2012 IEEE 26th International Parallel and Distributed Processing Symposium

ExPERT: Pareto-Efficient Task Replication on Grids and a Cloud

Orna Agmon Ben-Yehuda*, Assaf Schuster*, Artyom Sharov*, Mark Silberstein*, and Alexandru Iosup‡
*Technion–Israel Institute of Technology, Haifa, 32000, Israel, Email: {ladypine,assaf,sharov,marks}@cs.technion.ac.il
‡Parallel and Distributed Systems Group, TU Delft, the Netherlands, Email: A.Iosup@tudelft.nl

***Abstract*—Many scientists perform extensive computations by executing large bags of similar tasks (BoTs) in mixtures of computational environments, such as grids and clouds. Although the reliability and cost may vary considerably across these environments, no tool exists to assist scientists in the selection of environments that can both fulfill deadlines and fit budgets. To address this situation, we introduce the `ExPERT` BoT scheduling framework. Our framework systematically selects from a large search space the Pareto-efficient scheduling strategies, that is, the strategies that deliver the best results for both makespan and cost. `ExPERT` chooses from them the best strategy according to a general, user-specified utility function. Through simulations and experiments in real production environments, we demonstrate that `ExPERT` can substantially reduce both makespan and cost in comparison to common scheduling strategies. For bioinformatics BoTs executed in a real mixed *grid+cloud* environment, we show how the scheduling strategy selected by `ExPERT` reduces both makespan and cost by 30%-70%, in comparison to commonly-used scheduling strategies.**

***Keywords*—bags-of-tasks; cloud; grid; Pareto-frontier**

I. Introduction

The emergence of cloud computing creates a new opportunity for many scientists: using thousands of computational resources assembled from both grids and clouds to run their large-scale applications. This opportunity, however, also adds complexity, as the shared grid systems and the pay-per-use public clouds differ with regard to performance, reliability, and cost. How can scientists optimize the trade-offs between these three factors and thus efficiently use the mixture of resources available to them? To answer this question, we introduce `ExPERT`, a general scheduling framework which finds Pareto-efficient job execution strategies in environments with mixtures of *unreliable and reliable* resources.

Today's grids and clouds reside in two extremes of the reliability and cost spectrum. Grid resources are often regarded as unreliable. Studies [1]–[3] and empirical data collected in the Failure Trace Archive [2] give strong evidence of the low long-term resource availability in traditional and desktop grids, with yearly resource availability averages of 70% or less. The constrained resource availability in grids is often a result of the sharing policy employed by each resource provider—for example, the grid at UW-Madison [4] employs preemptive fair-share policies [5], which vacate running tasks of external users when local users submit tasks. Commercial clouds, in contrast, have service-level agreements that guarantee resource availability averages of over 99%. Cost-wise, scientists often perceive grids as being free of charge, whereas clouds are pay-per-use. Accordingly, many grid users are now exploring the opportunity to migrate their scientific applications to commercial clouds for increased reliability [6]–[8], which could prove prohibitively expensive [7].

Scientific grid applications are often executed as Bags of Tasks (BoTs)—large-scale jobs comprised of hundreds to thousands of asynchronous tasks that must be completed to produce a single scientific result. Previous studies [9], [10] have shown that BoTs consistently account for over 90% of the multi-year workloads of some production grids. Thus, BoTs have been the *de facto* standard for executing jobs in unreliable grid environments over the past decade.

When executing BoTs in a grid environment, scientists *replicate* tasks. Replication increases the odds of timely task completion despite resource unreliability [11]–[15], but also wastes CPU cycles and energy, and incurs other system-wide costs [14] such as scheduler overload and delays to other users. It is difficult to select a *replication strategy* that yields the desired balance between the BoT response time (makespan) and the BoT execution cost. A wrong strategy can be expensive, increasing *both* makespan and cost. Although various heuristics were devised to pick a "good" replication strategy, our study is the first to focus on explicitly identifying Pareto-efficient strategies, that is, strategies that incur only the necessary cost and take no longer than necessary to execute a given task.

We envision a world in which BoTs are executed on whatever systems are best suited to the user's preferences at that time, be they grids, clouds, dedicated self-owned machines, or any combination thereof. This vision presents many optimization opportunities; optimizing the structure of the reliable+unreliable environment is only one of many examples. These opportunities can be exploited only when taking into account the individual preferences of each scientist. One scientist might want to obtain results by completing a BoT as quickly as possible, regardless of cost. Another might choose to minimize the cost and complete the BoT only on grid resources. Yet another scientist might try to complete work as soon as possible but under strict budget constraints (e.g., [16]). What all users share is a desire for efficient scheduling strategies.

Our main research goal is to determine *which strategies*

1530-2075/12 $26.00 © 2012 IEEE
DOI 10.1109/IPDPS.2012.25

are Pareto-efficient and which of them the user should pick. The following four questions will guide us in helping the user choose the best possible strategy. *What mixture of reliable and unreliable resources should be used? How many times should tasks be replicated on unreliable resources? What deadline should be set for those replicas? What is the proper timeout between submitting task instances?* Although Pareto-efficient strategies have been investigated before in different contexts [17]–[20], they are generally considered too computationally-intensive for online scheduling scenarios. However, we show here that even low-resolution searches for Pareto-efficient strategies benefit scheduling large numbers of tasks online.

Our first contribution is a model for task scheduling in mixed environments with varying reliability and cost (Sections II and III). Our second contribution is `ExPERT`, a framework for dynamic online selection of a Pareto-efficient scheduling strategy, which offers a wide spectrum of efficient strategies for different user makespan-cost trade-offs, leading to substantial savings in both (Section IV). We evaluate `ExPERT` through both simulations and experiments in real environments (Section V), and show (Section VI) that `ExPERT` can save substantial makespan and cost in comparison to scheduling strategies commonly used for workload scheduling in grids.

II. The Basic System Model

In this section we introduce the basic system model used throughout this work. We first build towards the concept of the Pareto frontier, then present the model for the system and the environment.

A. Terminology

A *t*ask is a small computational unit. A task *i*nstance is submitted to a resource. If the resource successfully performs the task, it returns a *r*esult. For a successful task instance, the *result turnaround time* is the time between submitting an instance and receiving a result. For a failed instance, this is ∞. A *BoT* is a set of asynchronous, independent tasks, forming a single logical computation. Users submit BoTs to be executed task-by-task. We divide BoT execution into the *throughput phase* and the *tail phase*, as depicted in Figure 1. The *remaining tasks* are tasks which have not yet returned a result. The *tail phase start time* (T_{tail}) occurs when there are fewer remaining tasks than available unreliable resources. A BoT is completed when each of its tasks has returned a result. The *makespan of a BoT* is the period elapsed from its submission to its completion. Similarly, the *tail phase makespan* is the period elapsed from T_{tail} until the completion of the BoT.

Replication is the submission of multiple *instances* of the same task, possibly overlapping in time. A task is complete when one of its instances returns a successful result. The

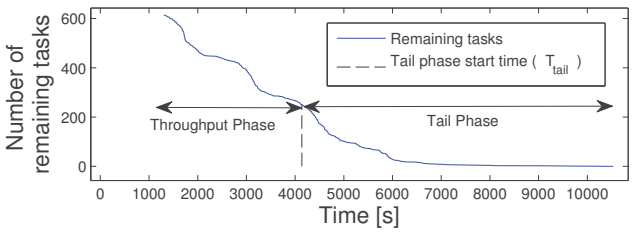


Figure 1: Remaining tasks over time during the throughput and tail phases. Input: Experiment 6 (Table V).

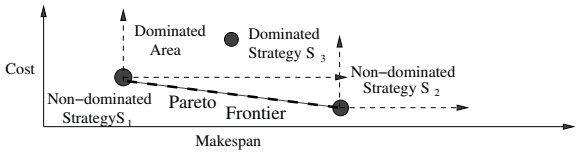


Figure 2: A Pareto frontier. Strategies S_1 and S_2 form the Pareto frontier. S_1 dominates S_3.

reliability of a resource pool is the probability that an instance submitted to that pool will return a result.

Cost is a user-defined price tag for performing a task, and may reflect monetary payments (e.g., for a cloud), environmental damage, or depletion of grid-user credentials. We ignore the costs of failed instances since it is difficult to justify charging for unobtained results.

The *user's scheduling system* (user scheduler) sends and replicates the user's tasks to the available resource pools. A user *strategy* is a set of input parameters indicating when, where, and how the user wants to send and replicate tasks.

The *performance metrics* are *cost per task* (the average cost of all BoT tasks) and makespan. A *user's utility function* is a function of the performance metrics of a strategy that quantifies the benefit perceived by the user when running the BoT. The user would like to optimize this function, for a given BoT and environment, when selecting a strategy. For example, a user who wants the cheapest strategy can use a utility function that only considers costs.

A strategy is *dominated* by another strategy if its performance is worse than or identical to the other for both metrics (cost and makespan) and strictly worse for at least one. A strategy that is not dominated by any other strategy is *Pareto-efficient*; the user cannot improve this strategy's makespan without paying more than its cost. As illustrated in Figure 2, several Pareto-efficient strategies may co-exist for a given *unreliable+reliable* system and workload (BoT). The *Pareto frontier* (or "Skyline operator" [21]) is the locus of all efficient strategies with respect to the searched strategy space. Any strategy that optimizes the user's utility function is Pareto-efficient. Furthermore, for any Pareto-efficient strategy, there exists a utility function that the strategy maximizes in the search space.

B. Model and Assumptions

We outline now the model and the assumptions for this work, first the environment, then the execution infrastructure. The assumptions are inspired by real–world user schedulers such as GridBoT [13], which are designed for CPU–bound BoTs that are not data bound.

Our model of the environment consists of two task queues. One queue is serviced by the *unreliable pool*, and the other is serviced by the *reliable pool*.

We characterize the reliable and unreliable pools in terms of speed, reliability, and effective size. Unreliable machines operate at various speeds; reliable machines are homoge–neous. (We assume they are of the same cloud instance type or belong to a homogeneous self–owned cluster. Thus, they are far more homogeneous than the unreliable machines.) Failures in the unreliable pool are abundant and unrelated across different domains [1]; reliable machines never fail (we justify the approximation by the large reliability difference between the unreliable and reliable pools). The reliable and unreliable pools have different *effective sizes* (number of resources that the user can concurrently use). We assume that effectively there are many more unreliable than reliable machines (typical effective sizes are hundreds of unreliable nodes and tens of reliable nodes), and thus we do not consider using only the reliable resources. Resources are charged as used, per charging period (one hour on EC2, one second on grids and self–owned machines).

We make no assumptions on task waiting time or on the unreliable system's scheduling policy, other than that both can be modeled statistically. Since we allow for loose connectivity between the scheduler and the hosts [15], it may be impossible to abort tasks, and the exact time of a task failure may not be known. A task which did not return its result by its deadline is considered failed. We assume the user has an overlay middleware that replaces malfunctioning hosts with new ones from the same pool. Our experiments show that such middleware can maintain an approximately constant number of unreliable resources when requesting up to 200 machines from a larger infrastructure.

III. The Scheduling Strategy Space

In this section we introduce our model for scheduling tasks with replication in an environment with mixed reli–ability, cost, and speed. The model generalizes state–of–the–art user strategies, e.g., of GridBoT users [13]. We focus on optimizing the tail phase makespan and cost by control–ling the tail phase scheduling strategy, for three reasons. First, in naive BOINC executions [15], the tail phase is an opportunity for improvement [22], as seen in Figure 1: the task return rate in the tail phase is low, while many resources are idle. Second, replication is inefficient during the throughput phase [23]. Third, setting the decision point after the throughput phase lets us base the optimization on the highly–relevant statistical data (e.g., of task turnaround times) collected during the throughput phase.

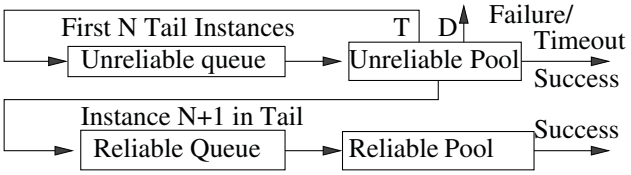


Figure 3: $NTDM_r$ task instance flow during throughput phase and tail phase. Reliable machines serve only instance $N+1$ during the tail phase (throughput phase instances are not counted). During the throughput phase, $T = D$, so there is no replication

During the throughput phase we use a "no replication" strategy, with a deadline of several times the *average task CPU time on the unreliable resource* (denoted by T_{ur} and estimated according to several random tasks). This dead–line length is a compromise between the time it takes to identify dysfunctional machines and the probability of task completion. A long deadline allows results to be accepted after a long time, but leads to long turnaround times. For the tail phase, we can consider strategies with deadlines set to the measured turnaround times. Deadlines much longer than T_{ur} are not interesting, because strategies with such deadlines are inefficient.

When the tail phase starts, all unreliable resources are occupied by instances of different tasks, and the queues are empty. From that point on, additional instances are enqueued by a scheduling process: first to the unreliable pool, then to the reliable one, as illustrated in Figure 3. This scheduling process, which we name $NTDM_r$, is controlled by four user parameters, N, T, D and M_r. Different strategies have different $NTDM_r$ values:

N is the maximal number of instances sent for each task to the unreliable system since the start of the tail phase. A last, $(N+1)^{th}$ instance is sent to the reliable system without a deadline, to ensure task completion. A user without access to a reliable environment is restricted to $N = \infty$ strategies. Increasing N improves the chance that the reliable instance will not be required, but increases the load on the unreliable pool. It also increases the probability of receiving and paying for more than one result per task.

D is a deadline for an instance, measured from its sub–mission to the system. Setting a large value for D improves the instance's chances to complete on time, but increases the time that elapses before the user becomes aware of failures. Short deadlines enable quick resubmission of failed tasks.

T is a timeout: the minimal waiting time before submit–ting another instance of the same task. Rather than having all instances submitted at the same time, each is submitted after a period T has passed from the previous instance submission, provided that no result has yet been returned.

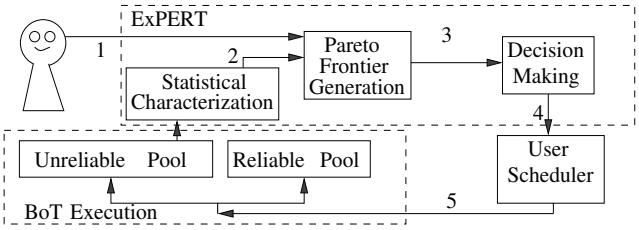


Figure 4: Flow of the ExPERT stages, with user intervention points. Numbered arrows indicate process steps.

T restricts resource consumption.

$\mathbf{M_r}$ is the ratio of the effective sizes of reliable and unreliable pools. It provides a user-defined upper bound on the number of concurrently used reliable resources. Small M_r values create long queues for the reliable pool. A long reliable queue may indirectly reduce costs by allowing unreliable instances to return a result and cancel the reliable instance before it is sent. We demonstrate M_r's contribution to the cost reduction of efficient strategies in Section VI.

The user's main goal is to choose values for N, T, D, and M_r, such that the resulting makespan and cost optimize a specific utility function. However, the user does not know the cost–makespan trade-off, or what parameter values would lead to a specific makespan or cost. To help the user choose these values, we introduce in the next section a framework for the selection of an efficient replication strategy.

IV. The ExPERT Framework

In this section we explain the design and use of the ExPERT scheduling framework. Our main design goal is to restrict the $NTDM_r$ space to Pareto-efficient strategies, from among which the user can then make an educated choice. To achieve this goal, ExPERT defines a scheduling process, which includes building a Pareto frontier of $NTDM_r$ strategies, out of which the best strategy for the user is chosen.

The ExPERT Scheduling Process: The $NTDM_r$ task instance flow is depicted in Figure 4. The user provides her parameters and, optionally, a utility function. ExPERT then statistically characterizes the workload and the unreliable system on the basis of historical data, analyzes a range of strategies, generates the Pareto frontier, and presents the user with makespan−and cost-efficient strategies. After either the user or ExPERT decides which strategy in the frontier to use, ExPERT passes the N, T, D, M_r input parameters of the chosen strategy to the user's scheduler, which then replicates tasks and submits them to the two resource queues.

The ExPERT framework is extensible in three ways. First, in Step 2 it allows for alternative methods of gathering and analyzing the system properties. Second, in Step 3 it allows for alternative algorithms for construction of the Pareto frontier. Third, in Step 4 it allows the user to employ any utility function which prefers lower makespans and costs:

Table I: User-defined parameters

Item	Definition
T_{ur}	Mean CPU time of a successful task instance on an unreliable machine
T_r	Task CPU time on a reliable machine
C_{ur}	Cents-per-second cost of unreliable machine
C_r	Cents-per-second cost of reliable machine
M_r^{max}	Maximal ratio of reliable machines to unreliable machines

using the Pareto frontier allows freedom of choice with regard to the utility function.

Traditionally, BoTs are executed through schedulers such as GridBoT [13], BOINC or Condor using a pre-set strategy, defined when the BoT is submitted. Though historical performance data has been used by others for resource exclusion [11] and for resource allocation adaptation [16], ExPERT is the first to use it to optimize general makespan and cost preferences. In addition, once the Pareto frontier is computed, it supplies the user with an understanding of the trade-offs available in the system, to be utilized in the future, possibly with different utility functions.

User Input: The user supplies ExPERT with data about mean CPU times (denoted T_r, T_{ur}), runtime costs in cents per second (denoted C_r, C_{ur}), and the reliable resource pool's effective size relative to the unreliable one (Table I). M_r^{max}, the upper bound of M_r, is derived from the unreliable pool's effective size, as well as from the number of self-owned machines, or from a restriction on the number of concurrent on-demand cloud instances (e.g., at most 20 concurrent instances for Amazon EC2 first-time users). Runtime costs might reflect monetary payments, energy waste, environmental damage, or other costs. For example, a user might set unreliable costs as zero, representing the grid as free of charge, or set it to account for power consumption. ExPERT uses this data to estimate the BoT's cost and makespan under different strategies, when it searches the strategy space.

Statistical Characterization: ExPERT statistically characterizes the workload and the unreliable system using $F(\cdot)$, the Cumulative Distribution Function (CDF) of *result turnaround time*. It also estimates the *effective size of the unreliable pool*, denoted as $\sharp ur$, by running iterations of the *ExPERT estimator* (described below) over the throughput phase until the estimated result rate matches the real result rate. The estimated $\sharp ur$ and $F(\cdot)$ are used to predict the makespan and cost of a given strategy and BoT. We describe here the estimation of $F(\cdot)$. The estimated $\sharp ur$ and $F(\cdot)$ are later used in step 3 to statistically predict the makespan and cost of applying a scheduling strategy to the execution of a given BoT.

$F(\cdot)$ effectively models many environmental, workload, and user-dependent factors. It is used to predict result turnaround time during the tail phase, so it is best estimated

in conditions that resemble those prevailing during this phase. The throughput phase supplies us with such data, but it can also be obtained from other sources. If the throughput phase is too short to collect enough data before the tail phase starts, public grid traces can be combined with statistical data about the workload to estimate the CDF.

The CDF is computed as follows:

$$F(t,t') = F_s(t)\gamma(t'). \quad (1)$$

Here t denotes instance turnaround time, and t' denotes instance sending time. $F_s(t)$ denotes the *CDF of successful task instances* (i.e., those which returned results). It can be directly computed from the turnaround times of results. $\gamma(t')$ denotes the *unreliable pool's reliability at time* t': the probability that an instance sent at time t' to the unreliable pool returns a result at all. $\gamma(t')$ is computed for disjoint sets of consecutively sent instances as the number of results received by the deadline, divided by the number of instances.

Because $F(\cdot)$ depends on t' through $\gamma(t')$, the CDF might change over time, necessitating a prediction model. ExPERT can either compute $\gamma(t')$ *offline* or estimate it *online*. The accuracy of the two models is compared in Section VI. In the offline model, $\gamma(t')$ is fully known (it is computed after all the results have returned). In the online model, $\gamma(t')$ is predicted according to information available at the decision making time T_{tail}. Depending on when the instance was sent, at time T_{tail} we might have full knowledge, partial knowledge, or no knowledge whether the instance will have returned a result by the time its deadline arrives. The timeline of the instance sending time t' is divided into three epochs as follows.

1) Full Knowledge Epoch: the instance was sent at time t' such that $t' < T_{tail} - D$. Instances sent during this first epoch that have not yet returned will not return anymore, so all the information about these tasks is known at time T_{tail}, in which the online reliability is evaluated. The online reliability model is identical to offline reliability during this epoch.
2) Partial Knowledge Epoch: $T_{tail} - D \leq t' < T_{tail}$. Instances sent during this second epoch that have not yet returned may still return. We use Equation 1 to approximate the probability that an instance sent at time t' will eventually finish. That is, we try to compute $\gamma(t')$ on the basis of the observable task success rate ($F_s(t)$). According to our model in Equation 1, $F(t,t')$ is separable. Hence, instead of computing $F_s(t)$ according to data of this second epoch to evaluate $F_{s_2}(t)$, we use $F_{s_1}(t)$, that is, the CDF of successful instances during the first epoch.
   Let $\hat{F}(t,t')$ denote $F(t,t')$ as was computed for instances sent at time t'. With the information known at time T_{tail}, the CDF is fully known ($F(t,t') = \hat{F}(t,t')$) for small values of t ($t \leq T_{tail} - t'$). However, for larger values of t, no information exists. As t' approaches T_{tail}, $\hat{F}(T_{tail} - t', t')$ becomes less accurate, because it relies on less data.
   We substitute the approximations $F_{s_1}(t)$ and $\hat{F}(t,t')$ in Equation 1 for the time t for which we have the most data ($t = T_{tail} - t'$):

$$\gamma(t') = \frac{F(T_{tail}-t',t')}{F_s(T_{tail}-t')} \approx \frac{\hat{F}(T_{tail}-t',t')}{F_{s_1}(T_{tail}-t')}. \quad (2)$$

   Due to the diminishing accuracy of the computation of $\hat{F}(T_{tail} - t', t')$, Equation 2 may result in fluctuating, unreasonable values, which need to be truncated. From below, we limit by the minimal historical value during the first epoch. From above we only limit it by 1 because resource exclusion [11] (that is, the mechanism of avoiding faulty hosts) might raise the reliability values above their maximal historical values.
3) Zero Knowledge Epoch: $t' \geq T_{tail}$, the instances have not yet been sent at the decision making time, and no result has yet returned. We use an average of the mean reliabilities during the Full Knowledge and the Partial Knowledge Epochs, thus incorporating old accurate data as well as updated, possibly inaccurate data. Our experiments indicate that an average of equal weights produces a good prediction for $\gamma(t')$ during this epoch.

Pareto Frontier Generation: ExPERT generates the Pareto frontier using data from the previous steps in two moves. First it samples the strategy space and analyzes the sampled strategies. Then it computes the Pareto frontier of the sampled strategies, from which the best strategy can be chosen. The sampling resolution is configurable, limited in range by the deadline used in the throughput phase. We found that focusing the resolution in the lower end of the range is more beneficial, as it accounts for the knee of the Pareto frontier, which improves with resolution.

The *ExPERT Estimator* estimates the mean makespan and cost of each sampled strategy through simulation. The *ExPERT Estimator* models $\sharp ur$ unreliable and $\lceil M_r \sharp ur \rceil$ reliable resources, each resource pool having a separate, infinite queue. For simplicity we assume the queues are FCFS: from each queue, tasks are submitted according to the order in which they entered the queue, unless they are canceled before they are submitted. If one instance of a task succeeds after another is enqueued but before it is sent, the other instance is canceled. If the other instance was already sent, it is *not* aborted. For each instance sent to the unreliable pool, a random number $x \in [0,1]$ is uniformly drawn. The instance turnaround time t solves the equation $F(t,t') = x$. If $t \geq D$, the instance is considered timed-out.

At each time-step the *ExPERT Estimator* first checks each running instance for success or timeout. Then, if a task has not yet returned a result, time T has already passed since its last instance was sent, and no instance of this task is currently enqueued, the *Estimator* enqueues one instance for this task. Finally, instances are allocated to machines.

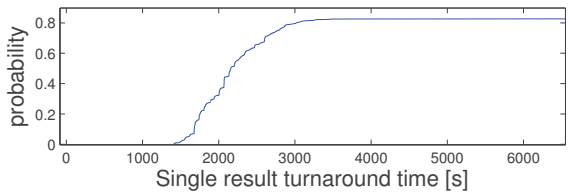


Figure 5: CDF of single result turnaround time. Input: Experiment 11 (Table V).

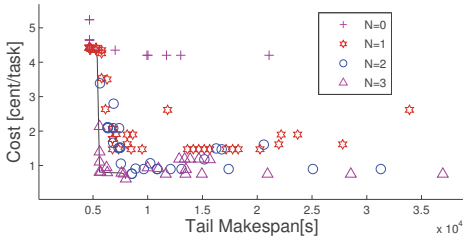


Figure 6: Pareto frontier and sampled strategies. Input: Experiment 11 (Table V).

ExPERT uses the average cost and makespan of several such estimations as expectation values of the real cost and makespan.

Once all the sampled strategies are analyzed, ExPERT produces the Pareto frontier by eliminating dominated strategies from the set of sampled strategies, such that only non–dominated points remain, as illustrated in Figure 2. Each point on the Pareto frontier represents a Pareto-efficient strategy. Under the rational assumption of monotonicity of the utility function, all strategies that may be the best within the sampled space for *any* utility function are included in the frontier. ExPERT uses a hierarchical approach, which resembles the s–Pareto frontier [20]: the strategies are first divided according to their N values, since different N values account for distinct separate conceptual solutions. Then ExPERT merges the different frontiers. The user's utility function is not explicitly required for frontier generation—the user may withhold information about his or her utility function, and only choose a strategy from the Pareto frontier after it is presented. Furthermore, once created, the same frontier can be used by different users with different utility functions.

Decision Making: After ExPERT generates the Pareto frontier, ExPERT chooses the best strategy for the user according to her utility function; otherwise, the user programs any other algorithm to choose the best strategy for her needs. We present an example of decision making for a scientific BoT, with a task turnaround time CDF as given in Figure 5 and user supplied parameters as listed in Table II.

We begin by showcasing the difficulty of selecting an appropriate scheduling strategy. Using an inefficient strategy (such as an $NTDM_r$ strategy that is *not* on the Pareto frontier) might waste a lot of time and money. For our example, Figure 6 displays only some of the sampled strategies and the resulting Pareto frontier (the depiction of the explored strategy space was diluted for clarity.) Here, using the Pareto frontier can save the user from paying an inefficient cost of $4\frac{cent}{task}$ using $N = 0$ (no replication), instead of an efficient cost of under $1\frac{cent}{task}$ (4 times better) when using $N = 3$. Furthermore, a user who chooses $N = 1$ and is willing to pay $2\frac{cent}{task}$ may obtain a poor makespan of over 25,000s (the top right–most hexagram symbol in Figure 6). In contrast, ExPERT recommends a strategy based on using $N = 3$, which leads to a makespan around 5,000s (5 times better) and a cost of under $1\frac{cent}{task}$ (the triangle symbol at the "knee" of the continuous curve in Figure 6).

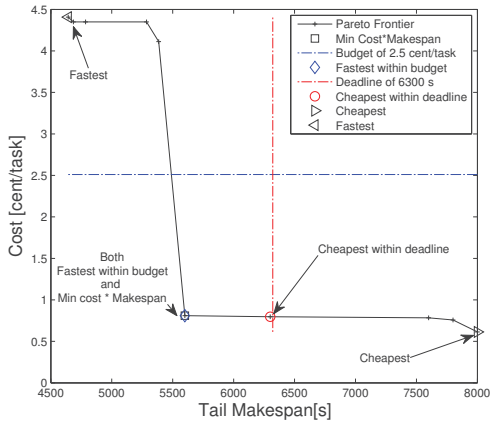


Figure 7: Pareto frontier and examples of best points for various user utility functions. Input: Experiment 11 (Table V).

We next illustrate how ExPERT assists the user's decision process. Figure 7 depicts the Pareto frontier in terms of cost and makespan. ExPERT marks the frontier for several strategies, which are best for some simple user preferences such as 'minimize tail phase makespan', 'minimize cost', 'minimize tail–phase–makespan × cost', and 'work within a budget' or 'finish in time'. If the user supplies ExPERT with a different utility function, ExPERT also finds the best strategy for it. A user who does not provide a utility function can choose one of the Pareto-efficient strategies presented at this stage. The Pareto frontier is discrete (we draw the connecting line for visual purposes only), so only the discrete points on it have attached input parameters. For a higher–density frontier, that is, a frontier that renders the connecting line in Figure 7, a higher–density sampling of the search space is required. However, even a low sampling resolution closely approaches the extreme strategies (the cheapest and the fastest).

The strategy is now chosen in terms of cost and makespan. To finalize the process, ExPERT presents the user with the parameters N, T, D and M_r, which define the chosen

Table II: Values for user-defined parameters

Item	Value
T_{ur}	Mean CPU time of successful instances on unreliable pool (2,066 seconds for Experiment 11)
T_r	For real/simulated experiment comparison: mean CPU time over reliable instances. Otherwise: T_{ur}.
C_{ur}	$\frac{1}{3600}\frac{cent}{second} = 10\frac{cent}{KWH} \cdot 100W$
C_r	$\frac{34}{3600}\frac{cent}{second}$: EC2's m1.large on-demand rate

Table III: Workloads with T, D strategy parameters and throughput phase statistics. WL denotes Workload index. WM is an execution environment from Table IV.

WL	♯Tasks	T[s]	D[s]	CPU time on WM[s]		
				Average	Min.	Max.
WL1	820	2,500	4,000	1,597	1,019	3,558
WL2	820	1,700	4,000	1,597	1,019	3,558
WL3	3276	5,000	8,000	1,911	1,484	6,435
WL4	3276	3,000	5,000	2,232	1,643	4,517
WL5	615	4,000	6,000	878	1,571	4,947
WL6	615	4,000	4,000	729	1,512	3,534
WL7	615	2,500	4,000	987	1,542	3,250

strategy. Those parameters are passed to the user's scheduler and are used to run the user's tasks.

V. The Experimental Setup

In this section we present our experimental setup. To evaluate ExPERT in a variety of scenarios yet within our budget, we ran a series of real-world experiments and augmented the results with simulated experiments. The simulator was created by re-using a prototype implementation of the *ExPERT Estimator*; our simulations can be seen therefore as emulations of the ExPERT process. We validated the simulator's accuracy by comparing simulation results with results obtained through real-world experiments performed on different combinations of unreliable and reliable pools, including grids, self-owned machines, and Amazon EC2. To validate the simulator, we used various BoTs which perform genetic linkage analysis, a statistical method used by geneticists to determine the location of disease-related mutations on the chromosome. The BoTs, which are a characteristic workload (real full applications) for the superlink-online system [24], are characterized in Table III. In pure simulation experiments we used the CDF shown in Figure 5.

Experimental Environments: The real-world experiments were conducted using GridBoT [13], which provides a unified front-end to multiple grids and clouds. GridBoT interprets a language for encoding scheduling and replication strategies on the basis of run-time data, to simultaneously execute the BoTs in multiple pools. GridBoT relies on BOINC, so it is based on weak connectivity.

To implement the limit to the CPU time consumed by a task instance, we used the BOINC parameter `rsc_fpops_bound`, which poses a limitation on the number of flops a host may dedicate to any a task instance. Since this parameter only approximates the limit, we manually verified that task instances never continued beyond D.

Table IV: Real resource pools used in our experiments

Reliable	Properties
Tech	20 self-owned CPUs in the Technion
EC2	20 m1.large Amazon EC2 cloud instances
Unreliable	**Properties**
WM	UW-Madison Condor pool. Utilizes preemption. http://www.cs.wisc.edu/condor/uwcs
OSG	Open Science Grid. Does not preempt. http://www.opensciencegrid.org
OSG+WM	Combined pool, half ♯ur from each
WM+EC2	Combined pool, 20 EC2 + 200 WM
WM+Tech	Combined pool, 20 Tech + 200 WM

The simulation-based experiments used the same discrete event-based *ExPERT Estimator* we developed for building the Pareto frontier. Although we considered using a grid simulator [25]–[27], ultimately we decided to build our own simulation environment. Our simulations are specifically tailored for running ExPERT and have a simple, trace-based setup. More importantly, as far as we know, no other simulator has been validated for the scheduling strategies and environments investigated in this work. For comparison, we augmented the $NTDM_r$ strategies already implemented in the *Estimator* with several static strategies described below.

The user-specified parameters used in our experiments are summarized in Table II. To estimate C_{ur} we used the characteristic power difference between an active and idle state according to AMD's ACP metric [28]. We multiplied those power differences for Opteron processors [28] by two, to allow for cooling system power, reaching a range of 52W–157W; hence we use 100W here.

The resource pools are detailed in Table IV. Each experiment used one unreliable resource combination (one row) and at most one reliable resource. Experiments 1-6 used old resource exclusion data, thus choosing more reliable machines from the unreliable pools. In experiments 7-13 this data was deleted at the beginning of each experiment, thus allowing any machine in the unreliable pool to serve the BoT.

Static Scheduling Strategies: Without a tool such as ExPERT, users (e.g., GridBoT users) have resorted to static strategies. A static strategy is pre-set before the BoT starts, and does not require further computations during the BoT's run-time. Unless otherwise stated, during the throughput phase these strategies are "no replication" ($N = \infty, T = D = 4T_{ur}$) and the reliable pool is idle. Although some of these strategies are $NTDM_r$ strategies, they are not necessarily Pareto-efficient. We compare them to Pareto efficient strategies found by ExPERT in the next section. The static strategies are:

AR: All to Reliable: use only reliable machines for the duration of the BoT. This is a fast strategy when there

are many fast reliable machines and the reliability of the unreliable machines is low.

TRR: all Tail Replicated to Reliable: at T_{tail}, replicate all remaining tasks to the reliable pool. This is an $NTDM_r$ strategy ($N = 0$,$T = 0$,$Mr = M_r^{max}$).

TR: all Tail to Reliable: at T_{tail}, enqueue every *timed out* tail task to the reliable pool. This is an $NTDM_r$ strategy ($N = 0, T = D, Mr = M_r^{max}$).

AUR: All to UnReliable, no replication: use the default throughput phase strategy during the tail phase. This is the cheapest option for a cheap unreliable system. This is an $NTDM_r$ strategy ($N = \infty, T = D$).

B=7.5: Budget of 7.5$ for a BoT of 150 tasks ($\frac{2}{3}\frac{cent}{task}$): replicate all remaining tasks on the reliable pool once the estimated cost of the replication is within the remaining budget. Until then, use the default throughput phase strategy.

CN∞: Combine resources, no replication: deploy tasks from the unreliable queue on the reliable pool if the unreliable pool is fully utilized. This is a common way of using the cloud, supplementing self-owned machines with cloud machines when the regular machines are busy.

CN1T0: Combine resources, replicate at tail with $N = 1$, $T = 0$: utilize all resources only during the throughput phase. At T_{tail}, replicate: for each remaining task, enqueue a reliable instance.

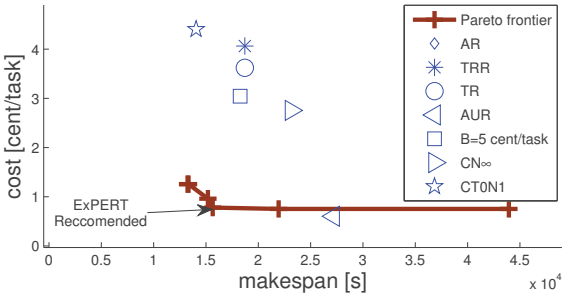
VI. The Experimental Results

We begin by evaluating $NTDM_r$ Pareto frontiers by comparing them to the static strategies introduced in Section V. We proceed to demonstrate the importance of M_r as a strategy parameter in Section VI. We then validate the *ExPERT Estimator* logic in Section VI and discuss the time it takes to run ExPERT in Section VI.

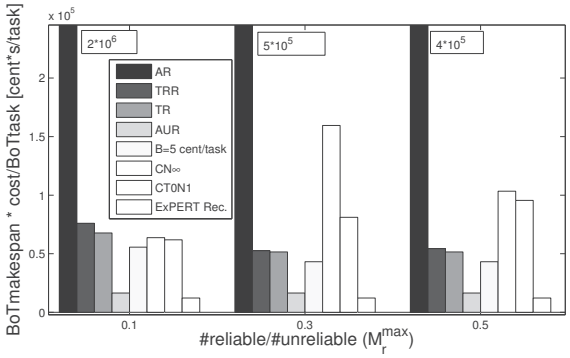
ExPERT vs. Static Scheduling Strategies: To evaluate the benefits of using $NTDM_r$ Pareto-efficient strategies, we compare them with the seven static scheduling strategies. The comparison is performed for a BoT of 150 tasks, with 50 machines in the unreliable resource pool. The Pareto frontier is obtained by sampling the strategy space in the range $N = 0 \ldots 3$, $M_r = 0.02 \ldots M_r^{max}$, and $0 \leq T \leq D \leq 4T_{ur}$. T, D were evenly sampled within their range at 5 different values each. M_r was sampled by at most 7 values, listed in Figure 9.

We first compare the makespan and cost of the static strategies to the Pareto frontier on a system where $M_r^{max} = 0.1$, and depict the results in Figure 8a. **The Pareto frontier found by ExPERT dominates all the tested static strategies except AUR**; that is, for *any* utility function, for each tested static strategy except AUR, ExPERT recommends at least one $NTDM_r$ strategy that improves both metrics. For example, ExPERT finds *several* strategies that dominate the commonly-used *CN∞* strategy. One such strategy is:

ExPERT recommended ($N = 3, T = T_{ur}, D = 2T_{ur}$,$M_r = 0.02$): send $N = 3$ instances to the unreliable pool during the tail phase, with timeout set to occur after twice the average task time ($D = 2T_{ur}$). Send each subsequent instance after the average task time ($T = T_{ur}$) from the sending of the prior instance had passed. Use only one ($\sharp ur = 50$, $50 \times M_r = 1$) reliable machine at a time.



(a) Performance of strategies on the Pareto frontier vs. that of static strategies, for $M_r^{max} = 0.1$. Strategy AR at (makespan around 70,000s, cost=22$\frac{cent}{task}$) is not shown.



(b) Makespan-cost product for static and *ExPERT recommended* strategies, for $M_r^{max} = 0.1, 0.3, 0.5$. Bars for strategy AR are truncated; their height appears beside them. Smaller values are better.

Figure 8: Static strategies compared to Pareto-efficient $NTDM_r$ strategies. Input: Experiment 11.

This strategy, which is located in Figure 8a at the "knee" of the Pareto frontier, yields a makespan of 15,640s for the cost of $0.78\frac{cent}{task}$, **cutting 72% of *CN∞*'s cost and 33% of its makespan.** This strategy does not dominate AUR, by definition the cheapest strategy. Nonetheless, several strategies found by ExPERT on the Pareto frontier lead to much better makespan than AUR, with only a small increase in cost.

The dominance of the $NTDM_r$ Pareto frontier demonstrates the power of Pareto-efficient scheduling over static strategies. The frontier's dominance is not a direct consequence of the way it is built, which only guarantees that it will dominate the $NTDM_r$ strategies in the sampled space. **The fact that the $NTDM_r$ Pareto frontier dominates the static strategies implies that $NTDM_r$ is a good scheduling model: the efficient strategies the user looks**

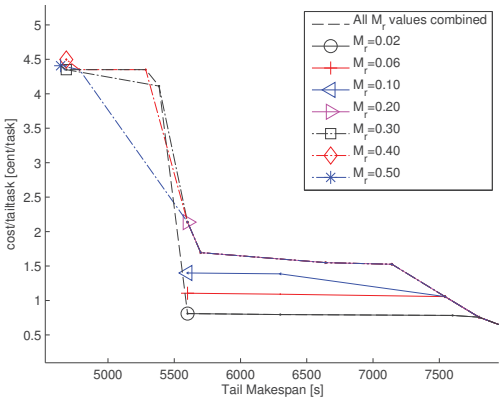


Figure 9: Pareto frontiers obtained for various M_r values. The topmost efficient point of each Pareto frontier is highlighted. Pareto frontiers of high M_r values have a wider makespan range. Low M_r values yield lower costs.

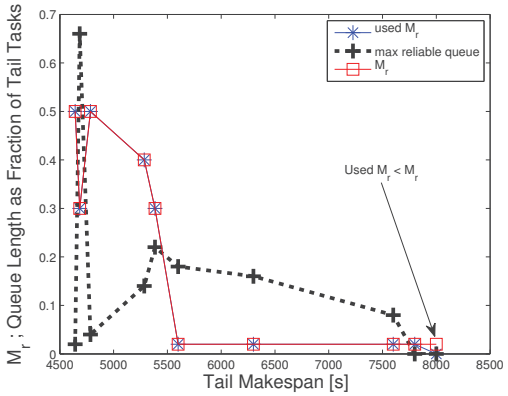


Figure 10: Reliable pool use by efficient strategies.

for can be expressed as points in the sampled $NTDM_r$ space.

Next, we focus on the performance of the strategies in terms of a specific utility function: *minimize tail-phase-makespan* $\times$ *cost per task*. We compare the utility obtained by the user when the scheduling strategy is ExPERT *recommended* or one of the seven static scheduling strategies. Figure 8b depicts the results of this comparison. ExPERT *recommended* is 25% better than the second-best performer, AUR, 72%–78% better than the third-best performer, and several orders of magnitude better than the worst performer, AR. We conclude that ExPERT *recommended* delivers significantly better utility than *all* the tested static strategies and outperforms (dominates) all these strategies except AUR.

Each static strategy might be tailored for a special scenario and a utility function. However, as Figure 8b demonstrates, using ExPERT to search the strategy space for that special scenario will provide the user with the best strategy in the search space, for a small computational cost (see below).

Impact of M_r ExPERT's Performance: M_r provides a bound on the number of concurrently used reliable resources (see Section III). We now demonstrate the benefit of elasticity, justifying the model decision which allows M_r to be a scheduling strategy parameter rather than a system constant. We consider $M_r = 0.02 \ldots 0.50$, which means that reliable resources are less than 50% of the resources available to the user.

First we demonstrate why users need to be able to set M_r as a parameter of their scheduling strategy. To this end, we compare the Pareto frontiers created by fixing M_r; we depict in Figure 9 seven such frontiers. As shown by the figure, high M_r values allow a wide range of makespan values overall, but low M_r values can only lead to relatively longer makespans. For example, the Pareto frontier for $M_r = 0.02$ starts at a tail makespan of over 5,500s, which is 25% larger than the makespans achievable when $M_r \geq 0.30$. We also observe that, for the same achieved makespan, lower M_r values lead in general to lower cost. We conclude that **to find Pareto-efficient $NTDM_r$ strategies, M_r should *not* be fixed in advance, but set in accordance with the desired makespan.**

We investigate next the impact of M_r in the execution of the BoT on the resources provided by the reliable pool. For each Pareto-efficient strategy operating in this environment, we compare three operational metrics: the strategy parameter M_r, the maximal number of reliable resources used during the BoT's run (denoted *used* M_r), and the maximal size of the reliable queue built during the run. Figure 10 depicts the results of this comparison. We find that for most Pareto-efficient strategies, the number of used resources from the reliable pool, *used* M_r, is equal to the number of resources set through the strategy parameter, M_r. This is because, during the BoT's tail phase, tasks sometimes wait in the queue to the reliable pool, as seen in Figure 10: the maximal length of the reliable queue is almost never zero; that is, the queue is almost always used. The right-most point on the M_r and *used* M_r curves, for which the values of M_r and *used* M_r are different, is the exception. We explain this by an intrinsic load-balancing property of the $NDTM_r$ systems: when the reliable pool queue is long, slow unreliable instances return results before the reliable instance is sent, which leads to the reliable instance being canceled and its cost being spared.

Simulator Validation: We conducted 13 large-scale experiments to validate the simulator and the ExPERT *Estimator*. In each experiment, we applied a single strategy to specific workload and resource pools. Since the simulations include a random component, we ensured statistical confidence by comparing the performance metrics (tail phase makespan, cost per task) of each real experiment with mean values of 10 simulated experiments. We compared real and simulated performance metrics for both the offline and the online models (defined in Section IV). The experiments are

Table V: Experimental results. WL denotes workload according to Table III. N is the $NTDM_r$ parameter. $\sharp ur$ is an estimate for the effective size of the unreliable pool. ur and r denote choice of pools according to Table IV. γ denotes the average reliability of the unreliable pool. RI denotes the number of task instances sent to the reliable pool. TMS and C denote tail phase makespan and cost per task. ΔTMS and ΔC denote deviation of simulated values from real ones. The strategy in Experiment 5 is *CN∞: Combine resources, no replication* (which is not an $NTDM_r$ strategy).

Experiment Parameters						Measured in Real Experiment				Simulated Experiment Deviation			
										Offline [%]		Online [%]	
No.	WL	N	$\sharp ur$	ur	r	γ	RI	TMS[s]	C$\left[\frac{cent}{task}\right]$	$\frac{\Delta TMS}{TMS}$	$\frac{\Delta C}{C}$	$\frac{\Delta TMS}{TMS}$	$\frac{\Delta C}{C}$
1	WL1	0	202	WM	Tech	0.995	50	6,908	1.60	8	3	35	33
2	WL1	2	199	WM	Tech	0.983	0	3,704	39	21	-4	8	-4
3	WL6	∞	200+20	WM+Tech	–	0.981	0	6,005	41	1	-4	4	-4
4	WL3	0	206	WM	Tech	0.974	49	10,487	1.10	2	2	-56	-32
5	WL6	∞	200+20	WM+EC2	–	0.970	41	6,113	1.48	37	-2	29	-2
6	WL5	∞	201	WM	–	0.942	0	6,394	0.42	3	-4	-40	-4
7	WL1	0	208	WM	Tech	0.864	77	10,130	2.38	3	2	32	26
8	WL2	1	208	WM	Tech	0.857	16	4,162	0.88	19	15	-37	-10
9	WL1	0	251	OSG+WM	Tech	0.853	108	14,029	3.28	7	0	-1	-4
10	WL7	0	208	WM	EC2	0.844	118	11,761	3.67	-14	-35	-7	-28
11	WL1	0	200	OSG	Tech	0.827	89	11,656	2.86	8	1	-7	-7
12	WL1	0	200	WM	Tech	0.788	107	12,869	3.09	-9	-13	-2	-5
13	WL4	0	204	WM	Tech	0.746	100	20,239	1.54	-3	-7	-7	-10
Average of absolute values						0.894	58	9,574	1.78	10	7	20	13

listed by decreasing order of average reliability in Table V.

On average, performance metrics of the offline simulations, which use full knowledge of the unreliable pool's reliability $\gamma(t')$, deviate from real experimental values by 7% and 10% for cost and tail phase makespan, respectively. The on-line simulations, which extrapolate $\gamma(t')$ during the tail phase, deviated from real experimental values by twice as much.

We identify four main causes for these deviations. First, the simulator provides expectation values of performance metrics. In contrast, a real experiment is a single, unreproducible sample. When a large number of tasks are replicated during the tail phase, the performance metrics tend to be close to the mean values of the simulated experiments. When the opposite occurs, for example in Experiment 2, where only four very long instances were sent after T_{tail}, the makespan observed in the real environment is further from the offline simulation. Second, the simulator assumes $F_s(t)$ does not depend on t' and attributes all CDF changes to $\gamma(t')$. However, in real experiments $F_s(t)$ does depend on t', due to resource exclusion [11] policies and a varying task length distribution. Third, ExPERT assumes it is never informed of failures before the deadline D. In real experiments, some machines do inform about failures and are replaced. Fourth, in real experiments, the effective size of the unreliable pool is variable and hard to measure. Hence, T_{tail} is detected when there are more free hosts than remaining tasks. The tasks remaining at this time are denoted *tail tasks*. This may be a transient state, before the actual start of the tail phase. In simulated experiments, the number of machines is fixed. T_{tail} is detected when the number of remaining tasks equals the number of tail tasks in the real experiment.

ExPERT Runtime The computational cost of running our ExPERT prototype, in the resolution used throughout this paper, is several minutes to sample the strategy space and analyze it, on an Intel(R) Core(TM)2 Duo CPU P8400 @ 2.26GHz. The space sampling is composed of dozens of single strategy simulations, each lasting several seconds. We consider a runtime in the order of minutes, appropriate for BoTs of hundreds of tasks that are the focus of this work. ExPERT's runtime may be further shortened at the expense of accuracy, by reducing the number of random repetitions from over 10 to just 1. Similarly, flexibility may be traded with time by changing the resolution in which the search space is sampled. Gradually building the Pareto frontier using evolutionary multi-objective optimization algorithms can also reduce ExPERT's runtime.

VII. Related Work

Much research on replication algorithms relied on the assumption that computation is free of charge [11], [14], [29]–[32] and limited only by its effect on load and makespan, whereas we explicitly consider execution costs. Dobber, van der Mei, and Koole [30] have created an on-the-fly criterion for choosing between immediate replication and dynamic load balancing. Casanova [14] has shown the impact of simple replication policies on resource waste and fairness. Kondo, Chien, and Casanova [11] have combined replication with resource exclusion. Cirne et al. [31] and Silva et al. [32] have analyzed immediate replication with no deadline for perfectly reliable heterogeneous machines. Wingstrom and Casanova [29] assumed a specific distribution of task failures and used it to maximize the probability of a whole BoT to finish, by choosing replication candidates. In contrast, we optimize cost and time simultaneously.

Bi-objective time-related problems were also analyzed in task scheduling. Vydyanathan et al. [33] aim to minimize latency while meeting strict throughput requirements using

replication, subject to a certain amount of resource waste, in terms of the number of occupied processors. They [34] also aim to maximize the throughput while meeting latency constraints, as do Agrawal et al. [35] for linear task graphs. Our work optimizes one time related and one monetary objective for BoTs.

The concept of utility functions as the target of the optimization process has also received attention. Buyya et al. [36] researched economic mechanisms for setting grid computation costs, for several utility functions. One of their estimation methods is Pareto-efficiency. Ding et al. [37] aim to minimize the utility function of the energy-delay product on a multi-CPU machine, by using a helper thread which collects statistics and determines a deployment strategy. Lee, Subrata and Zomaya [38] aim to minimize both grid resource use and makespan for a workflow application, by giving both an equal weight. Benoit et al. [39] assumed a linear risk model for machine unavailability on homogeneous remote machines, and considered overhead and operational costs. Our work allows for both a general user function and a general probability distribution of task success. Andrzejak, Kondo, and Anderson [40] controlled reliable and unreliable pool sizes in a combined pool to Pareto-optimize cost and availability for Web services.

Pareto frontier approximations were previously used in scheduling for the makespan and reliability objectives, but not for cost, by Dongarra et al. [41], who scheduled task graphs, and by Saule and Trystram [42], and by Jeannot et al. [43].

Ramírez-Alcaraz et al. [44] evaluate scheduling heuristics and optimize a combined objective for parallel jobs, because they assess that computing a Pareto frontier in a GRID environment is too slow. However, approximating the Pareto frontier for the cases we demonstrated here using ExPERT takes only minutes, which we do not consider "too slow" for a BoT that runs for hours.

Oprescu and Kielmann [16] learn the run-time CDF online from the execution of the same BoT, as we do. However, they do not deal with reliability, since they use only clouds, and they utilize a heuristic to minimize makespan for a given budget. In contrast, our approach provides the client with full flexibility of choice, without forcing the choice of budget first, and is valid for grids, too, where reliability is an issue.

Pareto frontiers were also used to concurrently optimizing the same objective for different users, to achieve socially efficient scheduling and resource management [45], [46]. Zhao et al. [47] design a market for BoTs, aiming to efficiently optimize social welfare under agent budget constraints. Our work focuses on multiple objectives of the same user.

VIII. Conclusion

We addressed one of the main problems facing scientists who rely on Bags-of-Tasks (BoTs) in mixtures of computational environments such as grids and clouds: the lack of tools for selecting Pareto-efficient scheduling strategies for general user-defined utility functions. For any user-provided utility function, ExPERT finds the best strategy in a large, sampled strategy space. ExPERT can achieve a 72% cost reduction and a 33% shorter makespan compared with commonly-used static scheduling strategies. For a utility function of *makespan* $\times$ *cost*, ExPERT provided a strategy which was 25% better than the second-best, and 72–78% better than the third best strategy. These improvements stem from ExPERT's ability to explore a large strategy space under minimal user guidance, and to automatically adapt to the varying reliability, cost, and speed of resources. They also show that the $NTDM_r$ strategy space is large enough to provide considerable flexibility in both makespan and cost. ExPERT's predictive accuracy has been verified through experiments on real grids and a real cloud. The Pareto frontier created by ExPERT provides users with an understanding of the cost-makespan trade-offs of executing their BoTs.

IX. Acknowledgments

We thank Amazon for a grant that allowed us to experiment with EC2 free of charge, Miron Livny for access to the OSG and UW-Madison grids, and Muli Ben-Yehuda for fruitful discussions. The work was partially supported by the Technion Hasso Plattner Center, by the STW/NWO Veni grant @larGe (11881), and by the European Community's Seventh Framework Programme (ICT-2009-C) under grant agreement number 270833 (DATA SIM).

References

[1] A. Iosup, M. Jan, O. O. Sonmez, and D. H. J. Epema, "On the dynamic resource availability in grids," in *GRID*, 2007.

[2] D. Kondo, B. Javadi, A. Iosup, and D. H. J. Epema, "The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems," in *CCGRID*, 2010.

[3] D. Kondo, G. Fedak, F. Cappello, A. A. Chien, and H. Casanova, "Characterizing resource availability in enterprise desktop grids," *Future Generation Com. Sys.*, vol. 23(7).

[4] "UW-Madison CS Dept. Condor pool," Website, http://www.cs.wisc.edu/condor/uwcs/.

[5] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the Condor experience." *Concurrency - Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[6] K. R. Jackson, L. Ramakrishnan, K. J. Runge, and R. C. Thomas, "Seeking supernovae in the clouds: a performance study," in *HPDC*, 2010, pp. 421–429.

[7] G. Singh, M.-H. Su, K. Vahi, E. Deelman, B. Berriman, J. Good, D. S. Katz, and G. Mehta, "Workflow task clustering for best effort systems with pegasus," in *MG '08*.

[8] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Trans. on Parallel and Distrib. Sys.*, vol. 22, 2011.

[9] A. Iosup, C. Dumitrescu, D. H. J. Epema, H. Li, and L. Wolters, "How are real grids used? The analysis of four grid traces and its implications," in *GRID*, 2006.

[10] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. J. Epema, "The Grid Workloads Archive," *Future Generation Comp. Syst.*, vol. 24, no. 7, 2008.

[11] D. Kondo, A. A. Chien, and H. Casanova, "Resource management for rapid application turnaround on enterprise desktop grids," in *SC'04*, 2004.

[12] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *OSDI'08*, 2008.

[13] M. Silberstein, A. Sharov, D. Geiger, and A. Schuster, "Gridbot: Execution of bags of tasks in multiple grids," in *SC'09*.

[14] H. Casanova, "On the harmfulness of redundant batch requests," in *HPDC*, 2006, pp. 255–266.

[15] D. P. Anderson, E. Korpela, and R. Walton, "High-performance task distribution for volunteer computing," in *e-Science*, 2005, pp. 196–203.

[16] A.-M. Oprescu and T. Kielmann, "Bag-of-tasks scheduling under budget constraints," in *CloudCom*, 2010.

[17] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*, 1st ed. John Wiley and Sons, 2001.

[18] A. Messac, A. Ismail-Yahaya, and C. A. Mattson, "The normalized normal constraint method for generating the Pareto frontier," *Structural and Multidisciplinary Optimization*, vol. 25(2), 2003.

[19] H. A. Abbass, R. Sarker, and C. Newton, "PDE: A Pareto-frontier differential evolution approach for multi-objective optimization problems," in *CEC*, 2001.

[20] C. A. Mattson and A. Messac, "Pareto frontier based concept selection under uncertainty, with visualization," *Optimization and Engineering*, vol. 6(1), 2005.

[21] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *ICDE*, 2001, pp. 421–430.

[22] M. Silberstein, "Building online domain-specific computing service over non-dedicated grid and cloud resources: Superlink-online experience," in *CCGRID '11*, 2011.

[23] G. D. Ghare and S. T. Leutenegger, "Improving speedup and response times by replicating parallel programs on a snow," in *JSSPP '04*.

[24] M. Silberstein, A. Tzemach, N. Dovgolevsky, M. Fishelson, A. Schuster, and D. Geiger, "On-line system for faster linkage analysis via parallel execution on thousands of personal computers," *Amer. J. of Human Genetics*, vol. 78(6), 2006.

[25] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: a generic framework for large-scale distributed experiments," in *10th IEEE International Conference on Computer Modeling and Simulation*, Mar. 2008.

[26] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14(13).

[27] A. Iosup, O. Sonmez, and D. Epema, "Dgsim: Comparing grid resource management architectures through trace-based simulation," in *Euro-Par '08*.

[28] AMD, "ACP — the truth about power consumption starts here," white paper, 2007, http://www.amd.com/us/Documents/43761C_ACP_WP_EE.pdf.

[29] J. Wingstrom and H. Casanova, "Probabilistic allocation of tasks on desktop grids," in *IPDPS*, 2008.

[30] M. Dobber, R. D. van der Mei, and G. Koole, "Dynamic load balancing and job replication in a global-scale grid environment: A comparison," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 2, 2009.

[31] W. Cirne, F. Brasileiro, D. Paranhos, L. F. W. Góes, and W. Voorsluys, "On the efficacy, efficiency and emergent behavior of task replication in large distributed systems," *Parallel Comput.*, vol. 33, no. 3, 2007.

[32] D. P. D. Silva, W. Cirne, F. V. Brasileiro, and C. Grande, "Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids," in *Euro-Par*, 2003.

[33] N. Vydyanathan, U. Catalyurek, T. Kurc, J. Saltz, and P. Sadayappan, "Toward optimizing latency under throughput constraints for application workflows on clusters," in *Euro-Par*, 2007.

[34] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Sadayappan, and J. Saltz, "A duplication based algorithm for optimizing latency under throughput constraints for streaming workflows," in *Proceedings of the 2008 37th International Conference on Parallel Processing*, ser. ICPP '08, 2008.

[35] K. Agrawal, A. Benoit, L. Magnan, and Y. Robert, "Scheduling algorithms for linear workflow optimization," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, 2010, pp. 1 –12.

[36] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic models for resource management and scheduling in grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14(13-15), pp. 1507–1542, 2002.

[37] Y. Ding, M. Kandemir, P. Raghavan, and M. J. Irwin, "A helper thread based EDP reduction scheme for adapting application execution in cmps," in *IPDPS*, 2008.

[38] Y. C. Lee, R. Subrata, and A. Y. Zomaya, "On the performance of a dual-objective optimization model for workflow applications on grid platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 9, pp. 1273–1284, 2009.

[39] A. Benoit, Y. Robert, A. L. Rosenberg, and F. Vivien, "Static strategies for worksharing with unrecoverable interruptions," in *IPDPS*, 2009.

[40] A. Andrzejak, D. Kondo, and D. P. Anderson, "Exploiting non-dedicated resources for cloud computing," in *NOMS'10*.

[41] J. J. Dongarra, E. Jeannot, E. Saule, and Z. Shi, "Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems," in *Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2007.

[42] E. Saule and D. Trystram, "Analyzing scheduling with transient failures," *Inf. Process. Lett.*, vol. 109, pp. 539–542, May 2009.

[43] E. Jeannot, E. Saule, and D. Trystram, "Bi-objective approximation scheme for makespan and reliability optimization on uniform parallel machines," in *Euro-Par 2008*, ser. Lecture Notes in Computer Science, 2008, vol. 5168, ch. 94, pp. 877–886.

[44] J. M. Ramírez-Alcaraz, A. Tchernykh, R. Yahyapour, U. Schwiegelshohn, A. Quezada-Pina, J. L. González-García, and A. Hirales-Carbajal, "Job allocation strategies with user run time estimates for online scheduling in hierarchical grids," *J. Grid Comput.*, vol. 9, pp. 95–116, Mar. 2011.

[45] D. Cordeiro, P.-F. Dutot, G. Mounié, and D. Trystra, "Tight analysis of relaxed multi-organization scheduling algorithms," in *IEEE Int'l Parallel & Distributed Processing Symposium (IPDPS)*, 2011.

[46] R. Chen and H. Li, "The research of grid resource scheduling mechanism based on pareto optimality," in *Software Engineering (WCSE), 2010 Second World Congress on*, 2010.

[47] H. Zhao, X. Liu, and X. Li, "Hypergraph-based task-bundle scheduling towards efficiency and fairness in heterogeneous distributed systems," in *IEEE Int'l Parallel & Distributed Processing Symposium (IPDPS)*, 2010.