

# ExPERT: Pareto-Efficient Task Replication on Grids and Clouds

Orna Agmon Ben-Yehuda  
Technion–Israel Institute of  
Technology  
Haifa, Israel  
ladypine@cs.technion.ac.il

Assaf Schuster  
Technion–Israel Institute of  
Technology  
Haifa, Israel  
assaf@cs.technion.ac.il

Artyom Sharov  
Technion–Israel Institute of  
Technology  
Haifa, Israel  
sharov@cs.technion.ac.il

Mark Silberstein  
Technion–Israel Institute of  
Technology  
Haifa, Israel  
marks@cs.technion.ac.il

Alexandru Iosup  
PDS Group,  
TU Delft  
The Netherlands  
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, in this work 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.

## Categories and Subject Descriptors

C.4 [Computer Systems Organization]: PERFORMANCE OF SYSTEMS—*distributed computing*

## General Terms

bags-of-tasks, cloud computing, grid

## 1. 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. The advantages of this opportunity,

however, may be canceled out by additional 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 resources available to them? To answer this question, we introduce ExPERT, a general scheduling framework which finds Pareto-efficient strategies for executing jobs in mixtures of *unreliable and reliable* environments.

Today’s grids and clouds reside in two extremes of the reliability and cost spectra. Grid resources are often regarded as unreliable; we have advocated in the past [19] that, for production grids, improving reliability may be as important as improving performance. Studies [19, 25, 26] and empirical data collected in the Failure Trace Archive [26] give strong evidence of the low long-term resource availability in traditional and desktop grids, with yearly resource availability averages of 70% or less being common. 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 [37] employs preemptive fair-share policies [35], which vacate from resources the running tasks of external users when the local users submit their tasks. Commercial clouds, in contrast, do not use such sharing policies, and promise far fewer resource breakdowns than grids; their Service-Level Agreements (SLAs) guarantee resource availability averages over 99%. Cost-wise, because grid infrastructure is freely shared, and university- or lab-sponsored, scientists often perceive grids as being free of charge, whereas they must pay actual money to use clouds.

Many grid users are now exploring the opportunity to migrate their scientific applications to commercial clouds [14, 21, 23]. However, this could be prohibitively expensive [34]. Our experiments showed, for example, that a 150-task BoT with a user-transparent electricity cost of \$1 and 8 hours completion time on 50 CPUs of the Open Science Grid [29] could incur a cost of \$34 on 50 `m1.large` instances of the Amazon EC2 commercial cloud (and take 2 hours). In this example, the cost of the cloud was larger by a factor of 34, but the makespan was only 4 times shorter.

Due in part to the low reliability of grids and in part to their

conveniently parallel structure, 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. In fact, previous studies [18,20] have shown that BoTs consistently account for over 90% of the multi-year workloads of production grids, both in terms of number of jobs they execute and of resources they consume. Thus, BoTs have been the *de facto* standard for executing jobs in grid environments over the past decade. When executing BoTs in a grid environment, scientists can *replicate* tasks. This increases the odds of timely task completion despite resource unreliability [4, 10, 24, 31, 38]. Replication, however, wastes CPU cycles and energy, and may carry other direct or indirect costs [10] (e.g., scheduler overload or management overheads). Moreover, it is difficult to select a *replication strategy* which 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, no study focused until now on explicitly identifying 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 needs and wants at that point in time, be they grids, clouds, dedicated self-owned machines, or any combination of the above. This vision presents many optimization opportunities, such as selecting the number of replicas per task and the structure of the reliable+unreliable environment, that can be exploited only when taking into account the specific preferences of each scientist. One scientist may want to obtain results by completing a BoT as quickly as possible, regardless of cost. Another may choose to minimize the cost and complete the BoT only on grid resources. Yet another scientist may try to complete work as soon as possible but under strict budget constraints. For example, a user with access to 10 grid (unreliable) machines and the modest budget of \$7.5 for executing a large BoT may decide to send the last 5-6 tasks to a cloud; another strategy could balance differently performance, cost, and resource use (through task replication). What all users share is a desire to use Pareto-efficient scheduling strategies (described in Section 2.1).

Our main research goal is to determine *which strategies 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? Finally, What is the proper timeout between submitting task instances?* Although Pareto-efficient strategies have been investigated before in different contexts [1, 13, 27, 28], 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 our scenario of scheduling large numbers of tasks online. Our main contribution towards choosing Pareto-efficient strategies is threefold:

1. We present a model for task scheduling in mixed environments with varying reliability and cost (Sections 2, 3).

2. We introduce **ExPERT**, a framework for dynamic selection of a Pareto-efficient scheduling strategy. **ExPERT** offers a wide spectrum of efficient strategies for different user makespan-cost trade-offs, leading to substantial savings in both (Section 4).
3. We validate and evaluate our framework through both simulations and experiments in real environments. By executing BoTs from the bioinformatics domain, with hundreds of tasks each, in various mixtures of *grid+cloud* environments (described in Section 5), we show (Section 6) that **ExPERT** can save substantial makespan and cost in comparison to scheduling strategies commonly used for workload scheduling in grids.

## 2. THE BASIC SYSTEM MODEL

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

### 2.1 Terminology

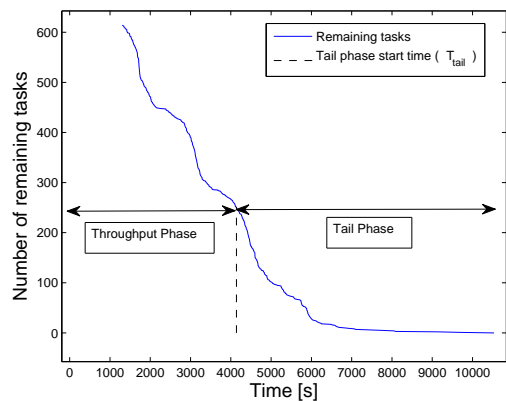
We begin with the terminology required for the problem of finding makespan- and cost-efficient strategies.

A **task** is a small computational unit that must be performed. A task **instance** is submitted to a resource. If the resource successfully performs the task, it returns a **result**. For a successful task instance, the *result turnaround time* is the period elapsed from when the instance was sent to when its result was received. For a failed instance, this period is defined as  $\infty$ . A **BoT** is a set of asynchronous, independent tasks, forming a single logical computation. In our system model, users submit BoTs to be executed task-by-task on the system resources. We divide the execution of a BoT into two major phases: the **throughput phase** followed by 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.

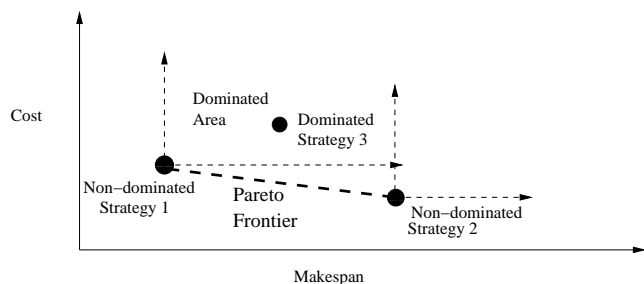
**Replication** is the invocation of multiple **instances** of the same task, possibly (but not necessarily) overlapping in time. The time of the first successful result of those instances is considered as the completion time for the task. The **reliability** of a resource pool is the probability that a task instance submitted to that pool returns a result.

The **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. In this work we ignore the costs of failed instances for two reasons. First, it is difficult to justify charging for an undelivered result. Second, if the user is willing to wait for results for a long period of time, and failed resources can resume work after repair, little partial work is lost and little extra cost is incurred by failures.

The **user’s scheduling system** (user scheduler) sends and,



**Figure 1: Remaining tasks as a function of time during the throughput phase and the tail phase. Input data: Experiment 6 (Section 6.3).**



**Figure 2: Illustration of Pareto frontier and a dominated strategy in the makespan  $\times$  cost space. Strategy 3 is dominated by strategy 1. Strategies 1 and 2 are non-dominated and form the Pareto frontier.**

if needed, replicates the user’s tasks to the resources provided by the *unreliable+reliable* mixed environment. A user **strategy** is a set of input parameters for this system, indicating when, where, and how to send and/or replicate tasks.

The **performance metrics** we use in this work are **cost per task** (the average cost of all BoT tasks) and **makespan** (either tail phase makespan or BoT 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 an 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” [7]) is the locus of all efficient strategies with respect to the searched strategy space. Any strategy which 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.

## 2.2 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 [31].

### 2.2.1 Environment

The environment consists of two queues. One is serviced by the **unreliable pool**, which consists of the unreliable resources owned by various owners. The other is serviced by the **reliable pool**.

We characterize, in turn, the unreliable and reliable resources, in terms of speed, reliability, and availability. The machines in the unreliable pool operate at various speeds; the reliable pools are homogeneous.

The unreliable pool is geographically and administratively distributed, and does operate by a Service Level Agreement (SLA). As we have shown in our previous work [19], failures in the unreliable pool are abundant and unrelated across different domains. Reliable machines never fail. In reality all machines may fail, but we justify the approximation by the large reliability difference between the unreliable and reliable pools.

Unreliable pool owners limit the number of concurrent unreliable resources available to the user. The reliable pool is available on demand: resources are charged as used, per charging period (one hour on EC2, one second on self-owned machines, etc.)

### 2.2.2 Execution Infrastructure Capabilities

We make no assumptions on task waiting time or on the unreliable system’s scheduling policy, other than that both can be modeled statistically (see Section 4.3).

We assume the user has an overlay middleware, which replaces malfunctioning hosts by 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.

We allow for loose connectivity between the user scheduler and the hosts. In such loosely connected systems (e.g., desktop grids using the BOINC [4] BoT runtime system), the exact time of a failure may be unknown. Hosts may lose and renew connections. Though queued tasks can be canceled, tasks on disconnected machines cannot be aborted. Every task instance has a deadline. If an instance’s result does not arrive by the deadline it is ignored, and the instance is considered failed. Hosts do not waste work on an instance whose deadline cannot be met.

## 3. THE SCHEDULING STRATEGY SPACE

In this section we introduce our model for scheduling tasks with replication in environments with mixed reliability, cost,

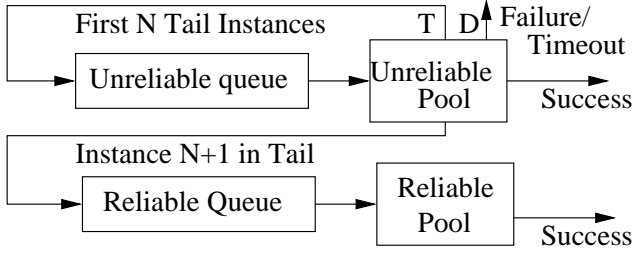


Figure 3:  $NTDM_r$  task instance flow during throughput phase and tail phase. Reliable machines serve only instance  $N + 1$  during the tail phase (throughput phase instances are not counted). During the throughput phase,  $T = D$ , so there is no replication.

and speed. The model generalizes state-of-the-art user strategies, e.g., of GridBoT users [31]. The user needs to choose a strategy in the space defined by this model, a choice we optimize.

We focus on optimizing the tail phase makespan and cost by controlling the tail phase scheduling strategy, for three reasons. First, in naive BOINC executions [4], the tail phase is an opportunity for improvement [30], 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 (as proved by Ghare and Leutenegger [17]), so we use it only during the tail phase, where significant optimization opportunities exist. Third, setting the decision point after the throughput phase makes it possible to base the optimization on the highly-relevant statistical data (e.g., of task turnaround times) collected during the throughput phase.

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 deadline length is a compromise between the time to the identification of 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. Very long deadlines, that is, much larger 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  $M_r$ :

1.  $N$  is the maximal number of instances sent for each task to the unreliable system from the start of the tail phase. A last,  $(N + 1)^{th}$  instance is always sent to the reliable system, 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

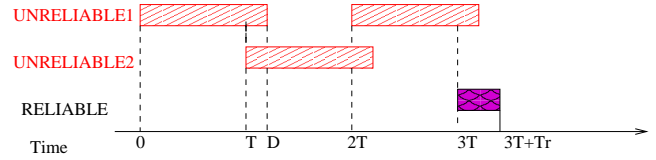


Figure 4: Machine occupancy for a single task under  $NTDM_r$  scheduling when  $N = 3$ , according to user information. The horizontal axis represents the time elapsed since the first instance was submitted. New instances are sent after time  $T$  and timeout after time  $D$ . In the case presented, all unreliable instances time-out and only the reliable instance succeeds. The user does not know whether an unreliable machine is functioning until the deadline.

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.

2.  $D$  is a deadline for an instance, measured from its submission 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 tasks which were assigned to faulty hosts.

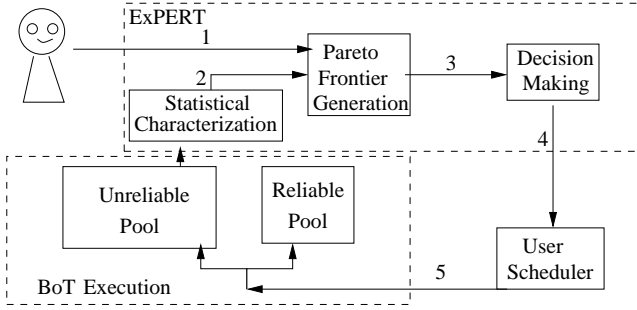
3.  $T$  is a timeout: the minimal time to wait before submitting another instance of the same task. Rather than having all instances submitted at the same time, each is submitted after a period  $T$  has passed from the previous instance submission, provided that no result has yet been returned. This restricts resource consumption. Figure 4 illustrates machine occupancy for  $N = 3$ ,  $0 < T < D$ , when all the unreliable instances fail.

4.  $M_r$  is the ratio of the reliable and unreliable pool sizes. 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 6.2.

The user’s main goal is to choose values for the parameters  $N$ ,  $T$ ,  $D$ , 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.

## 4. 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 complete scheduling process, which includes building a Pareto frontier of  $NTDM_r$  strategies, out of which the best strategy for the user is



**Figure 5: Flow of the ExPERT scheduling process, with user intervention points. Numbers indicate process stages.**

picked. To ensure flexibility, the user can interact with the process and reconfigure it.

#### 4.1 The ExPERT Scheduling Process

The ExPERT scheduling process has five steps, depicted in Figure 5. The process is *fully automatic* if the user’s utility function is expressed *a priori*. Alternatively, the process is *semi-automatic*, supplying the user with the Pareto frontier and letting the user choose a best strategy *a posteriori*. The semi-automatic approach suits users who prefer not to formalize their utility function. We now describe, in turn, the five steps:

1. *User Input*: The user supplies ExPERT with user-defined parameters (see Section 4.2) and, optionally, a utility function.
2. *Statistical Characterization*: ExPERT statistically characterizes the workload and the unreliable system on the basis of historical data (see Section 4.3).
3. *Pareto Frontier Generation*: ExPERT analyzes a range of strategies, and presents strategies which are efficient in both makespan and cost (see Section 4.4).
4. *Decision Making*: ExPERT designates efficient strategies which are the best for some predefined utility functions, including the user’s utility function, if the user formalizes it. Otherwise, the user directly chooses a strategy. ExPERT passes the  $N, T, D, M_r$  input parameters of the chosen strategy to the user’s scheduler—a resource management framework (e.g., GridBoT [31]).
5. *Replication*: The user’s scheduler replicates tasks and submits them to the two resource queues (introduced in Section 2.2.1) according to these parameters.

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

Traditionally, BoTs are executed through schedulers such as BOINC or Condor using a pre-set strategy, defined when the

**Table 1: 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

BoT is submitted. Though historical performance data has been used by others for resource exclusion [24], it was not used for adapting strategy parameters during the BoT’s run. ExPERT, on the other hand, leverages the power of the current BoT’s statistics to dynamically adapt the BoT’s replication strategy, to efficiently execute the BoT.

#### 4.2 User Input (Step 1)

In Step 1 the user supplies ExPERT with data about mean CPU times, costs, and the reliable resource pool’s size. ExPERT uses the CPU times (denoted  $T_r, T_{ur}$ ) and runtime costs in cents per second (denoted  $C_r, C_{ur}$ ) to estimate the BoT’s cost under different strategies. The user input is detailed in Table 1.  $M_r^{max}$ , the upper bound of  $M_r$ , is derived from the unreliable pool’s 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).

The cost data may reflect various kinds of cost, such as monetary payments, energy waste, and environmental damage. We give in the following three examples of user input selection. First, a user concerned only with environmental damage may set  $C_r = \alpha \frac{T_{ur}}{T_r} C_{ur}$ , where  $\alpha$  stands for the ratio of carbon footprints of large data centers and of desktops. Second, a monetary-oriented user may set unreliable costs as zero, representing the grid as free of charge, and reliable costs as cloud costs. Third, a user may use a complex environment, one for which the unreliable resource is composed of GPGPU machines that consume twice as much power as CPUs, and the BoT can run 100 times faster on them. This user may set  $T_r = 100T_{ur}$ ,  $C_r = 0.5C_{ur}$ , thus reflecting the better throughput of GPGPU machines, along with their increased energy consumption.

#### 4.3 Statistical Characterization (Step 2)

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  $\sharp_{ur}$ , on the basis of the requested size and the rate of instances sent and results received. This section describes 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 factors of the environment, the user, and the workload. 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 from historical data as the fraction of results out of the number of instances in discrete sets of consecutively sent instances.

The dependency of  $F(\cdot)$  on  $t'$  through  $\gamma(t')$  enables the CDF to change over time, and thus necessitates a prediction model. ExPERT can either compute  $\gamma(t')$  *offline* or estimate it *online*. The accuracy of the two models is compared in Section 6.3. In the offline model,  $\gamma(t')$  is fully known. In the online model, the number of returned results is counted at  $T_{tail}$  (the tail phase start time). The extent to which the reliability is known varies along the time line, which is divided into three epochs, according to the time in which instances are sent:

1. Full Knowledge:  $t' < T_{tail} - D$ . Instances sent at these times that have not yet returned will not return anymore. Online reliability is identical to offline reliability.
2. Partial Knowledge:  $T_{tail} - D \leq t' < T_{tail}$ . Instances sent at these times that have not yet returned, might still return. Let  $\hat{F}(t, t')$  denote  $F(t, t')$  as was measured for instances sent at time  $t'$ .  $\hat{F}(t, t') = F(t, t')$  for  $t \leq T_{tail} - t'$ . We use Equation 1 to estimate

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

We further limit  $\gamma(t')$  during this epoch. From below, we limit by the minimal historical value during the first epoch. From above we only limit it by 1 because resource exclusion [24] (avoiding faulty hosts) may raise the reliability values above their maximal historical values.

3. Zero Knowledge:  $t' \geq T_{tail}$ . No result has yet returned. We use an average of the mean reliabilities during the full knowledge epoch and the partial knowledge epoch, thus incorporating old accurate data as well as updated, inaccurate data. Our experiments indicate that an average of equal weights produces the best prediction for  $\gamma(t')$  during this epoch.

#### 4.4 Pareto Frontier Generation (Step 3)

ExPERT generates the Pareto frontier using data from steps 1 and 2 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 is configurable. The sampling resolution used in our evaluations is described in Section 5.3. The *ExPERT*

*Estimator* estimates the mean makespan and cost of each sampled point through simulation, which is performed as follows.

The *ExPERT Estimator* models  $\#ur$  unreliable and  $[M_r\#ur]$  reliable resources, each resource pool having a separate, infinite queue. For simplicity, in this work the queues are FCFS. 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 of the *ExPERT Estimator*, the following steps are taken:

1. Each running instance is checked for success or time-out.
2. If a task has not yet returned a result, and time  $T$  has already passed since its last instance was sent, and there is no instance for this task in any queue, then one instance is enqueued for this task.
3. Idle machines are given an instance from the appropriate queue, unless it is empty.

Once all the sampled strategies are analyzed, ExPERT produces the Pareto frontier. It eliminates 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: a viable option for the user. 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 Pareto frontier. 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.

#### 4.5 Decision Making (Step 4)

We now show how ExPERT can be used to choose a strategy, with a focus on explaining the choices presented by ExPERT to the user. We use as an example the execution of a scientific BoT. The CDF of the turnaround time of its tasks is depicted in Figure 6. For our example, the values supplied by the user to ExPERT are listed in Table 2.

First, we showcase 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 lead to significant waste of time and money. For our example, Figure 7 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

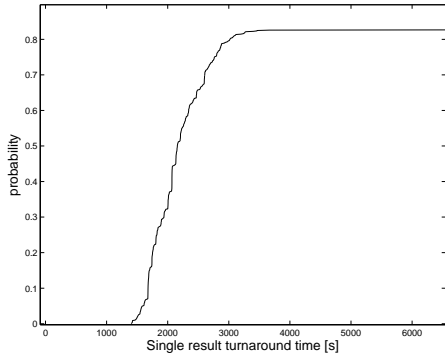


Figure 6: CDF of single result turnaround time, obtained from Experiment 11 (Section 6.3).

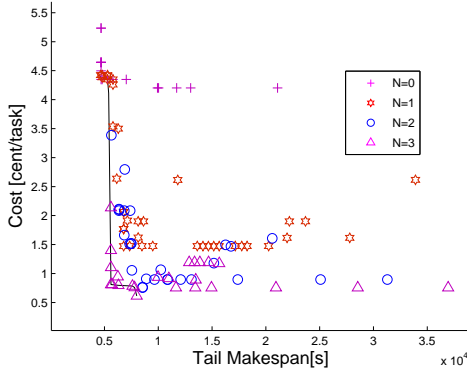


Figure 7: Pareto frontier and sampled strategies. Input data: Experiment 11 (Section 6.3).

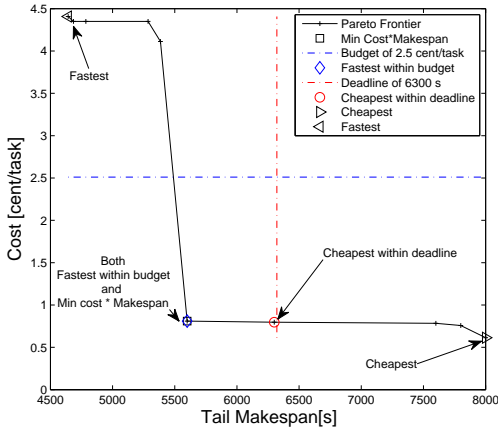


Figure 8: Pareto frontier and examples of best points: by a utility function of tail-phase-makespan $\times$ cost, within budget, within deadline on tail phase makespan, by cost only and by tail phase makespan only. Input data: Experiment 11 (Section 6.3).

(the top right-most star symbol in Figure 7). In contrast, ExPERT recommends a strategy based on using  $N = 3$ , which

Table 2: 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: $T_{ur}$ .
$C_{ur}$	$\frac{1}{3600} \frac{\text{cent}}{\text{second}} = 10 \frac{\text{cent}}{\text{KWH}} \cdot 100W$
$C_r$	$\frac{34}{3600} \frac{\text{cent}}{\text{second}}$ : EC2's m1.large on-demand rate

leads to a makespan around 5,000s (5 times better) and a cost of under  $1 \frac{\text{cent}}{\text{task}}$  (the triangle symbol at the “knee” of the continuous curve in Figure 7).

Second, we illustrate how ExPERT assists the user’s decision process. Figure 8 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  $\times$  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 8, a higher-density sampling of the search space is required. However, even a low sampling resolution is enough to find 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  $N$ ,  $T$  and  $D$  and  $M_r$ , which define the chosen strategy. Those parameters are passed to the user’s scheduler and are used to run the user’s tasks.

## 5. THE EXPERIMENTAL SETUP

In this section we present the experimental setup used for our simulation and real-world experiments. To evaluate ExPERT in a variety of scenarios yet within our budget, we used simulated experiments coupled with data obtained from real-world experiments. The simulator was created by re-using a prototype implementation of the *ExPERT Estimator* introduced in Section 4.4; in this sense, our simulations can be seen as emulations of the ExPERT process. We validated the simulator’s accuracy by comparing the results of simulations with the results obtained through real-world experiments performed on different combinations of unreliable and reliable pools, including grids, self-owned machines, and a real cloud (Amazon EC2).

### 5.1 Evaluation Workload

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 [32]. The BoTs are characterized in Table 3. To evaluate the performance of the  $NTDM_r$  Pareto frontier, we used (as in Section 4.5) the CDF obtained from Experiment 11 and shown in Figure 6.

**Table 3: Workloads with  $T, D$  strategy parameters and throughput phase performance statistics. WL denotes Workload index. WM is an execution environment defined in Table 4.**

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

## 5.2 Experimental Environments

The real-world experiments were conducted using GridBoT [31]. GridBoT provides a unified front-end to multiple grids and clouds. It employs dynamic run-time scheduling and replication strategies to execute the BoTs in multiple environments simultaneously. It relies on the BOINC infrastructure. To implement the limit to the CPU time consumed by a task instance, we used the BOINC parameter `rs_c_flops_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  $D$ .

The simulation-based experiments used the same discrete event-based *ExPERT Estimator* we developed for building the Pareto frontier (Section 4.4). For comparison, we augmented the  $N, T, D, M_r$  strategies already implemented in the *Estimator* with other strategies, described in Section 5.3. Although we considered using a grid simulator such as SimGrid [11], GridSim [9], or DGSim [22], we have ultimately decided to build our own simulation environment. Our simulations are specifically tailored for running ExPERT and have a simple, trace-based setup. Other simulators require elaborate definitions of scheduling parameters, grid resources, and background load descriptions. More importantly, as far as we know, no other simulator has been validated for the scheduling strategies and environments investigated in this work.

Our simulation environment requires several user-specified parameters (introduced in Section 4.2). The parameter values used in our experiments are summarized in Table 2. To estimate  $C_{ur}$  we used the power difference between an active and idle state according to AMD’s ACP metric [3]. Multiplying power difference values for Opteron processors [3] by two, to allow for cooling system power, results in the range 52W-157W; hence we use 100W here. To get  $C_{ur}$  we multiply the power by a residential electricity cost.

The resource pools are detailed in Table 4. In each experiment we used one unreliable resource combination (one row) and at most one reliable resource.

## 5.3 Static Scheduling Strategies

In this section we present *static* strategies that were used or considered in the past by GridBoT users because they are easy to express. A static strategy is a strategy which

**Table 4: Real resource pools used in our experiments.**

Reliable	Properties
Tech	20 self-owned CPUs in the Technion
EC2	20 m1.large EC2 [2] cloud instances
Unreliable	Properties
WM	UW-Madison Condor pool [37]. Utilizes preemption.
OSG	Open Science Grid [29]. Does not preempt.
OSG+WM	Combined pool, half $\#ur$ from each
WM+EC2	Combined pool, 20 EC2 + 200 WM
WM+Tech	Combined pool, 20 Tech + 200 WM

is pre-set before the BoT starts. Unless otherwise stated, during the throughput phase these strategies are “no replication” ( $N = \infty, T = D = 4T_{ur}$ ) and the reliable pool is idle. Though some of these strategies are  $NTDM_r$  strategies, they are not necessarily Pareto-efficient for any specific scenario. In Section 6.1 we will compare these strategies to Pareto-efficient  $NTDM_r$  strategies, to demonstrate ExPERT’s cost and makespan savings. The static strategies are:

1. *AR: All to Reliable*: use only reliable machines for the duration of the BoT. When the reliable machines are ample and fast, and when the reliability of the unreliable machines is low, this is a fast strategy.
2. *TRR: all Tail Replicated to Reliable*: at  $T_{tail}$ , replicate all remaining tasks to the reliable pool, to decrease the tail phase makespan. This is an  $NTDM_r$  strategy ( $N = 0, T = 0, M_r = M_r^{max}$ ).
3. *TR: all Tail to Reliable*: at  $T_{tail}$ , enqueue every *timed out* tail task to the reliable pool. This is an  $NTDM_r$  strategy ( $N = 0, T = D, M_r = M_r^{max}$ ).
4. *AUR: All to UnReliable, no replication*: use the default throughput phase strategy during the tail phase. If the unreliable system is cheap, this is the cheapest option. This is an  $NTDM_r$  strategy ( $N = \infty, T = D$ ).
5. *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. This heuristic strategy does not necessarily use the entire budget.
6. *CN $\infty$ : Combine resources, no replication*: throughout the execution of the BoT, deploy tasks from the unreliable queue on the reliable pool if the unreliable pool is fully utilized.
7. *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 two instances, one for each pool.

## 6. THE EXPERIMENTAL RESULTS

We begin by evaluating  $NTDM_r$  Pareto frontiers by comparing them to the static strategies introduced in Section 5.3. We proceed to demonstrate the importance of  $M_r$  as a strategy parameter in Section 6.2. We then validate the *ExPERT Estimator* logic in Section 6.3 and discuss the time it takes to run ExPERT in Section 6.4.

## 6.1 Comparison of $\text{ExPERT}$ -Recommended with Static Scheduling Strategies

To evaluate the benefits of using  $N, T, D, M_r$  Pareto-efficient strategies, we compare them with the seven static scheduling strategies introduced in Section 5.3. 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  $N = 0 \dots 3$ ,  $M_r = 0.02 \dots M_r^{\max}$ , and  $0 \leq T \leq D \leq 4T_{ur}$ .  $T, D$  were sampled within their range at 5 different points each. For higher  $N, T, D$  values, the cost-makespan trade-off becomes practically nonexistent; then, no decision can lead to significant improvements.

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 9a. **The Pareto frontier found by  $\text{ExPERT}$  dominates all the tested static strategies except AUR**, that is, for *any* utility function, for each tested static strategy except AUR,  $\text{ExPERT}$  recommends at least one  $\text{NTDM}_r$  strategy that improves both metrics. For example,  $\text{ExPERT}$  finds *several* strategies that dominate strategy the commonly-used  $\text{CN}\infty$  strategy (combine resources, no replication, see Section 5.3). One such strategy found by  $\text{ExPERT}$ , is:

*ExPERT recommended*  $\equiv$

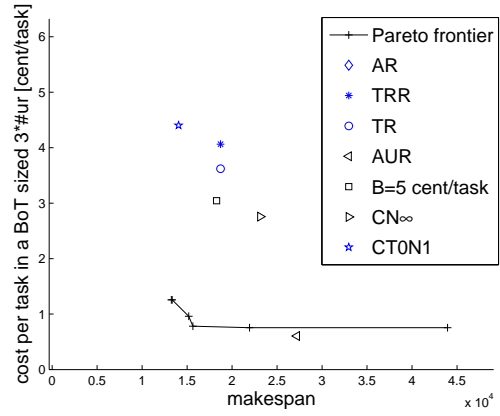
( $N = 3, T = T_{ur}, D = 2T_{ur}, M_r = 0.02$ ), in words: Send  $N = 3$  instances to the unreliable pool during the tail phase, each timed out after twice the average task time ( $D = 2T_{ur}$ ). Send the next instance after the average task time passes ( $T = T_{ur}$ ). Use only one ( $\#ur = 50, 50 \times M_r = 1$ ) reliable machine at a time.

*ExPERT recommended*, which is depicted in Figure 9a at the “knee” of the Pareto frontier, yields a makespan of 15,640s for the cost of  $0.78 \frac{\text{cent}}{\text{task}}$ . *ExPERT recommended cuts down 72% of  $\text{CN}\infty$ 's cost and 33% of its makespan.* Although it does not dominate AUR, which by definition (Section 5.3) is the cheapest scheduling strategy and thus cannot be dominated, several strategies on the Pareto frontier found by  $\text{ExPERT}$  lead to much better makespan at small cost increase versus AUR.

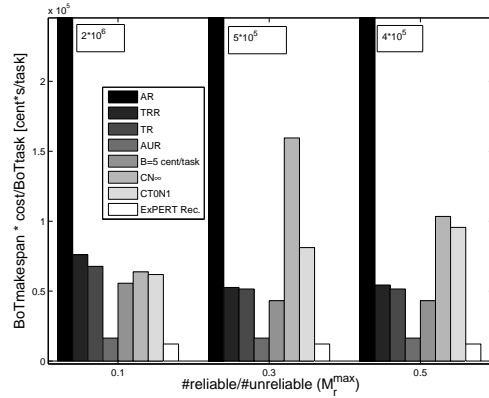
Next, we focus on the performance of the strategies in terms of a specific utility function: *minimize tail-phase-makespan  $\times$  cost per task*, that is, a utility function that values equally one-second and one-cent improvements in the execution of a BoT. We compare the utility obtained by the user for this function, when the scheduling strategy is *ExPERT recommended* or one of the seven static scheduling strategies. Figure 9b depicts the results of this comparison. *ExPERT recommended* is 25% better than the second-best performer AUR, several orders of magnitude better than the worst performer AR, and 72%-78% better than the other tested strategies. We conclude that *ExPERT recommended delivers significantly better utility than all the tested static strategies* and outperforms all these strategies except AUR.

## 6.2 Impact of $M_r$ on the Performance of $\text{ExPERT}$

We now demonstrate the benefit of elasticity: allowing  $M_r$ , which provides a bound on the number of concurrently used reliable resources (see Section 3), to be a scheduling strategy parameter rather than a system constant. We consider



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

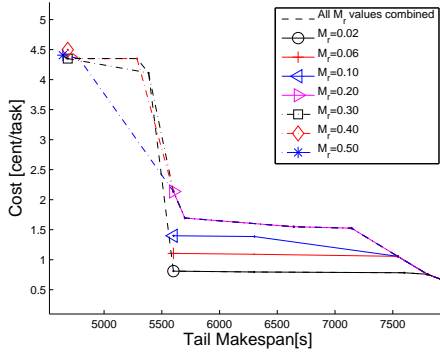


(b) Makespan-cost product for static and *ExPERT recommended* strategies, for  $M_r^{\max} = 0.1, 0.3, 0.5$ . Bars for strategy AR are truncated; their height appears beside them. Smaller values are better.

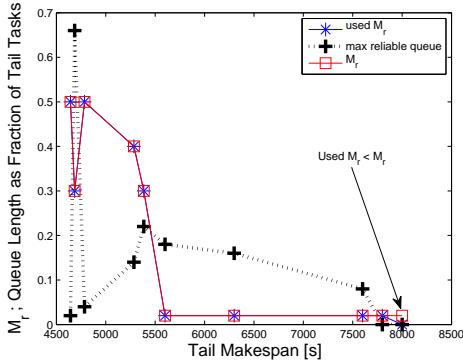
**Figure 9: Static strategies compared to Pareto-efficient  $\text{NTDM}_r$  strategies, for a BoT of 150 tasks, using 50 unreliable machines and 25 reliable ones. Performance is evaluated by full BoT makespan, because the strategies differ during the throughput phase. Input data: Experiment 11.**

$M_r = 0.02 \dots 0.50$ , which means that reliable resources are less than 50% of the resources available to the user.

We first analyze the need for allowing the users 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 10 seven such frontiers. As shown by the figure, overall, high  $M_r$  values allow a wide range of makespan values, 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  $\text{NTDM}_r$  strategies,  $M_r$  should not be fixed in advance, but set in accordance with the desired makespan.**



**Figure 10: 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 11: Reliable pool use by efficient strategies.**

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 11 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 a consequence of the appearance of queues of tasks at the reliable pool, which is demonstrated by the non-zero values of the queue-related curve in Figure 11. For the right-most point on the  $M_r$  and *used  $M_r$*  curves, the values of  $M_r$  and *used  $M_r$*  are different. We explain this through 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.

### 6.3 Simulator Validation

We conducted 13 large-scale experiments to validate the simulator and the wrapped *ExPERT Estimator*. In each experiment, we applied a single strategy to specific workload and resource pools. The simulations include a random component, so to ensure statistical confidence, we compared the performance metrics (tail phase makespan, cost per task) of each real experiment with mean values of 10 simulated

experiments (averaging over 100 instances did not result in notable improvement). We compared real and simulated performance metrics for both the offline and the online models (defined in Section 4.3). The experiments are listed by decreasing order of average reliability in Table 5.

On average, performance metrics of the offline simulations deviate from real experimental values by 7% and 10% for cost and tail phase makespan, respectively. The on-line simulations deviated from real experimental values by twice as much. We identify four main causes for these deviations:

1. The simulator provides expectation values of performance metrics. In contrast, a real experiment is a single sample. It is also unreproducible, because grid conditions are not under our control. 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 off from the offline simulation.
2. 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 [24] policies and a varying task length distribution.
3. ExPERT assumes it is never informed of failures before the deadline  $D$ . In real experiments, some machines do inform about failures, and are replaced.
4. In real experiments, the 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.

### 6.4 ExPERT Runtime: Online or Offline?

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 a Intel(R) Core(TM)2 Duo CPU P8400 @ 2.26GHz. For space sampling, dozens of single strategy simulations are performed, each lasting several seconds. We consider a runtime in the order of minutes appropriate for BoTs of hundreds of tasks or more that are the focus of this work.

The runtime of ExPERT may be further improved. For ExPERT, time may be traded for prediction 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.

## 7. RELATED WORK

**Cost-unaware replication algorithms** Much research on replication algorithms relied on the assumption that computation is free of charge and only related to its effect on

**Table 5: Experimental results.** *WL* denotes workload according to Table 3. *N* is the  $NTDM_r$  parameter.  $\#ur$  is an estimate for the size of the unreliable pool. *ur* and *r* denote choice of pools according to Table 4.  $\gamma$  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.  $\Delta TMS$  and  $\Delta C$  denote deviation of simulated values from real ones. The strategy in Experiment 5 is  $CN_\infty$ : *Combine resources, no replication* (which is not an  $NTDM_r$  strategy).

No.	Experiment Parameters					Measured in Real Experiment				Simulated Experiment Deviation			
	WL	N	$\#ur$	ur	r	$\gamma$	RI	TMS[s]	$C \left[ \frac{cent}{task} \right]$	Offline [%] $\frac{\Delta TMS}{TMS}$	$\frac{\Delta C}{C}$	Online [%] $\frac{\Delta TMS}{TMS}$	$\frac{\Delta C}{C}$
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	$\infty$	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	$\infty$	200+20	WM+EC2	-	0.970	41	6,113	1.48	37	-2	29	-2
6	WL5	$\infty$	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

load and makespan [10, 12, 16, 24, 33, 36], whereas we explicitly consider execution costs. Dobber, v.d. Mei, and Koole [16] have created an on-the-fly criterion for choosing between immediate replication and dynamic load balancing. Casanova [10] has shown the impact of simple replication policies on resource waste and fairness. Kondo, Chien, and Casanova [24] have combined replication with resource exclusion. Cirne et al. [12] and Silva et al. [33] have analyzed immediate replication with no deadline for perfectly reliable heterogeneous machines. Wingstrom and Casanova [36] 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.

**Cost-aware replication algorithms** The concept of utility functions as the target of the optimization process has also received attention in the past [6, 8, 15]. Buyya, Abramson, Giddy and Stockinger [8] researched economic mechanisms for setting grid computation costs, for several utility functions. One of their estimation methods is Pareto-efficiency. Benoit et al. [6] assumed a linear risk model for machine unavailability on homogeneous remote machines, and considered overhead costs and operational costs. Our work allows for both a general user function and a general probability distribution of task success. Andrzejak, Kondo, and Anderson [5] controlling reliable and unreliable pool sizes in a combined pool to Pareto-optimize cost and availability for web services.

## 8. CONCLUSION

We addressed in this work one of the main problems facing the large number of scientists who rely on Bags-of-Tasks (BoTs) in mixtures of computational environments such as grids and clouds, namely the lack of tools available for selecting efficient scheduling strategies for user-defined utility functions. Our ExPERT BoT scheduling framework selects the Pareto-efficient strategies from a range of replication strategies for running BoTs on mixtures of environ-

ments with varying reliability, cost, and speed. For any user-provided utility function, ExPERT finds the best strategy in a large, sampled strategy space.

We validated ExPERT’s predictive accuracy and demonstrated its viability through a variety of experiments in real and simulated environments. We found that ExPERT can achieve a 72% cost reduction and a 33% shorter makespan compared with commonly-used static scheduling strategies such as combining the reliable and unreliable resources. We found that the  $NTDM_r$  strategy space is large enough to provide considerable flexibility in utility functions based on both execution time and cost. 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. We explain the performance improvement by 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.

## 9. REFERENCES

- [1] H. A. Abbass, R. Sarker, and C. Newton. PDE: A Pareto-frontier differential evolution approach for multi-objective optimization problems. In *CEC*, 2001.
- [2] Amazon Elastic Compute Cloud (Amazon EC2). Website. <http://aws.amazon.com/ec2/>.
- [3] ACP — the truth about power consumption starts here. Website. [http://www.amd.com/us/Documents/43761C\\_ACP\\_WP\\_EE.pdf](http://www.amd.com/us/Documents/43761C_ACP_WP_EE.pdf).
- [4] D. P. Anderson, E. Korpela, and R. Walton. High-performance task distribution for volunteer computing. In *e-Science*, pages 196–203, 2005.
- [5] A. Andrzejak, D. Kondo, and D. P. Anderson. Exploiting non-dedicated resources for cloud computing. In *NOMS 2010*, 2010.
- [6] A. Benoit, Y. Robert, A. L. Rosenberg, and F. Vivien. Static strategies for worksharing with unrecoverable interruptions. In *IPDPS*, 2009.

- [7] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.
- [8] 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 (CCPE)*, 14(13-15):1507–1542, 2002.
- [9] 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 (CCPE)*, 14(13):1175–1220, 2002.
- [10] H. Casanova. On the harmfulness of redundant batch requests. In *HPDC*, pages 255–266, 2006.
- [11] 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.
- [12] 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.*, 33(3), 2007.
- [13] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley and Sons, first edition, 2001.
- [14] E. Deelman, G. Singh, M. Livny, G. B. Berriman, and J. Good. The cost of doing science on the cloud: the montage example. In *SC*, page 50, 2008.
- [15] 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.
- [16] 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.*, 20(2), 2009.
- [17] G. D. Ghare and S. T. Leutenegger. Improving speedup and response times by replicating parallel programs on a snow. In *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing*, 2004.
- [18] 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.
- [19] A. Iosup, M. Jan, O. O. Sonmez, and D. H. J. Epema. On the dynamic resource availability in grids. In *GRID*, pages 26–33, 2007.
- [20] 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.*, 24(7), 2008.
- [21] 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.*, 2010. (in print) [Online] [www.st.ewi.tudelft.nl/~iosup/cloud-perf10tpds\\_in-print.pdf](http://www.st.ewi.tudelft.nl/~iosup/cloud-perf10tpds_in-print.pdf).
- [22] A. Iosup, O. Sonmez, and D. Epema. Dgsim: Comparing grid resource management architectures through trace-based simulation. In *Euro-Par '08: Proceedings of the 14th International Euro-Par Conference on Parallel Processing*, pages 13–25, Berlin, Heidelberg, 2008. Springer-Verlag.
- [23] K. R. Jackson, L. Ramakrishnan, K. J. Runge, and R. C. Thomas. Seeking supernovae in the clouds: a performance study. In *HPDC*, pages 421–429, 2010.
- [24] D. Kondo, A. A. Chien, and H. Casanova. Resource management for rapid application turnaround on enterprise desktop grids. In *SC'04*, 2004.
- [25] D. Kondo, G. Fedak, F. Cappello, A. A. Chien, and H. Casanova. Characterizing resource availability in enterprise desktop grids. *Future Generation Comp. Syst.*, 23(7):888–903, 2007.
- [26] 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*, pages 398–407, 2010.
- [27] C. A. Mattson and A. Messac. Pareto frontier based concept selection under uncertainty, with visualization. *Optimization and Engineering*, 6(1), 2005.
- [28] A. Messac, A. Ismail-Yahaya, and C. A. Mattson. The normalized normal constraint method for generating the Pareto frontier. *Structural and Multidisciplinary Optimization*, 25(2), 2003.
- [29] The open science grid. <http://www.opensciencegrid.org>.
- [30] M. Silberstein. Building online domain-specific computing service over non-dedicated grid and cloud resources: Superlink-online experience. In *CCGRID '11*, 2011.
- [31] M. Silberstein, A. Sharov, D. Geiger, and A. Schuster. Gridbot: Execution of bags of tasks in multiple grids. In *SC'09*, 2009.
- [32] 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. *American Journal of Human Genetics*, 78(6):922–935, 2006.
- [33] 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.
- [34] 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: Proceedings of the 15th ACM Mardi Gras Conference*, pages 1–8, 2008.
- [35] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.
- [36] J. Wingstrom and H. Casanova. Probabilistic allocation of tasks on desktop grids. In *IPDPS*, 2008.
- [37] UW-Madison CS Dept. Condor pool. Website. <http://www.cs.wisc.edu/condor/uwcs/>.
- [38] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *OSDI'08*, 2008.