Phi_x = 100$, then Φ_x degrades Γ_x by 400 percent, same as in version A. In general, in the case of an improvement, the ratio improvement computed by version A is smaller than that by version B; in the case of a degradation, both versions compute the same degradation.

TABLE 6
Goal-Oriented Scheduling Policies

Goal	Policy	Goal	Policy

Minimizing T_w and $avgX$	Lexical($T_w \rightarrow avgX$)
	Lexical($avgX \rightarrow T_w$)
	Tradeoff(T_w:$avgX$)
		Minimizing T_w and $avgW$	Lexical($T_w \rightarrow avgW$)
			Lexical($avgW \rightarrow T_w$)
			Tradeoff(T_w:$avgW$)

Minimizing $maxW$ and $avgX$	Lexical($maxW \rightarrow avgX$)
	Lexical($avgX \rightarrow maxW$)
	Tradeoff($maxW$:$avgX$)
		Minimizing $maxW$ and $avgW$	Lexical($maxW \rightarrow avgW$)
			Lexical($avgW \rightarrow maxW$)
			Tradeoff($maxW$:$avgW$)

Note: For each Tradeoff policy, a prefix A-, B-, or C- will be used to indicate the version of reference values used
Note: For a policy that uses the strict difference, "Strict" will be shown as part of the policy name.

Thus, $\Phi \prec_A \Gamma \Longrightarrow \Phi \prec_B \Gamma$, but the reverse is not true. In addition, it can be shown that version B is the same as minimizing $x \times y$. As can be seen, version B is more aggressive than version A in replacing the best schedule found so far, possibly by a schedule that substantially improves only one measure to reduce the product. Thus, version B could potentially oscillate between schedules that favor different measures.

In *version* C, the reference value of each measure is an approximately lower bound of the measure, which is dynamically computed at the beginning of each scheduling epoch. It is designed to provide possibly better reference value of each measure than that of version B, while still keeping the uniqueness property. Interestingly, it turns out that version C is equivalent to minimizing a weighted sum of the measures to be minimized, where the weight for each measure x is $1/\hat{x}$, i.e., the inverse of the *current* reference value. To avoid a high overhead of computing the reference values, we do not actually search for the schedule that optimizes each measure. Instead, we compute the reference value of each measure using a priority backfill policy that may favor the measure. Specifically, we use FCFS-backfill for computing the reference values for $maxW$ and T_w, and LXF-backfill for $avgX$ and $avgW$. We comment that SJF-backfill may achieve better $avgX$ and $avgW$ than that of LXF-backfill, but they are impractical as references due to the potential starvation problem.

3.3 Relaxed Differences

When a schedule Φ improves another schedule Γ for all measures to be optimized, Φ is clearly a better schedule. However, when Φ improves some measures by only a margin and is worse for other measures, it is arguable whether Φ is better. The idea of using *relaxed differences* is to tolerate marginal differences, i.e., a sufficiently small difference is considered no difference. Specifically, for a measure x, the relaxed version of $\Delta_x(\Phi, \Gamma)$ is defined as follows:

$$\Delta_x(\Gamma, \Phi) = \begin{cases} 0, & if \ -\epsilon_x < \Gamma_x - \Phi_x < \epsilon_x, \\ \Gamma_x - \Phi_x, & \text{otherwise,} \end{cases}$$

where ϵ_x is a *very small* positive value, which will be discussed later. The above definition is used only when Φ improves one measure but degrades the other. Otherwise, strict differences are used.

The question is how small is sufficiently small. To take into account the magnitude of the values compared, ϵ_x is defined as follows: 1) for $x = maxW$, $avgW$, or $avgX$, we set ϵ_x to be 1 percent of the x value of the best schedule found so far; 2) for $x = T_w$, we set ϵ_x to be 1 percent of the current maximum wait time of the jobs still waiting in the queue (i.e., the dynamic wait time threshold used for computing T_w).

The choice of 1 percent is somewhat arbitrary but should be reasonable. Take $x = maxW$ for an example: if the best schedule found so far has an expected maximum wait (i.e., $maxW$) of 100 hours, then ϵ_x is 1 hour. Thus, if the expected maximum wait of another schedule is 99-101 hours, it is considered the same as that of the best schedule, if the relaxed difference is used. As for $x = T_w$, we do not use the T_w value of the best schedule to compute ϵ_x, because T_w may be zero, which, if used to compute ϵ_x, would prevent any schedule with a nonzero T_w (albeit tiny) from being considered better.

Same as the versions of Eq-Tradeoff, we do not intend to make the definition of differences a configurable parameter. Instead, we will choose the better definition to define objective models. Finally, we point out that as a result of using relaxed differences, all objective models studied lose the uniqueness property. Nevertheless, tolerating some marginal differences should be more reasonable than using strict differences.

3.4 Goal-Oriented Policies Based on Search

Using the objective models and the measures discussed earlier, Table 6 defines a set of goal-oriented policies. Below, we comment on the policy naming and implementation.

For each policy using the Eq-Tradeoff model, there are three versions of the policy, corresponding to the three versions of the reference values. We will use a prefix "A-," "B-," or "C-" to indicate which version of reference values is used, e.g., A-Tradeoff($T_w : avgX$). In addition, by default, our goal-oriented policies use relaxed differences allowing 1 percent slack; policies that use strict differences contain the label "Strict" in their names, e.g., Strict Lexical ($T_w \rightarrow avgX$). Note that policies using T_w^ω, i.e., total excessive wait using a fixed threshold, are only discussed in Section 5.1 and omitted from Table 6.

Our policies use systematic search algorithms to find a good enough schedule at each scheduling decision point. As discussed in Section 2.4, a systematic search consists of a tree traversing heuristic and a branching heuristic. We study two traversing algorithms: LDS and DDS, and two branching heuristics: FCFS and LXF. Thus, there is a combination of four search algorithms. Note that the FCFS heuristic favors the goal of "preventing starvation," whereas the LXF heuristic favors shorter jobs. For convenience, we denote each algorithm by concatenating the heuristics used, e.g., DDS-LXF.

For comparison, we impose a limit on L, the number of tree nodes visited at each scheduling decision point. Note that any nonleaf nodes may be visited more than once (because each appears in more than one path), and any node visit is counted. Section 5.2 will compare search algorithms and the impact of L on the performance, and provide some information of the scheduling overhead.

4 Evaluation Methodology

We evaluate scheduling policies using event-driven simulation of job traces, as discussed in Section 2.1. Goal-oriented policies are compared against FCFS-backfill and LXF-backfill, which favor the maximum wait and the average performance measures (average slowdown and wait), respectively. In our simulation, backfill policies give reservation to the highest priority waiting job only, since we do not find giving more than one reservation beneficial for the workloads studied.

An extensive set of measures is used for performance evaluation, including the average and maximum of wait time and bounded slowdown, as well as total *normalized* excessive wait, to be defined later. Same as in many previous papers, we use the *bounded* slowdown, rather than the actual slowdown, to reduce the dramatic effect of very short jobs on the average slowdown measure. We use 1 minute to lower bound the actual job runtime, i.e., we compute the slowdown of a job of under 1 minute as if it was a 1-minute job.

The *normalized excessive wait time* measures provide some information of jobs that incur a relatively long wait time under each policy. For each month and the load level simulated, two thresholds are used to compute these normalized measures: the maximum wait and the ninety-eighth percentile wait time under FCFS-backfill for that month and load level. The same thresholds are used for all policies so that different policies can be compared. The two per-job normalized excessive wait measures are denoted by $E^{maxW}_{\text{FCFS-bf}}$ and $E^{98\%W}_{\text{FCFS-bf}}$, respectively. The total normalized excessive wait is the sum of the normalized excessive wait of all jobs. Note that by definition, FCFS-backfill has a zero total $E^{maxW}_{\text{FCFS-bf}}$ in every month.

The performance measures of each month are computed for jobs submitted during the month. However, to be realistic, the simulation of each month includes a 1-week (from previous month) warmup and a cooldown period in which jobs (from next month) continue to arrive until all jobs submitted during the month analyzed have started. The cooldown period is typically a few days only.

Two levels of loads are simulated: 1) $\rho = \text{original load}$ and 2) $\rho = 0.9$. Recall that most IA-64 monthly processor demand studied is 70 percent-80 percent, except for July 2003 where it is close to 90 percent. The workloads of $\rho = 0.9$ are artificially created by shrinking job interarrival times, as in previous papers (e.g., [14] and [16]), for lack of a better model. We focus on the $\rho = 0.9$ results, since the performance difference is larger for high load.

Finally, in this paper, we assume that job runtime information is known a priori to the scheduler. This allows us to focus on the full potential performance of goal-oriented policies (and the backfill policies studied), without the complex interference from inaccurate job runtime information. Our future work will include the results that use imprecise job runtimes.

5 Evaluating Policy Design Choices

To design and implement goal-oriented policies, many choices need to be studied, including fixed or dynamic excessive wait time, which search algorithm, strict or relaxed differences, and what reference values for the Eq-Tradeoff model. We consider these parameters part of the model or policy design choices, as opposed to the parameters that can be configured by system administrators. In this section, we evaluate these design choices. Alternative scheduling policies will be studied in Section 6.

5.1 Sensitivity of Performance to Fixed Wait Time Thresholds

A wait time threshold is needed for computing the total excessive wait, which is one of the starvation measures studied. A fixed wait time threshold, such as 50 hours, seems simple and fairly intuitive. The question is how sensitive is the scheduling performance to the value of a fixed threshold.

To study this question, we use $\text{Strict Lexical}(T^{\omega}_{w} \rightarrow avgX)$, in which T^{ω}_{w} is the total excessive wait, computed using a fixed threshold, ω. Using the strict differences allows us to focus on the impact of the threshold alone, but the results of using the relaxed difference with 1 percent slack are qualitatively similar (not shown). The policy is implemented using DDS-LXF and $L = 1{,}000$. More about the performance impact of search algorithms and L will be studied in Section 5.2.

We vary the value of ω between 0 and 300 hours. Fig. 2 shows the results for $\omega = 50$, 100, and 300 hours. Figs. 2a and 2b plot the maximum wait of using each ω for each month under the original load and $\rho = 0.9$, respectively. For the original load, as shown in Fig. 2a, a maximum wait of 50 hours is achievable for each month except July 2003 (which has the highest load). However, if $\omega > 50\ \text{hours}$ is used, the maximum wait typically increases; in particular, with an ω of 100 hours, the maximum wait is roughly 100 hours in all months.

For $\rho = 0.9$, shown in Fig. 2b, a maximum wait of 50 hours cannot be achieved for most months even if $\omega = 50\ \text{hours}$, but a maximum wait of roughly 100 hours is achievable each month by using $\omega = 100\ \text{hours}$. However, increasing ω to 300 hours results in a maximum wait of 300 hours in all but two months. On the other hand, as shown in Fig. 2c, the average bounded slowdown can be improved when the maximum wait degrades as a result of a larger ω, but the improvement is typically small (except for July 2003 and August 2003).

Obviously, the maximum wait cannot be arbitrarily reduced by using a small wait time threshold. For example, under $\rho = 0.9$ (in Fig. 2b), the maximum wait using $\omega = 50\ \text{hours}$ is worse than or similar to that of using $\omega = 100\ \text{hours}$ in most months (except October 2003). In fact, too small a threshold can cause poor maximum wait. In the extreme case, where $\omega = 0$, the maximum wait is as poor as thousands of hours in several months (not shown

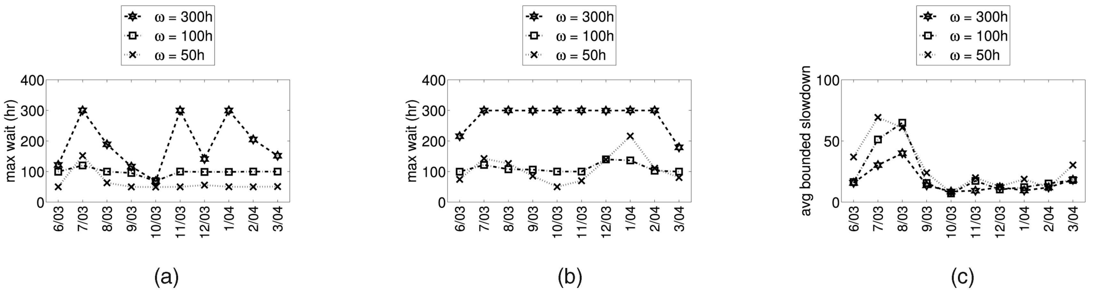


Fig. 2. Sensitivity of $\text{Strict Lexical}(T_w^\omega \rightarrow avgX)$ to fixed wait time thresholds (ω); $L = 1{,}000$. (a) Maximum wait; $\rho = \text{original load}$. (b) Maximum wait; $\rho = 0.9$. (c) Average bounded slowdown; $\rho = 0.9$.

to conserve space). This is because minimizing T_w^ω using $\omega = 0$ is equivalent to minimizing the total wait time, which is the same as minimizing the average wait.

The key point is that the value of ω has a significant impact on the maximum wait; using too small or too large a wait time threshold is detrimental to the performance. The results motivate the use of a dynamic threshold. In the remainder of this paper, we assume that the excessive wait is computed using the *dynamic* threshold, as defined in Section 3.1.

5.2 Efficiency of Search Algorithms

Since it is computationally infeasible to fully explore the potentially large search space, an efficient search algorithm is critical to the goal-oriented policies. An algorithm is said to be more *efficient* than another if it is faster in finding good solutions with respect to the given objectives. As discussed in Section 3.4, four search algorithms are studied: DDS-LXF, DDS-FCFS, LDS-LXF, and LDS-FCFS. The question is which one is the most efficient algorithm for the problem studied.

To answer the question, we could compare the number of tree nodes visited (i.e., L) required for each search algorithm to approximate the "true" performance of each policy for each month; the algorithm that requires the smallest L would be the most efficient. The dilemma is that to know the true performance, we need to explore all possible schedules at each scheduling decision point, which is computationally infeasible. As an alternative, we project the true performance of each policy. Our approach is to examine how the performance of each policy using each search algorithm changes as L increases, up to a certain point that is still computationally feasible to simulate. We then compare the results of different search algorithms for each policy, to project where their performance converges. Finally, we compare how fast each algorithm approaches the projected performance convergence point.

We simulate L in the range of 100 and 64,000, unless noted otherwise. Note that in most scheduling decision points for $\rho = 0.9$, at least 10 jobs are waiting, i.e., almost 10 million nodes in the tree (Fig. 1d). This range of L does not even cover 1 percent of the tree in most cases. Thus, it is critical to explore good schedules early in time.

The execution time at each scheduling decision increases roughly linearly with L and, to a less extent, the number of waiting jobs. In our simulation, it typically takes under a few tens of milliseconds for $L = 1{,}000 - 8{,}000$ in a tree of 30 jobs, larger than the trees used in 70 percent of the scheduling decision points for $\rho = 0.9$. Our simulator is written in Java PL and run on a 2-GHz Intel Pentium-IV Windows XP with 512-Mbyte memory.

Below, we report the results of $\text{Strict Lexical}(T_w \rightarrow avgX)$ and $\text{Strict Lexical}(maxW \rightarrow avgX)$. Other policies will be commented. The results of $\text{Strict Lexical}(T_w \rightarrow avgX)$ are shown for a typical month (September 2003) in Fig. 3 and for the most exceptional month (January 2004) in Fig. 4. For simplicity, only three algorithms are shown: DDS-FCFS, DDS-LXF, and LDS-LXF. We plot their average bounded slowdown, maximum wait, and total normalized excessive wait, as a function of L.

As shown in both Figs. 3 and 4, the performance difference among different algorithms roughly reduces as L increases. In the typical month, the three algorithms almost converge at $L = 64{,}000$. In addition, DDS-LXF quickly $(400 \leq L \leq 1{,}000)$ approaches the convergence point for all relevant performance measures. On the other hand, it takes a much larger L for LDS-LXF to improve the

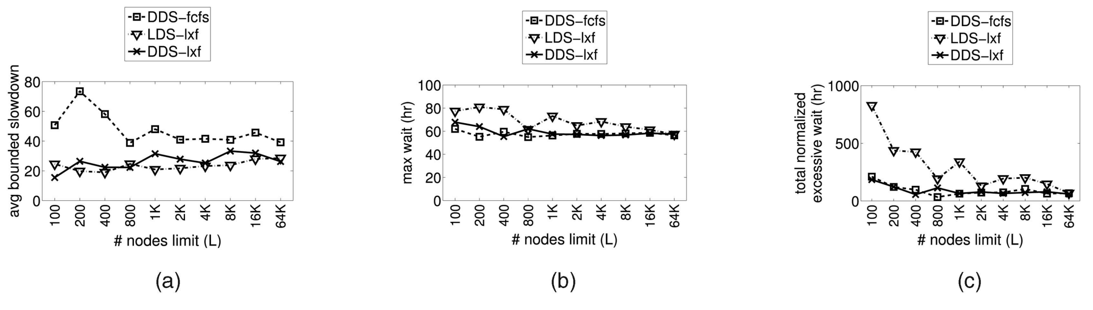


Fig. 3. Comparing search efficiency for $\text{Strict Lexical}(T_w \rightarrow avgX)$; a typical month (September 2003); $\rho = 0.9$. (a) Average bounded X versus L. (b) Maximum wait versus L. (c) Total $E_{\text{FCFS-bf}}^{98\%W}$ versus L.

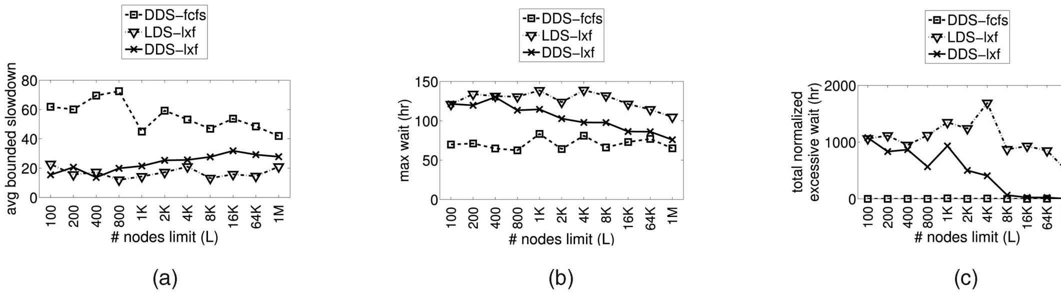


Fig. 4. Comparing search efficiency for $\mathrm{Strict\ Lexical}(T_w \rightarrow avgX)$; the most exceptional month (January 2004); $\rho = 0.9$. (a) Average bounded X versus L. (b) Maximum wait versus L. (c) Total $E^{maxW}_{\mathrm{FCFS\text{-}bf}}$ versus L.

maximum wait and total normalized excessive wait (Figs. 3b and 3c) and for DDS-FCFS to improve the average bounded slowdown (Fig. 3a). An even larger L is needed for LDS-FCFS (not shown) to improve the average slowdown than that for DDS-FCFS. The exceptional month is different in that it takes a much larger L to reduce the performance gap of the algorithms, due to a larger backlog (and thus a larger search space) in that month. Thus, for this month, we simulate L up to 1 million.

Based on the above results, DDS-LXF appears to be more efficient than the other three search algorithms studied for $\mathrm{Strict\ Lexical}(T_w \rightarrow avgX)$. We found this to be the case for most policies studied, except $\mathrm{Lexical}(maxW \rightarrow avgX)$ and $\mathrm{Lexical}(maxW \rightarrow avgW)$, which minimize $maxW$ as the primary objective. These two policies are discussed next, followed by more discussions of comparing search algorithms.

For $\mathrm{Lexical}(maxW \rightarrow avgX)$ and $\mathrm{Lexical}(maxW \rightarrow avgW)$, DDS-LXF still appears to be the most efficient algorithm for most months studied. However, it is questionable whether this is true for several months, especially January and February 2004. Fig. 5 shows the results of $\mathrm{Strict\ Lexical}$ $(maxW \rightarrow avgX)$ in January 2004, in which the performance difference among the search algorithms is the largest. As shown in Figs. 5a and 5b, even at $L = 64{,}000$, DDS-FCFS and DDS-LXF still have very different performance, in that DDS-LXF favors the average bounded slowdown and DDS-FCFS favors the maximum wait, as expected.

Increasing L to 1 million sufficiently improves the average bounded slowdown of DDS-FCFS, but DDS-LXF still has over 60 percent worse maximum wait than that of DDS-FCFS. Using LDS-LXF is even less efficient than using DDS-LXF in reducing the maximum wait, but their difference is not as large as that for $\mathrm{Strict\ Lexical}(T_w \rightarrow avgX)$. The results of the total normalized excessive wait in Fig. 5c are similar to that of the maximum wait. Thus, for January 2004 and February 2004 (not shown), and perhaps a few other months, the performance of using DDS-FCFS may be the closest to the true performance of $\mathrm{Strict\ Lexical}(maxW \rightarrow avgX)$, for the range of L studied.

The results in this section suggest that DDS is more efficient than LDS for the problem studied, as expected and discussed in Section 2.4. Second, the results of $\mathrm{Lexical}(T_w \rightarrow avgX)$ suggest that it is easier to achieve low T_w by closely following the LXF priority than to achieve low $avgX$ by closely following the FCFS priority, and that the LXF branching heuristic is more efficient than FCFS for the objective studied. Third, the LXF priority is more consistent with minimizing T_w than with minimizing $maxW$. These two starvation measures will be further compared in Section 6.

Although using DDS-LXF with a limited L may favor the slowdown measure to some extent, this should be acceptable since low average job slowdown is typically important. The only exceptions are policies whose primary objectives are in strong conflict with optimizing the average slowdown, e.g., $\mathrm{Lexical}(maxW \rightarrow avgX)$. In these cases, a branching heuristic favoring the primary objective can be used.

Finally, we comment on the effect of L. In most cases where DDS-LXF is more efficient than others, we find that a small L between 400 and 1,000 results in a fairly similar performance to that of using a larger L up to 64,000, in terms of the maximum wait, average wait, and the average bounded slowdown. For the purpose of comparing policies in the remainder of this paper, we show the results of using $L = 4{,}000$ (unless otherwise noted). This size is large enough to show potential advantages or problems of the policies, yet small enough for practical implementation to

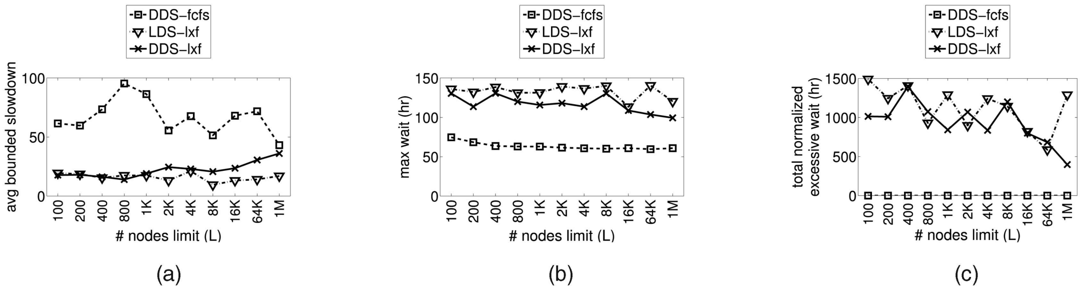


Fig. 5. Comparing search efficiency for $\mathrm{Strict\ Lexical}(maxW \rightarrow avgX)$; January 2004 (the largest performance gap); $\rho = 0.9$. (a) Average bounded X versus L. (b) Maximum wait versus L. (c) Total $E^{maxW}_{\mathrm{FCFS\text{-}bf}}$ versus L.

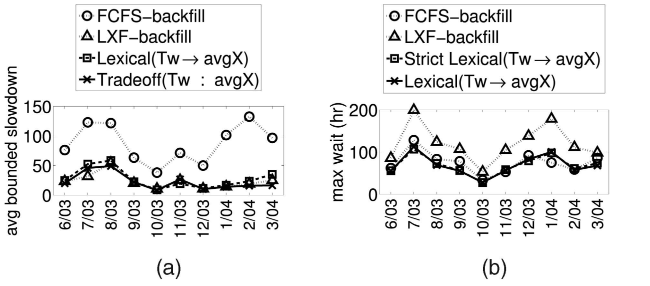


Fig. 6. Strict versus relaxed $\mathrm{Lexical}(T_w \rightarrow avgX)$; $\rho = 0.9$; $L = 4{,}000$. (a) Average bounded X. (b) Maximum wait.

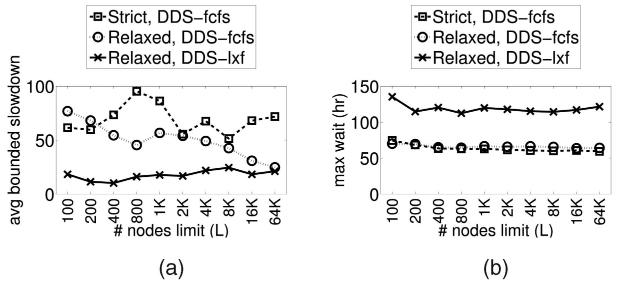


Fig. 7. Strict versus relaxed $\mathrm{Lexical}(maxW \rightarrow avgX)$; $\rho = 0.9$; January 2004. (a) Average bounded X versus L. (b) Maximum wait versus L.

represent the performance that can be achieved on real systems.

5.3 Strict versus Relaxed Difference

So far, we have assumed that the strict difference is used when comparing candidate schedules. However, a relaxed difference with a small slack may be preferred, to avoid emphasizing too much on marginal differences between schedules compared. In this section, we evaluate the impact of using the relaxed difference with 1 percent slack. Results are shown for $\mathrm{Lexical}(T_w \rightarrow avgX)$ and $\mathrm{Lexical}(maxW \rightarrow avgX)$ in Figs. 6 and 7. Other policies will be commented. Recall that "Strict" is part of the policy names if the strict difference is used; otherwise, the relaxed difference is assumed.

Figs. 6a and 6b plot the average bounded slowdown and maximum wait of Strict and Relaxed $\mathrm{Lexical}(T_w \rightarrow avgX)$ for each month. The two baseline backfill policies are also included. The results show that the two Lexical policies have similar performance, except that using relaxed differences considerably improves the average bounded slowdown in July 2003.

In contrast, using the relaxed difference has a much larger impact on $\mathrm{Lexical}(maxW \rightarrow avgX)$, because minimizing $maxW$ does not leave as much space for optimizing the secondary objective. In particular, the effect of the relaxed difference is the largest for January 2004, which is the month that has the largest performance gap among different search algorithms when the strict difference is used, as discussed in Section 5.2. Figs. 7a and 7b plot the average bounded slowdown and maximum wait versus L of three versions of $\mathrm{Lexical}(maxW \rightarrow avgX)$: Strict version with DDS-FCFS (repeated from Fig. 5), relaxed version with DDS-FCFS, and relaxed version with DDS-LXF.

Fig. 7a shows that using relaxed differences significantly improves the average bounded slowdown of DDS-FCFS for $L \geq 16{,}000$. In addition, both DDS-FCFS and DDS-LXF have comparable average bounded slowdown at $L = 64{,}000$, now that relaxed differences are used. At the same time, using the relaxed difference does not affect the maximum wait much, as shown in Fig. 7b (and total normalized excessive wait measures, not shown). Thus, using relaxed differences, DDS-FCFS is even more efficient than DDS-LXF to achieve $\mathrm{Lexical}(maxW \rightarrow avgX)$ for January 2004. On the other hand, in most of the other months, the effect of relaxed differences is considerably smaller and that DDS-LXF is still more efficient for this policy.

Finally, for Eq-Tradeoff policies, using relaxed differences may improve or degrade measures used in both objectives, but we found more improvement than degradation. Based on these results, the relaxed difference is adopted in the models.

5.4 Definition of Reference Values in the Eq-Tradeoff Model

In this section, we compare three versions (A, B, and C) of reference values for the Eq-Tradeoff model. As discussed in Section 3.2, version A is more intuitive, but versions B and C are more theoretically attractive because of their uniqueness property (if strict differences are used).

Fig. 8 shows the results for $\mathrm{Tradeoff}(T_w : avgX)$, using the DDS-LXF search algorithm with $L = 4{,}000$ and the relaxed difference. Fig. 8a shows that the three versions of $\mathrm{Tradeoff}(T_w : avgX)$ have almost identical maximum wait each month. They also have similar average wait and total normalized excessive wait times (not shown). In fact, as shown in Fig. 8b, they also have fairly similar average bounded slowdown in each month, except August 2003. On the other hand, Fig. 8c shows that their maximum bounded

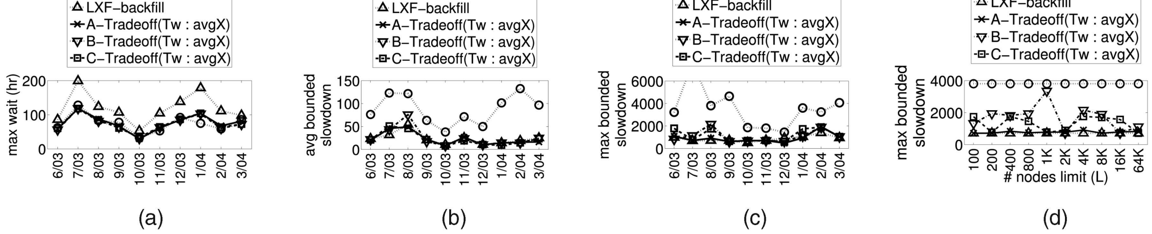


Fig. 8. Comparing three versions of $\mathrm{Tradeoff}(T_w \rightarrow avgX)$; $\rho = 0.9$; graphs (a)-(c): $L = 4{,}000$; graph (d): August 2003 (exceptional). (a) Maximum wait. (b) Average bounded slowdown. (c) Maximum bounded slowdown. (d) Maximum bounded slowdown versus L.

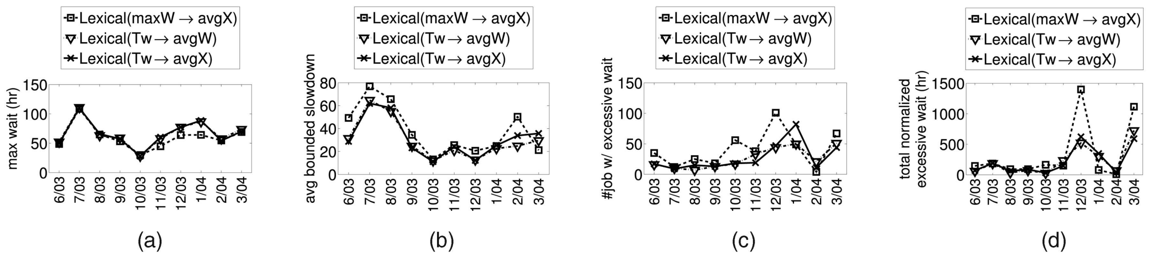


Fig. 9. Impact of alternative objectives using the Lexical model; $\rho = 0.9$; $L = 64{,}000$. (a) Maximum wait. (b) Average bounded X. (c) Number of jobs with $E^{98\%W}_{\text{FCFS-bf}} > 0$. (d) Total $E^{98\%W}_{\text{FCFS-bf}}$.

slowdown performance differs in more than one month, in that version A outperforms the other two versions in three months (June 2003, August 2003, and January 2004). A problem of versions B and C is that their maximum bounded slowdown fluctuates with L more than that of A in these three months and a few other months. The worst case occurs in August 2003, shown in Fig. 8d, which plots the maximum bounded slowdown versus L under each policy in that month. In other months, the maximum bounded slowdown of B and C does not fluctuate with L as much, relative to the difference between them and FCFS-backfill.

Since version A is more robust in the maximum bounded slowdown performance and is also more intuitive, it is adopted to define the reference values used in the Eq-Tradeoff model.

6 Policy Comparisons

In Section 5, we have made several choices for designing and implementing the goal-oriented policies. They are the dynamic threshold for computing the excessive wait, the DDS-LXF search algorithm for all policies except $\text{Lexical}(maxW \rightarrow avgX)$ and $\text{Lexical}(maxW \rightarrow avgW)$, the relaxed difference with 1 percent slack, and the A-Tradeoff model for modeling equal trade-offs. Now, we are ready to compare the performance of alternative goal-oriented policies. All of them are designed with the same two performance goals in mind, i.e., preventing starvation and favoring shorter jobs, but they differ in the measures optimized and/or objective models used. Our purpose is to understand the performance impact of alternative objectives and measures and help define appropriate objectives.

We evaluate alternative objectives using the Lexical model in Section 6.1 and the Eq-Tradeoff model in Section 6.2. They will be compared with the backfill policies in Section 6.3. Most results are fairly intuitive, except when making an equal trade-off between $maxW$ and average performance measures.

6.1 Alternative Lexical Policies

In this section, we compare alternative Lexical policies, including alternative starvation measures, alternative average performance measures, and alternative orders of measures.

Fig. 9 compares three policies: $\text{Lexical}(maxW \rightarrow avgX)$, $\text{Lexical}(T_w \rightarrow avgW)$, and $\text{Lexical}(T_w \rightarrow avgX)$. Note that $\text{Lexical}(maxW \rightarrow avgX)$ uses the DDS-FCFS algorithm, but the other policies use DDS-LXF, based on the discussions in Sections 5.2 and 5.3. Since $\text{Lexical}(maxW \rightarrow avgX)$ requires a large L to reduce the average bounded slowdown, the results are shown for $L = 64{,}000$, the largest L simulated for most months.

Figs. 9a and 9b plot the maximum wait and average bounded slowdown, respectively, for each policy. Fig. 9c plots the number of *relatively* long waiting jobs ($>$ the ninety-eighth percentile wait under FCFS-backfill, typically 30-50 hours) in each month. Fig. 9d plots total normalized excessive wait of these jobs.

First, for $\text{Lexical}(maxW \rightarrow avgX)$ and $\text{Lexical}(T_w \rightarrow avgX)$, which differ only in the starvation measure used, Fig. 9a shows that $\text{Lexical}(maxW \rightarrow avgX)$ achieves considerably lower maximum wait in several months (December 2003, January 2004, and February 2004). However, it has worse average bounded slowdown for many months, as shown in Fig. 9b. Furthermore, as a result of minimizing the maximum wait, more jobs under $\text{Lexical}(maxW \rightarrow avgX)$ incur a relatively long wait time than under $\text{Lexical}(T_w \rightarrow avgX)$ in most months, as shown in Fig. 9c. Note that this is still true if $\text{Lexical}(maxW \rightarrow avgX)$ uses the other three search algorithms. On the other hand, Fig. 9d shows that these two policies have similar total normalized excessive wait time (w.r.t. the ninety-eighth percentile wait under FCFS-backfill) for most months (except December 2003 and March 2004). The results in Figs. 9c and 9d imply that relatively long-waiting jobs under $\text{Lexical}(maxW \rightarrow avgX)$ in fact incur a shorter wait time in average than that under $\text{Lexical}(T_w \rightarrow avgX)$ for most months.

The bottom line is if minimizing the maximum wait is the most important objective, $\text{Lexical}(maxW \rightarrow avgX)$ would be a better policy. However, most likely, minimizing the average slowdown is just as important, in which case $\text{Lexical}(T_w \rightarrow avgX)$ is preferable, if the Lexical model is used.

Regarding the impact of alternative average performance measures, we can see that, by comparing $\text{Lexical}(T_w \rightarrow avgX)$ and $\text{Lexical}(T_w \rightarrow avgW)$, whether $avgW$ or $avgX$ is optimized in the secondary objective makes only minimal difference in the performance. On the other hand, if the primary objective optimizes $avgW$ or $avgX$, then there is a starvation problem, in that the maximum wait in some months is even over 1,000 hours (not shown to conserve space).

6.2 Alternative Eq-Tradeoff Policies

In this section, we study the impact of using alternative measures on the performance of Eq-Tradeoff policies. The results are shown in Figs. 10a, 10b, 10c, and 10d, which

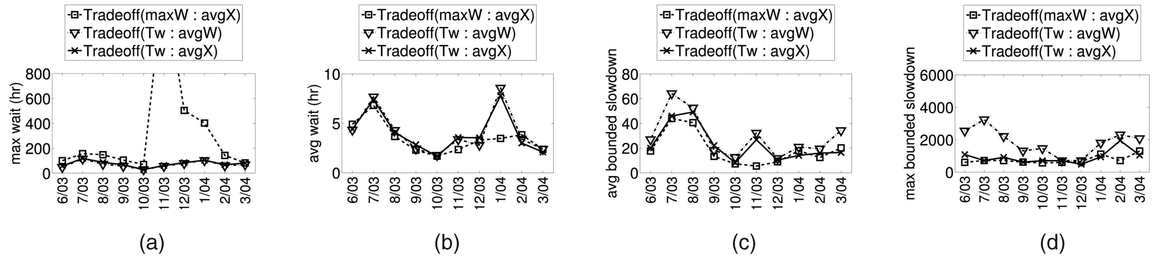


Fig. 10. Impact of alternative objectives using the Eq-Tradeoff model; $\rho = 0.9$; $L = 4{,}000$. (a) Maximum wait. (b) Average wait. (c) Average bounded slowdown. (d) Maximum bounded slowdown.

plot the average and maximum bounded slowdown and wait time of three policies: $\text{Tradeoff}(maxW : avgW)$, $\text{Tradeoff}(T_w : avgW)$, and $\text{Tradeoff}(T_w : avgX)$, using version A of reference values and the DDS-LXF search with $L = 4{,}000$.

First, we see that whether $avgX$ or $avgW$ is optimized makes a larger difference now than when they are used as the secondary objective in Lexical policies. More specifically, as shown in Figs. 10c and 10d, $\text{Tradeoff}(T_w : avgX)$ has significantly lower average bounded slowdown in a few months and maximum bounded slowdown in most months, compared with $\text{Tradeoff}(T_w : avgW)$. The results are similar for the range of L studied. The performance of these two Eq-Tradeoff policies are otherwise fairly similar, e.g., maximum and average wait, as shown in Figs. 10a and 10b. Thus, to favor shorter jobs, optimizing $avgX$ may be preferred.

Second, as shown in Fig. 10a, $\text{Tradeoff}(maxW : avgX)$ has a starvation problem (400-1500 hours of maximum wait time in several months). The same problem also happens to $\text{Tradeoff}(maxW : avgW)$ (not shown). This is surprising as minimizing $maxW$ is in the objectives. Note that the problem occurs for all implementations studied, including different search algorithms, definitions of differences, values of L (up to 64,000), and versions of reference values.

To investigate the problem, we simulate another policy, which is similar to $\text{Tradeoff}(maxW : avgX)$ but uses the *maximum excessive wait* in place of the maximum wait. We found that this policy has a similar starvation problem (not shown). Note that the maximum wait or maximum excessive wait measures the performance of a single job, but the average wait and slowdown or total excessive wait are all-job measures. The results suggest that making an equal trade-off between single-job and all-job measures is problematic.

An important lesson is that optimizing measures that seem similar (i.e., $maxW$ and T_w) may result in very different performance; care should be taken when designing objectives. In addition, T_w is a better starvation measure than $maxW$, because there is a problem of using $maxW$ and also that minimizing $maxW$ may be too strict an objective for the purpose of preventing starvation.

6.3 Further Comparisons of Policies

We now compare the performance of the best Lexical and Eq-Tradeoff policies studied (i.e., optimizing T_w and $avgX$) and also compare their performance with the backfill policies.

Figs. 11a, 11b, 11c, and 11d plot the maximum wait, the total normalized excessive wait, and the average and maximum bounded slowdown of each policy in each month. First, as expected, there is a large performance difference between the two baseline backfill policies, in that FCFS-backfill favors the maximum wait and total normalized excessive wait, but LXF-backfill favors the slowdown measures and the average wait (not shown). In contrast, by simultaneously optimizing T_w and $avgX$ through search, $\text{Tradeoff}(T_w : avgX)$ outperforms both backfill policies for all performance measures studied. $\text{Lexical}(T_w \rightarrow avgX)$ is fairly similar to $\text{Tradeoff}(T_w : avgX)$, except it has much worse maximum bounded slowdown, which is nevertheless still much better than that of FCFS-backfill in most months.

To further understand the characteristics of each policy, Fig. 12 plots the average wait time of each $N \times T$ job class under each policy, for a representative month (July 2003). Jobs are partitioned according to five disjoint ranges of actual job runtime (T) and five disjoint ranges of requested nodes (N). The upper bounds of each range of T and N are shown in the graphs. These results demonstrate a trend observed in most months: 1) FCFS-backfill tends to provide poor performance for wide jobs $(N > 32)$, regardless of

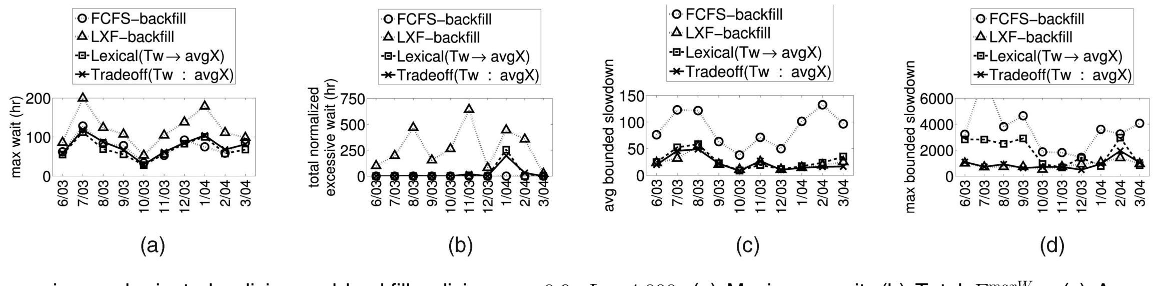


Fig. 11. Comparing goal-oriented policies and backfill policies; $\rho = 0.9$; $L = 4{,}000$. (a) Maximum wait. (b) Total $E^{maxW}_{\text{FCFS-bf}}$. (c) Average bounded slowdown. (d) Maximum bounded slowdown.

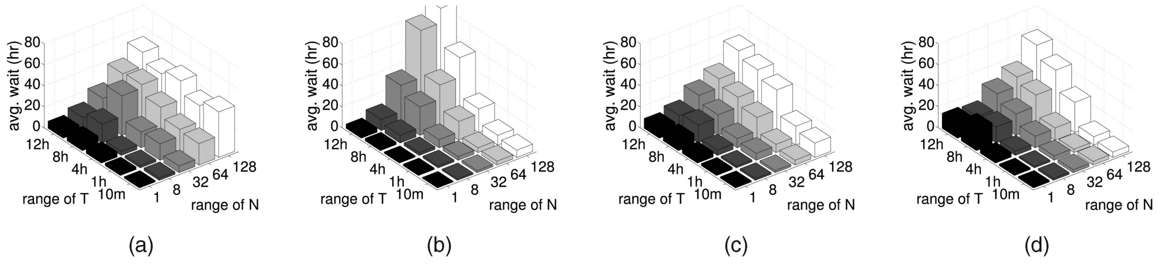


Fig. 12. Average wait time of each job class $(T \times N)$ in a representative month (July 2003); $\rho = 0.9$; graphs (c) and (d): $L = 4{,}000$. (a) FCFS-backfill. (b) LXF-backfill. (c) $\mathrm{Lexical}(T_w \rightarrow avgX)$. (d) $\mathrm{Tradeoff}(T_w : avgX)$.

their runtime; 2) LXF-backfill significantly improves short-wide jobs ($T \leq 1$ hour and $N > 32$) by sacrificing long-wide jobs ($T > 8$ hours and $N > 8$), compared with FCFS-backfill; 3) in contrast to LXF-backfill, both goal-oriented policies, and especially $\mathrm{Tradeoff}(T_w : avgX)$, improve short-wide jobs of FCFS-backfill, but not so much as to sacrifice long-wide jobs. In fact, the performance of the short-wide jobs under $\mathrm{Tradeoff}(T_w : avgX)$ is as good as that under LXF-backfill. To achieve such performance, $\mathrm{Tradeoff}(T_w : avgX)$ trades some of the performance of long-narrow jobs ($T > 8$ hours and $N \leq 8$), which could have taken advantage of the backfilling mechanism (because they are narrow) under backfill policies.

The key conclusion is that goal-oriented policies can achieve a low average wait and slowdown as good as that of LXF-backfill, while having a low maximum wait similar to that of FCFS-backfill for most months studied.

7 Summary and Conclusions

To deal with multiple performance goals, current job scheduling policies that run on production parallel computers use many parameters for defining job priority and/or various scheduling limits. Providing many parameters may seem flexible, but determining their values is difficult. Rather than requiring tuning low-level scheduling parameters, we propose goal-oriented scheduling policies, which allow system administrators to specify only the high-level scheduling performance goals. As can see seen, using such policies places the burden on the policy designers rather than the system administrators. We investigate several design and implementation choices and evaluate the potential performance of the goal-oriented policies. Search algorithms are used to automatically find the schedule that provides the best compromise at each scheduling decision point, according to the given objectives.

A key component of goal-oriented policies is the multi-objective models, which should be intuitive and require no manual tuning of low-level parameters. Furthermore, they should be designed such that the schedulers can automatically and efficiently optimize for the given objectives. Under these considerations, we define two models: 1) the Lexical model, based on the previous lexicographical ordering approach, which is simple but requires that the objectives be ranked in their importance, and 2) the Eq-Tradeoff model, proposed in this study to model objectives that are equally important. Three versions (A, B, and C) of the Eq-Tradeoff model are studied. We find that they have fairly similar performance for the workloads and objectives studied. We adopt version A because it is simple and more intuitive than the others.

We focus on two scheduling performance requirements, commonly placed on general-purpose parallel computer systems: preventing starvation and favoring shorter jobs. To prevent starvation, we consider optimizing the total excessive wait (T_w) or maximum wait time $(maxW)$. To favor shorter jobs, we consider optimizing the average slowdown $(avgX)$ or average wait time $(avgW)$. Using these objectives and the two objective models, a set of goal-oriented policies is defined. We study these policies to understand the impact of alternative objective models and alternative objectives, with respect to the two performance requirements considered.

We compare goal-oriented policies with FCFS-backfill and LXF-backfill, which represent the two extremes of backfill policies in that FCFS-backfill favors the maximum wait and LXF-backfill favors the average wait and slowdown. Policies are evaluated by simulation using job traces from three parallel computer systems. The results are reported for the 10 monthly NCSA/IA-64 workloads. The conclusions of using two SP2 traces are qualitatively similar. Both original-load and artificially created high-load $(\rho = 0.9)$ workloads are studied. A wide range of policy performance measures was used, including average performance measures, the maximum wait, and two total normalized excessive wait time measures.

Our results show that goal-oriented policies have the potential to significantly improve backfill policies. More specifically, $\mathrm{Tradeoff}(T_w : avgX)$, which simultaneously optimizes T_w and $avgX$, achieves the best or close to the best average performance measures as well as the maximum wait and total normalized excessive wait for the workloads studied. The performance of $\mathrm{Lexical}(T_w \rightarrow avgX)$ is similar except for a worse maximum bounded slowdown, because optimizing $avgX$ is the secondary objective. Second, T_w is a better starvation measure than $maxW$. One reason is that optimizing T_w, as the primary objective, leaves more room for optimizing other measures. Even more importantly, when simultaneously optimizing $maxW$ and average performance measures, a starvation problem occurs. The results suggest that making an equal trade-off between single-job $(maxW)$ and all-job $(avgX, avgW)$ measures is problematic. Other results are fairly intuitive: optimizing $avgX$ can achieve better average and maximum bounded slowdown than that of optimizing $avgW$, when used in the Eq-Tradeoff model; optimizing average performance measure as the primary objective results in a starvation problem.

For the search algorithms, the tree traversing algorithms (DDS and LDS) as well as the branching heuristics (LXF and FCFS) have a significant impact on the search efficiency. For the multiobjective scheduling problem studied, DDS-LXF appears to be more efficient than the other three algorithms studied for the policies studied, except it is DDS-FCFS for Lexical$(maxW \rightarrow avgX)$ and Lexical$(maxW \rightarrow avgW)$.

Allowing specifying high-level performance goals and automatically optimizing the performance through search not only reduces the administrator effort and error but also has the potential to improve the scheduling performance. The work reported here represents a strong step in that direction. Our ongoing work includes further improving the efficiency of search, studying the performance impact of inaccurate job runtime estimates, and studying other important performance goals, including fair share and special priority.

Acknowledgments

The work of Sangsuree Vasupongayya was done while she was studying at Portland State University.

References

[1] *OpenPBS,* http://www.openpbs.org/docs.html, Aug. 2000.
[2] *LSF Scheduler,* Platform Computing, http://www.platform.com/, 2008.
[3] *Maui Scheduler,* http://www.supercluster.org/maui/, 2008.
[4] B. Chun and D. Culler, "User-Centric Performance Analysis of Market-Based Cluster Batch Schedulers," *Proc. Second IEEE Int'l Symp. Cluster Computing and the Grid (CCGRID '02),* pp. 30-38, May 2002.
[5] K. Kurowski, J. Nabrzyski, A. Oleksiak, and J. Weglarz, "Scheduling Jobs on the Grid-Multicriteria Approach," *Computational Methods in Science and Technology,* vol. 12, no. 2, pp. 123-138, 2006.
[6] *Parallel Workloads Archive,* http://www.cs.huji.ac.il/labs/parallel/workload/models.html, 2008.
[7] S. Vasupongayya, S.-H. Chiang, and B. Massey, "Search-Based Job Scheduling for Parallel Computer Workloads," *Proc. IEEE Int'l Conf. Cluster Computing (CLUSTER '05),* Sept. 2005.
[8] S. Vasupongayya and S.-H. Chiang, "Multi-Objective Models for Scheduling Jobs on Parallel Computer System," *Proc. IEEE Int'l Conf. Cluster Computing (CLUSTER '06),* short paper, Sept. 2006.
[9] D. Lifka, "The ANL/IBM SP Scheduling System," *Proc. First Job Scheduling Strategies for Parallel Processing (JSSPP '95),* Apr. 1995.
[10] S. Kannan, M. Roberts, P. Mayes, D. Brelsford, and J.F. Skovira, *Workload Management with Loadleveler,* IBM, Nov. 2001.
[11] D. Zotkin and P.J. Keleher, "Job-Length Estimation and Performance in Backfilling Schedulers," *Proc. Eighth IEEE Int'l Symp. High Performance Distributed Computing (HPDC '99),* pp. 236-243, Aug. 1999.
[12] S.-H. Chiang and M.K. Vernon, "Production Job Scheduling for Parallel Shared Memory Systems," *Proc. 15th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '01),* Apr. 2001.
[13] S.-H. Chiang, A. Dusseau-Arpaci, and M.K. Vernon, "The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance," *Proc. Eighth Job Scheduling Strategies for Parallel Processing (JSSPP '02),* pp. 103-127, July 2002.
[14] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Selective Reservation Strategies for Backfill Job Scheduling," *Proc. Eighth Job Scheduling Strategies for Parallel Processing (JSSPP '02),* pp. 55-71, July 2002.
[15] A.W. Mu'alem and D.G. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 6, pp. 529-543, June 2001.
[16] S.-H. Chiang and C. Fu, "Re-Evaluating Reservation Policies for Backfill Scheduling on Parallel Systems," *Proc. 16th IASTED Int'l Conf. Parallel and Distributed Computing and Systems (PDCS '04),* Nov. 2004.
[17] D. Talby and D.G. Feitelson, "Improving and Stabilizing Parallel Computer Performance Using Adaptive Backfilling," *Proc. 19th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '05),* Apr. 2005.
[18] B. Lawson and E. Smirni, "Self-Adaptive Scheduler Parameterization via Online Simulation," *Proc. 19th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '05),* Apr. 2005.
[19] M. Ehrgott and X. Gandibleux, "A Survey and Annotated Bibliography of Multiobjective Combinatorial Optimization," *OR Spektrum,* vol. 22, pp. 425-460, 2000.
[20] J.M. Crawford, "Solving Satisfiability Problems Using a Combination of Systematic and Local Search," *Second DIMACS Challenge: Cliques, Coloring, and Satisfiability,* Oct. 1993.
[21] W.D. Harvey and M.L. Ginsberg, "Limited Discrepancy Search," *Proc. 14th Int'l Joint Conf. Artificial Intelligence (IJCAI '95),* Aug. 1995.
[22] R.E. Korf, "Improved Limited Discrepancy Search," *Proc. 13th Nat'l Conf. Artificial Intelligence (AAAI '96),* pp. 209-215, Aug. 1996.
[23] T. Walsh, "Depth-Bounded Discrepancy Search," *Proc. 15th Int'l Joint Conf. Artificial Intelligence (IJCAI '97),* vol. 2, pp. 1388-1393, Aug. 1997.
[24] R. Gangadharan and C. Rajendran, "A Simulated Heuristic for Scheduling in a Flowshop with Bicriteria," *Computers and Industrial Eng.,* vol. 27, nos. 1-4, pp. 473-476, 1994.

Su-Hui Chiang received the BS and MS degrees in applied mathematics from the National Chung-Hsing University, Taichung, Taiwan, R.O.C., and the MS and PhD degrees in computer science from the University of Wisconsin, Madison. She is an assistant professor in the Department of Computer Science, Portland State University, Portland, Oregon. Her research interests include performance analysis and modeling, and job scheduling for parallel and distributed systems.

Sangsuree Vasupongayya received the BEng degree from the Prince of Songkla University, Songkla, Thailand, in 1996, the MSc degree from California State University, Chico, in 2001, and the PhD degree in computer science from Portland State University, Portland, Oregon, in 2008. She is a lecturer in the Department of Computer Engineering, Prince of Songkla University. Her research interests include scheduling jobs in parallel computer environment and computer security.