# Risk-Aware Scheduling Algorithms for Variable Capacity Resources

Lucas Perotin, Chaojie Zhang, Rajini Wijayawardana, Anne Benoit, Yves Robert, Andrew A Chien

# Risk-Aware Scheduling Algorithms for Variable Capacity Resources

Lucas Perotin
Laboratoire LIP, ENS Lyon
Lyon, France
lucas.perotin@ens-lyon.fr

Chaojie Zhang
Microsoft Research
Seattle, WA, USA
chaojiezhang@microsoft.com

Rajini Wijayawardana
University of Chicago
Chicago, IL, USA
rajini@uchicago.edu

Anne Benoit
Laboratoire LIP, ENS Lyon
Lyon, France
anne.benoit@ens-lyon.fr

Yves Robert*
Laboratoire LIP, ENS Lyon
Lyon, France
yves.robert@ens-lyon.fr

Andrew A. Chien†
University of Chicago
Chicago, IL, USA
aachien@uchicago.edu

## ABSTRACT

The drive to decarbonize the power grid to slow the pace of climate change has caused dramatic variation in the cost, availability, and carbon-intensity of power. This has begun to shape the planning and operation of datacenters. This paper focuses on the design of scheduling algorithms for independent jobs that are submitted to a platform whose resource capacity varies over time. Jobs are submitted online and assigned on a target machine by the scheduler, which is agnostic to the rate and amount of resource variation. The optimization objective is the goodput, defined as the fraction of time devoted to effective computations (re-execution does not count). We introduce several novel algorithms that: (i) decide which fraction of the resources can be used safely; (ii) maintain a risk index associated to each machine; and (iii) achieves a global load balance while mapping longer jobs to safer machines. We assess the performance of these algorithms using one set of actual workflow traces together with three sets of synthetic traces. The goodput achieved by our algorithms increases up to 10% compared to standard first-fit approaches, while we never experience any loss in complementary metrics such as the maximum or average stretch.

## KEYWORDS

Online scheduling, independent jobs, variable capacity, power source.

**ACM Reference Format:**
Lucas Perotin, Chaojie Zhang, Rajini Wijayawardana, Anne Benoit, Yves Robert, and Andrew A. Chien. 2023. Risk-Aware Scheduling Algorithms for Variable Capacity Resources. In *PMBS - Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3624062.3624194

## 1 INTRODUCTION

With growing global concern about climate change [17, 18], the end of Dennard scaling, and continued exponential growth of computing use [9, 16], there is growing interest in how to reduce the negative environmental impacts of computing, that is, improve its sustainability [5, 8, 13, 21]. A popular approach that exploits the variation of renewable generation (wind, solar), is the idea of temporal and spatial load shifting to match computing power consumption with the availability of low-carbon power [7]. Viewed at a single computing site, this appears as a datacenter or HPC platform whose capacity evolves with time, depending upon the cost and the environmental policy (e.g., reduce power supply when the power source is coal instead of wind or solar), termed the *variable capacity scheduling* problem [23].

Online scheduling techniques for independent jobs have received considerable attention since they lie at the heart of *batch schedulers*. The traditional setting deals with fixed-capacity resources that do not evolve over time, such as a parallel HPC platform. From the platform owner's perspective, the standard objective is to maximize utilization, defined as the fraction of time where platform resources execute computations [10]. From the user's perspective, the standard objective is to minimize the maximum (or sometimes average) stretch. The stretch of a job is the ratio of its response time (time elapsed between submission and completion) over its execution time, and is preferred to the plain response time for fairness [4].

This paper focuses on scheduling techniques for independent jobs, when variations in power supply imply changes in the number of available computing resources over time. The scheduler is agnostic to the rate and amount of resource variations but must prepare for such variations. In particular, a given resource may be shut down abruptly because of a power shortage, in which case all the jobs executing on that resource are interrupted and must be re-executed later on. Therefore, we design risk-aware strategies that assign incoming jobs to the *right* target machine, with some optimization criteria in mind. Because today's power grids have rapid (hourly, daily, weekly) and large variation (3-5x today, growing to 10x), and shifting benefit increases with the magnitude of load shifted, we consider dynamic ranges as large as 80% of the maximum capacity. With varying resources, platform utilization is no longer an adequate criterion, because partial executions of jobs that get interrupted do not count as actual progress of the jobs. We use the goodput instead to account for this.

This paper lays the foundation for a risk-aware strategy that maps jobs onto machines. We make several simplifying assumptions that enable us to design a model and assess the efficiency of our algorithms by simulation:

- The target platform consists of a collection of identical parallel

*Also with: University of Tennessee Knoxville, USA.
†Also with: Argonne National Laboratory, USA.

*PMBS'23, November 2023, Denver, CO, USA*
2023. ACM ISBN 979-8-4007-0785-8/23/11…$0.00
https://doi.org/10.1145/3624062.3624194

machines, each equipped with many cores;
• Variations in power capacity imply changes in the number of machines alive at a given time $t$. Alive machines at time $t$ are defined as machines that are switched on and execute jobs at time $t$;
• Power consumption at time $t$ is directly proportional to the number of machines that are switched on at time $t$; thereby we ignore possible variations due to the actual load, operating frequency and application behavior on each machine at time $t$;
• The number of alive machines at time $t$ is not known before time $t$. Instead, it obeys a random walk that evolves from the number of alive machines at time $t-1$.

Maybe the most drastic assumption is the last one: machines are added to and removed from the pool of available machines without explicit warning. It implies that jobs that are running on an alive machine get interrupted without notice when that machine gets removed from the pool. This scenario is particularly relavant to deal with stranded power [21]. Alternative approaches include:
(1) Execution can continue when the machine is removed from the pool, but at a higher price. This would model the scenario where alive machines are powered by green sources, and brown power can complement green power when needed;
(2) Variations of capacity are known some time in advance, so that jobs running on machines that are going to be switched off soon could take a proactive checkpoint and then continue on another machine, instead of resuming execution from scratch.
The first approach requires to use total cost as a complementary objective, and a model to take it into account. The second approach requires many additional parameters, both for the prediction mechanism (prediction time, recall and precision) and for job characteristics (checkpointable or not, checkpoint durations). Instead, we rely on a simple but reasonable Markov model for machine availability, which has been shown accurate to model resource variation in several frameworks [19]. Variations in wind power nicely obey such a Markov model [12]. Variations in solar power would require a more complicated model, such as a heterogeneous Markov chain to account for different contexts (e.g., day or night) [22].

Our main contributions are the following:
• We provide a simplified yet realistic model for power variation that translates into the number of available machines obeying a random walk at each step, and we present the first complexity results for the *variable capacity scheduling* problem;
• We design novel risk-aware scheduling and mapping algorithms that are capable of mitigating the impact of power variation by using several new techniques, such as: (i) keeping an ordered list of machines ordered by potential risk; (ii) mapping each job to a given set of machines according to its relative length w.r.t. the set of jobs released so far; (iii) maintaining local queues to achieve a bounded maximum stretch; (iv) re-execute interrupted jobs on new machines when power supply decreases; and (v) re-distribute pending jobs on new machines when power supply increases;
• We assess the performance of these novel algorithms using an extended set of simulations, and report significant gains in the achieved goodput over standard first-fit algorithms. Nicely, this increase of the goodput is achieved without any loss in complementary metrics such as maximum or average stretch.

## 2 RELATED WORK

Due to lack of space, we refer to the extended version [11] for a survey of related work and only quote a few references here. A body of research deals with the addition and removal of resources with time: [20] considers the case of volatile computing resources that are turned fully on or off to best utilize stranded renewable generation. Live migration of VMs between hosts as an energy efficiency mechanism is explored in [3]. Past studies have explored scheduling policies that assure secure job execution in the presence of unpredictable resource failures. [14] extends the known scheduling heuristics, preemptive, replication and delay-tolerant, to provide security assurances. [15] constructs statistical models to assess the reliability of resources based on prior performance and behavior, considering this reputation-based reliability rating in the job allocation algorithm. Reputation-based scheduling on unreliable resources [1, 15] can be considered parallel to our work. However, this body of work deals with assigning jobs to redundancy groups, while our work relates to scheduling independent jobs on volatile nodes. Opportunities to mitigate performance degradation from capacity variation through intelligent termination of jobs are explored in [23], which takes a reactive approach on handling variable resource capacity. Our work takes a proactive approach in scheduling jobs on these resources, and provides novel algorithms to assign jobs to machines.

None of the aforementioned work directly addresses the problem of online risk-aware scheduling on variable capacity resources. However, they provide an important backdrop to techniques that could potentially be adapted for this purpose. We build upon widely accepted heuristics, assigning jobs to individual physical nodes in a highly volatile renewable based environment.

## 3 FRAMEWORK

### 3.1 Target Platform

We consider a parallel platform that consists of a set $\mathcal{M}$ of $M^+$ identical parallel machines, each equipped with $n_c$ cores. Each machine $m \in \mathcal{M}$ requires a certain amount of power $P$ to run. For simplicity, we assume that $P$ is constant whenever the machine is switched on, even if some of the cores are unused. In other words, $P$ is proportional to the number of cores $n_c$ available in machine $m$, i.e. $P = p \times n_c$ for some constant $p$. Our scheduling problem includes capacity variations, where the overall available power capacity is a function of time $t$, denoted as $P(t)$. We discretize time and assume that $t$ takes integer values expressed in the appropriate unit, e.g., seconds. We never need a power exceeding $PM^+$, so we can safely assume that $P(t) \leq PM^+$ at any time $t$. We use $b_{m,t}$ as a boolean decision variable which is equal to 1 if machine $m$ is active at time $t$ and 0 otherwise. Then, at any time $t$, the total number of resources used by all machines must remain below $P(t)$, i.e. $\sum_{m \in \mathcal{M}} b_{m,t} \times P \leq P(t)$.

### 3.2 Jobs

We schedule a set of independent jobs $\mathcal{J}$ on the $\mathcal{M}$ parallel machines. Each job $\tau_i \in \mathcal{J}$ is released at date $r_i$, needs $c_i$ cores for execution, and has length $w_i$. We allocate each job $\tau_i$ to a machine $m_i$ at a starting date $s_i$. We use $e_i$ as the (predicted) completion date

of job $\tau_i$. If job $\tau_i$ is not interrupted, we have $e_i = s_i + w_i$. We let $b_{i,m,t}$ be the boolean decision variable which is equal to 1 if job $\tau_i$ is running on machine $m$ at time $t$, and is equal to 0 otherwise. More formally, $b_{i,m,t} = 1 \Leftrightarrow (m_i = m \text{ and } s_i \leq t < e_i)$. We note that a machine $m$ is on at time $t$ if and only if one job is executing: $b_{m,t} = 1 \Leftrightarrow \sum_{\tau_i \in \mathcal{J}} b_{i,m,t} > 0$. As already mentioned, it might happen that a job $\tau_i$ needs to be interrupted, see below. In this case, we let $s_i = e_i = 0$ and re-update these values accordingly whenever the job is rescheduled.

The goal is to schedule all jobs on the machines. More precisely, if $t \in \mathcal{T}$ denotes any time of the whole execution (all jobs), the schedule must verify: (1) $\forall m \in \mathcal{M}, \forall t \in \mathcal{T}, \sum_{\tau_i \in \mathcal{J}} b_{i,m,t} c_i \leq n_c$; and (2) $\forall \tau_i \in \mathcal{J}, s_i \geq r_i$. (1) expresses the constraint on the number of cores on each machine, while (2) simply states that execution cannot start before release time.

### 3.3 Variable Resources

At a given time, a low power capacity may impose to turn some machines off. The jobs currently executing on these machines have to be interrupted immediately, and rescheduled at a further time. Of course, all previous constraints must still be enforced for re-execution. Because we target identical parallel machines, we directly consider the variation in the number of available machines $M_{alive}(t)$ instead of the power variation, using $M_{alive}(t) = \left\lfloor \frac{P(t)}{p n_c} \right\rfloor$. To simulate resource variations, we use a bounded random walk for the number of machines. This walk is defined by a lower bound $M^-$ and a higher bound $M^+$ on the number of machines, a variation period $\phi$, and a variation step *step* on the number of machines. Every $\phi$ time units, the number of machines can decrease or increase by *step* or remain the same, while respecting the constraints that $M_{alive}(t) \in [M^-, M^+]$.

### 3.4 Objective Function

As already mentioned, the objective is to optimize the goodput [2, 23], which measures useful platform utilization by accounting only for jobs that have been completed, so that re-execution does not count. For any time $T$, we say that job $\tau_i \in \mathcal{J}_{comp,T}$ if $e_i \leq T$. Hence $\mathcal{J}_{comp,T}$ is the set of jobs that are complete at time $T$. Similarly, we say $\tau_i \in \mathcal{J}_{started,T}$ if $s \leq T < e_i$ (and $\tau_i$ is not dead at time $T$). At any time $t \in [0, T-1]$, the maximal number of cores that may be turned on is at most $M_{alive}(t) n_c$, otherwise the number of machines turned on would require a power larger than $P(t)$. The total number of units of work that can be executed in $[0, T]$ is at most $\sum_{t \in [0,T-1]} M_{alive}(t) n_c$, Then $\textsc{Goodput}(T)$, the goodput at time $T$, is the fraction of useful work up to time $T$:

$$\textsc{Goodput}(T) = \frac{\sum_{\tau_i \in \mathcal{J}_{comp,T}} w_i c_i + \sum_{\tau_i \in \mathcal{J}_{started,T}} (T - s_i) c_i}{n_c \sum_{t \in [0,T-1]} M_{alive}(t)}.$$

**Complexity–** In the extended version [11], we give two complexity results that show that resource variation dramatically complicates the online scheduling problem.

## 4 ALGORITHMS

Each scheduling algorithm will be defined by its actions on the following four key events that occur during execution: job arrival, job completion, machine addition, and machine removal. Specifically:

- **Job Arrival Event**: When a job is released, a decision is made to decide when to schedule it and on which machine;
- **Job Completion Event**: When a job is completed, its cores are released, possibly allowing for additional jobs to be scheduled;
- **Machine Addition Event**: When a new machine becomes available, a decision must be made on how to utilize it;
- **Machine Removal Event**: When a machine has to be turned off, jobs currently executing on that machine are killed, and a decision is made on how to reallocate them.

Defining a strategy for each of these events fully describes the algorithm's decision-making process in response to changes in resource availability.

### 4.1 FFaware

The baseline heuristic FirstFitAware labels the machines from 1 to $M^+$, and schedules jobs on the machine with the smallest available index that has enough free resources to execute it. Similarly, when a machine needs to be removed, FirstFitAware kills the machine with the highest index:

- **Job Arrival Event**: For each incoming job, FirstFitAware assigns it to the machine with the smallest available index that has enough free resources to execute it. If no machine can execute the job, it is placed in a waiting queue.
- **Job Completion Event**: When a job is completed, FirstFitAware checks the queue for the job with the smallest release date that fits in the machine where the job was completed, and assigns it to this machine. If there are no jobs in the queue or if the machine does not have enough cores available to process any of the waiting jobs, no action is taken. If a job is assigned, FirstFitAware continues to search the queue for additional jobs that can be assigned to the same machine until none fits.
- **Machine Addition Event**: When a machine is added, FirstFitAware examines the queue and assigns jobs to the new machine in order of increasing release date until no further jobs can be assigned or the queue is empty.
- **Machine Removal Event**: When a machine must be removed, FirstFitAware shuts down the machine with the highest index and places all jobs running on that machine in the queue. It then examines the queue and assigns jobs to available machines in order of increasing release date until no further jobs can be assigned or the queue is empty.

FirstFitAware is risk-aware in the sense it maintains an ordered list of machines from left (small indices) to right (large indices). Jobs are mapped to leftmost machines whenever possible, and rightmost machines are those that are shutdown whenever necessary.

### 4.2 FirstFitUnaware

The second baseline heuristic FirstFitUnaware is identical to FirstFitAware except that it selects a random machine to be removed instead of always choosing the machine with the highest index. Comparing both variants will help assess whether a very simple risk-aware approach is more efficient than a traditional approach unaware of power variation.

### 4.3 TargetStretch

Even though FirstFitAware gives priority to machines with lower indices when assigning jobs, jobs running on machines with higher indices may have been running for a long time, and their interruption could result in significant work loss. To address this limitation, TargetStretch schedules smaller jobs on machines that are likely to be turned off after some time; and longer jobs on machines that will never be turned off. Like FirstFitAware, we always turn off the machine with the highest index. However, for job assignment, we make decisions based on job lengths. This requires several new concepts:

• Unlike before, instead of using a single queue for all machines, we have one queue per machine. Specifically, when a job arrives, we choose a machine for it and schedule it on that machine. If the job can start immediately on the machine, we proceed and remember its (expected) end date. Otherwise, we plan for the earliest possible start date for the job, which corresponds to the smallest start-finish interval that contains enough cores to schedule our job. At each time $t$, all jobs planned for this machine have an exact predicted start date $s_i$ and end date $e_i$. If the machine has one or more jobs planned but not yet started (for which $s_i > t$), we say its utilization $u(m)$ is 1. Otherwise, its utilization is the proportion of active cores $u(m) = \frac{c_m^A}{n_c}$. Here, $c_m^A$ denotes the total number of cores used by all the jobs running at time $t$.

• In addition, the number $M_{use}$ of usable, i.e., risk-free, machines at any given time, $M_{use}$, is updated as follows. Initially, we set $M_{use} = M^-$: when a job is assigned to a machine with an index between 1 and $M_{use}$, we take no initial risk. However, it would be a waste not to use the more risky machines at all, if the current set of jobs cannot fit on the risk-free machines. Hence, we update the number of usable machines based on the utilization of the machines $U$ defined as $U = \frac{\sum_{m\in[1,M_{use}]} u(m)}{M_{use}}$. Whenever we have $U > 0.95$, if the number of active machines $M_{alive}(t)$ is greater than $M_{use}$, we increase $M_{use}$. Conversely, if $U$ drops below 0.8, we decrease $M_{use}$ without interrupting the jobs running on the machines whose index is larger than $M_{use}$. This allows us to avoid taking too many risks unnecessarily, while using all machines when needed.

• Finally, we calculate for each job $\tau_i$ its category $C_i$, which is based upon its relative area (defines as $w_i \times c_i$) with respect to a set $\mathcal{J}'$ of other jobs, as $C_i = \frac{\sum_{k\in\mathcal{J}', w_k \geq w_i} w_k c_k}{\sum_{k\in\mathcal{J}'} w_k c_k}$. Here, the set of jobs $\mathcal{J}'$ is chosen as a set of jobs that resemble the set of jobs $\mathcal{J}$ that we are currently scheduling. For example, if we are studying a job trace, we can consider for $\mathcal{J}'$ the set of jobs that were scheduled during the previous week. If a job $\tau_i$ has category $C_i = 0$, it means that it is longer than all the jobs in the reference trace. Conversely, if $\tau_i$ has category $C_i = 1$, it means that it is shorter than all the jobs in the reference trace. Categories will allow us to select the target machine for the jobs. More precisely, we assign job $\tau_i$ to machine

$$M_i^c = \begin{cases} \lfloor C_i M_{use} \rfloor + 1 & \text{if } C_i < 1; \\ M_{use} & \text{otherwise.} \end{cases} \quad (1)$$

• At any given time, we store $S^+$, the maximum stretch obtained so far. Recall that the stretch of a completed job $\tau_i$ is defined as the ratio $\frac{e_i - r_i}{w_i}$ of its response time $e_i - r_i$ (end time minus release time, or time spent in the system) over its length $w_i$. Maximum stretch is preferred to plain response time for fairness to short jobs.

Now, here are the different decision processes for the events:

• **Job Arrival Event**: When a job arrives, we calculate its target machine $M_i^c$, and attempt to schedule it on this machine. If the job can start immediately, or is scheduled such that its estimated stretch does not exceed the maximum stretch $S^+$, we do schedule it on $M_i^c$. Otherwise, we choose the machine that can provides its earliest start time among all machines whose index does not differ more than $D^+$ of that of $M_i^c$. In other words, we bound the distance from $M_i^c$ to explore alternate target machines.

• **Job Completion Event**: When a job completes, no action is required since we have already scheduled the next jobs on each machine. However, since the number of cores of the machine changes, we need to update the total utilization of the machines and potentially change $M_{alive}$ if necessary. We may also need to update $S^+$.

• **Machine Addition Event**: When a machine is added, no action is required unless $M_{alive}$ corresponds to the previous number of machines and the total utilization is greater than 95%. In this case, we update $M_{alive}$ and place all jobs that are in the waiting lists of machines and have not started into a global waiting list, and we re-allocate them: we process these jobs in ascending order of release dates $r_i$ and allocate them to a machine using the same procedure as described above for job arrivals (which amounts to considering them as newly submitted jobs for mapping).

• **Machine Removal Event**: When a machine is removed and jobs are interrupted, we recalculate $M_{use}$, and reallocate all pending jobs as described in the previous point.

### 4.4 TargetASAP

TargetStretch ensures that those machines that get killed only contain jobs among the shortest ones. However, some machines may be under-utilized if few jobs target them initially. Indeed, if the maximum stretch $S^+$ is very high, the algorithm will always favor the target machine, and thus could neglect a machine very close in terms of index that has idle cores, which is bad for goodput. For this reason, we have developed a second algorithm that differs from the previous one only in one aspect: when a job arrives, instead of scheduling it on its target machine $M_i^c$ if its stretch is not higher than the maximum stretch, we proceed as follows:

• **Job Arrival Event:** If the job can start immediately upon arrival on its target machine $M_i^c$, we launch it there. Otherwise, we go through all machines whose index is at a distance smaller than $D^+$ and look for the closest machine to our target machine on which the job can run immediately, if one exists. If all machines around are full or do not have enough cores to run the job, we check whether the job can be scheduled on the target machine without increasing the maximum stretch $S^+$. If it is indeed possible, we schedule the job on the target machine. Otherwise, we schedule it on the machine that can start it at the earliest time among those within an acceptable distance.

• **Any Other Event:** Same as TargetStretch.

### 4.5 PackedTargetASAP

While TargetASAP solves the issue of under-utilized machines, it may leave some machines partially empty, in contrast to FirstFitAware that fills the machines perfectly by always using them

in ascending order. For example, if jobs only use 60% of the available cores, FirstFitAware would use approximately 60% of the machines, whereas TargetASAP may use all the machines each at at 60% capacity. Furthermore, if a new job arrives and requires a large number of cores, TargetASAP may not be able to start it immediately while FirstFitAware could.

To address this issue, we group machines into packs. Instead of defining the target machine as $M_i^c$ as in Equation (1), we will round it to the nearest multiple of 5. The 5 machines corresponding to a pack are filled one after the other using $D^+$ instead of all at once: e.g., if three jobs were assigned to machines 0, 1, and 2 by TargetASAP, all three jobs will be assigned to machine 0 by PackedTargetASAP, leaving machines 1 and 2 available for future jobs.

# 5 EXPERIMENTAL SETUP

We conduct experiments using an in-house simulator, both on synthetic traces and on actual workflow traces. An instance of the simulation consists of a combination of two traces, a resource variation trace that represents the number of machines alive at any given time, and a job trace. We generate multiple simulation traces using the method given below. The complete setup and full experimental results are available in [11].

## 5.1 Resource Traces

The generation of the resource variation trace takes three parameters: $M_{avg}$, $M_{ra}$, and $\phi$. We start the resource variation trace at time 0 and end it at time $T_{end}$, where $T_{end}$ is three weeks. The window begins at time $T_{begin}$ after one week, so that the first week is a warmup. The number of machines follows a random walk that evolves periodically, with period $\phi$, hence this number is a constant within each period and changes only at the end of a period. Specifically, the average number of machines is $M_{avg}$. The total number of available machines always stays within the range $[M_{avg}-M_{ra}, M_{avg}+M_{ra}]$. It evolves randomly, staying constant, increasing or decreasing with equal probability unless one bound of the range is reached. In the latter case, the number of machines either stays constant or evolves in the unique possible direction, with same probability. Changes in the number of available machines always involve $step = \lfloor \frac{M_{ra}}{4} \rfloor$ machines, hence *step* controls the magnitude of resource variation from one period to the next.

## 5.2 Job Traces: Borg

We experiment on traces of workloads running on Google compute cells that are managed by the cluster management software internally known as Borg [6]. We cut the Borg trace into slices of windows $[T_{begin}, T_{end}]$ of length 2 weeks. Inside each window, we keep the jobs $\tau_i$ for which $r_i \in [T_{begin} - w_i, T_{end}]$. We assume that the jobs $\tau_i$ that have been released before the beginning of the window ($r_i < T_{begin}$) have actually been running since their release date, hence job $\tau_i$ has $w_i - (T_{begin} - r_i)$ units of work remaining at time $T_{begin}$. Several jobs in the Borg trace are permanently running, we ignore them (or assume that these jobs execute on specific risk-less machines), and we focus on jobs that are lasting less than a day. Finally, we prune the traces so that the total work hours do not exceed the maximum capacity of 26 machines, each with 24 cores, running during 2 weeks with full peak load. This is to match the dimensioning of the experimental window.

## 5.3 Job Traces: Synthetic Traces

A synthetic trace consists of a set of jobs $\mathcal{J}$ of $n$ jobs, where each job $\tau_i$ is defined by the three parameters $(r_i, c_i, w_i)$, where $r_i$ is its release date, $c_i$ its number of cores, and $w_i$ its length.

*Release dates $r_i$* Similarly to the Borg trace, we study a two-week window. Because we want to start with a steady state, we add one week before the start of the window to generate jobs, therefore we assume the trace starts at time 0, and finishes at time $T_{end}$ corresponding to 3 weeks, with $T_{begin}$ corresponding to 1 week. For the beginning of the window at date $T_{begin}$, we will again assume that all jobs released at time $r_i < T_{begin}$ have been running for $T_{begin} - r_i$ units of time, for the jobs verifying $w_i > T_{begin} - r_i$. We assume the jobs get released regularly, therefore if we generate a trace of $n$ jobs for a total window $[0, T_{end}]$, one job will be released each $T_{end}/n$ units of time. Therefore, $r_i = \frac{i}{n} T_{end}$.

*Number of cores $c_i$* The number of cores per jobs is drawn randomly, following roughly the distribution of the Borg traces, more specifically $\forall \tau_i \in \mathcal{J}, \mathbb{P}\{c_i = 1\} = \frac{1}{6}, \mathbb{P}\{c_i = 2\} = \frac{1}{3}, \mathbb{P}\{c_i = 4\} = \frac{1}{3}, \mathbb{P}\{c_i = 8\} = \frac{1}{6}$.

*Length $w_i$* Depending on the workflow type, the execution time of jobs is generated using different probability distributions. Similarly to Borg traces, the average job length $T_{avg}$ is defined accordingly to $n$, so that the total number of machines required to process all jobs if there were no resource variations and if all machines were always used at maximum capacity is around 26 (e.g. such that the total core hour is around 209664, which means $T_{avg} = 209664/n$ hours). We create jobs along three different workflow types:

- For SyntheticUniform, we generate the length of the jobs uniformly in $[0, 2T_{avg}]$, so that their average length is around $T_{avg}$.
- For SyntheticLogScale, we draw the category $c$ as a random integer in $[1, 4]$ such that $\mathbb{P}\{c = 1\} = 4\mathbb{P}\{c = 2\} = 4\mathbb{P}\{c = 3\} = 4\mathbb{P}\{c = 4\}$, then we draw the length uniformly in $[5^{c-1}K, 5^cK]$, where $K$ is chosen so that the expected length of this random variable matches $T_{avg}$. Because $c$ is not drawn uniformly, all categories have a non-negligible impact on the total core hours of the trace, although the category with the highest length is the most significant. We also experimented drawing $c$ uniformly in $[1, 4]$ so that the longest jobs have a more significant impact. This version is called SyntheticLogScaleU; the results are very close to SyntheticUniform; most of them are available only in the extended version [11].
- For Synthetic3types, we generate three types of jobs of three different length, $t_{short}$, $t_{middle}$ and $t_{high}$, such that $t_{high} = 3t_{middle} = 9t_{short}$. We generate the jobs so that the total work hours of these three type of jobs is equal, i.e., $\mathbb{P}\{w_i = t_{short}\} = 3\mathbb{P}\{w_i = t_{middle}\} = 9\mathbb{P}\{w_i = t_{high}\}$. Therefore, we get $t_{short} = \frac{13}{27} T_{avg}$, $t_{middle} = \frac{39}{27} T_{avg}, t_{high} = \frac{117}{27} T_{avg}, \mathbb{P}\{w_i = t_{high}\} = \frac{1}{13}, \mathbb{P}\{w_i = t_{middle}\} = \frac{3}{13}$, and $\mathbb{P}\{w_i = t_{short}\} = \frac{9}{13}$.

## 5.4 Experimental Parameters

An instance has four parameters:

- The average number of machines $M_{avg} = 24$, to be slightly below the number of machines required in average;
- The period of machine variation $\phi = 1200s$, corresponding to one

change every 20 minutes;
• The range of machine variation $M_{ra} = 8$, therefore the machines will always be in $[M^- = 16, M^+ = 32]$; half the machines are safe;
• The number of cores per machine $n_c = 24$.
One last parameter for the synthetic traces is the number of jobs $n = 20000$. The values above are the ones used by default.

We further study the impact of each parameter separately. Due to lack of space, we detail experimental results only when varying the number $M_{avg}$ of machines, and the period $\phi$ and range $M_{ra}$ of machine variation. See the extended version [11] for varying the number $n_c$ of cores per machine; and the number $n$ of synthetic jobs. Specifically, we consider:
• $M_{avg} \in [20, 22, 24, 26, 28]$ (with $\phi = 1200s$, $M_{ra} = 8$ and $n_c = 24$). This experiment is to explore what happens if the number of machines is too small to process the workload, or, in contrary if there are (in average) enough machines to process every job;
• $\phi \in [400, 1200, 3600, 10800, 32400]$ (between 7 minutes and 9 hours, with $M_{avg} = 24$, $M_{ra} = 8$ and $n_c = 24$). The average number of available machines remains the same, but the number of job interruptions is likely to decrease when $\phi$ increases;
• $M_{ra} \in [4, 6, 8, 12, 16]$ (with $M_{avg} = 24$, $\phi = 1200s$, and $n_c = 24$): in the most extreme scenarios, only 8 machines are safe while we could have up to 40 machines available.
For each set of parameters, we run each heuristic under six 2-week traces per workflow type, each of them with 30 different variation traces. Results are shown using boxplots where the average is represented by a star, the boxes show the 25th to 75th percentiles, and the whiskers indicate the 10th and 90th percentiles.

Finally, while Goodput remains the major focus, we also report the performance of each heuristic for three other metrics:
• MaximumStretch: Recall that the stretch of a completed job $\tau_i \in \mathcal{J}_{comp,T}$ is defined as the ratio $\frac{e_i - r_i}{w_i}$ of its response time $e_i - r_i$ (end time minus release time, or time spent in the system) over its length $w_i$. The maximum stretch corresponds to $S^+$ defined earlier taken at time $T_{end}$, e.g., $S^+ = \max_{\tau_i \in \mathcal{J}_{comp,T_{end}}} \left( \frac{e_i - r_i}{w_i} \right)$.
• AbortedVolume is defined as the total amount of core hours lost because of job interruptions, normalized by the total amount of core hours that were available. More precisely, if we define a family of events corresponding to all interruptions $\mathcal{I}$, where each interruption $i_k \in \mathcal{I}$ corresponds to a time $t_k$ and a related job $\tau_k$ that started at time $s_k$ with $c_k$ cores, then $\text{AbortedVolume} = \frac{\sum_{i_k \in \mathcal{I}} (t_k - s_k) c_k}{n_c \sum_{t \in [0, T-1]} M_{alive}(t)}$. We always have Goodput + AbortedVolume $\leq 1$, because we have normalized by the total core hours of work available during the processing. This sum may be lower than 1 because cores are sometimes idle.
• AverageAbortedTime is the average time lost at each interruption, defined as $\frac{\sum_{i_k \in \mathcal{I}} (t_k - s_k)}{card(\mathcal{I})}$.
Two more metrics are addressed in the extended version [11].

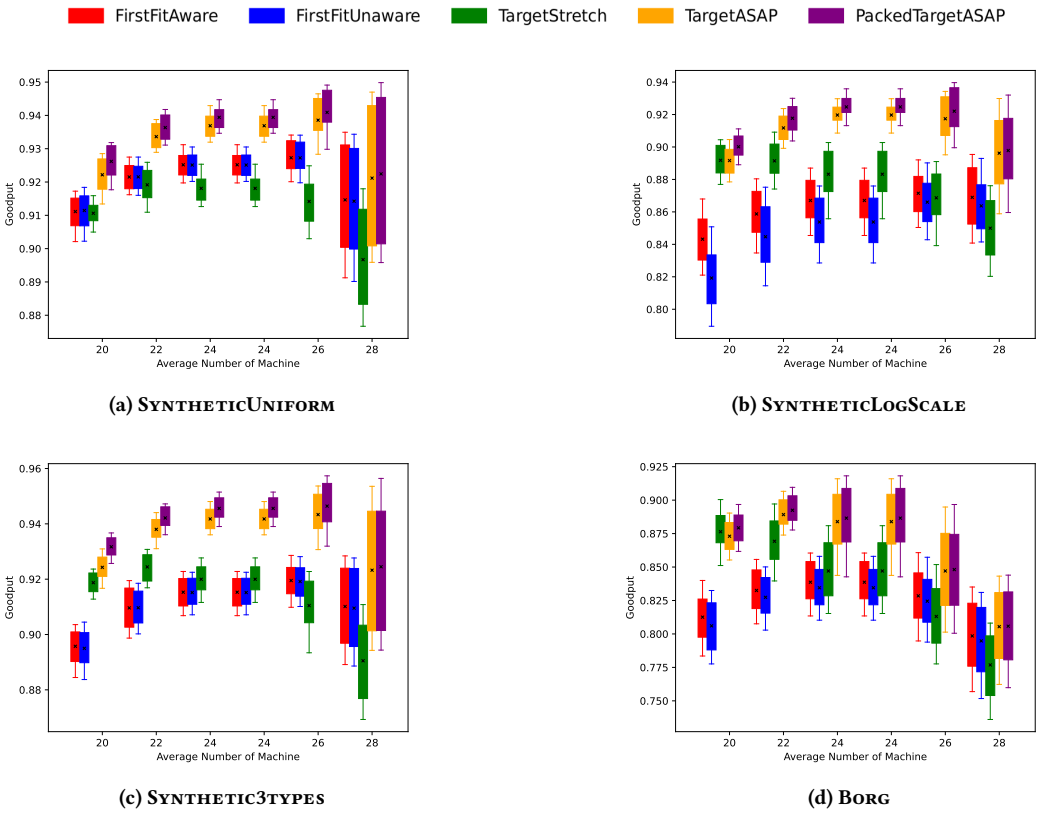
## 6 EXPERIMENTAL RESULTS

**1. Varying the number $M_{avg}$ of machines** Figure 1 shows the results of all five heuristics for the Goodput metric when the average number of machines $M_{avg}$ vary. First, note that Goodput corresponds to the total work successfully executed divided by the number of available core hours. Therefore, we compute the proportion of machines used over time, rather than the total amount of work done. This explains why the results of all the heuristics decrease between 26 and 28 average machines in terms of goodput. For $M_{avg} = 28$, even if all jobs were completed (which is almost the case for TargetASAP and PackedTargetASAP), the goodput would not be high because not all machines are fully utilized. The increasing portion between 20 and 26 machines can be explained by the fact that $M_{ra}$ is fixed. Thus, the proportion of machines experiencing variability is higher for a low number of average machines, resulting in more aborted work.

Regarding the differences between the heuristics, we observe that the TargetASAP and PackedTargetASAP heuristics are always better than the two competitors, FirstFitAware and FirstFitUnaware. In fact, this observation is over all the experimental results, including those in [11]. The overall trend of TargetASAP, PackedTargetASAP, FirstFitAware, and FirstFitUnaware is similar because these four heuristics share the characteristic of having a good allocation of jobs on the machines, with cores rarely being idle when jobs are waiting. The difference in results is due to the fact that FirstFitAware and FirstFitUnaware do not specifically preserve the longest jobs, which leads to more interrupted work. TargetStretch has a different behavior because even though it takes into account the length of jobs that need to be preserved, it lacks flexibility, and some machines tend to remain partially inactive even when jobs are waiting. This phenomenon increases as the number of machines grows, because the likelihood of having discrepancy between machines increases with the number of jobs.

Finally, TargetASAP and PackedTargetASAP have fairly similar results, with a slight advantage for PackedTargetASAP. Similarly, FirstFitAware and FirstFitUnaware have fairly similar results (except for SyntheticLogScale where FirstFitUnaware is much lower). While this may seem surprising at first, it can be explained by the fact that when there are too many jobs for the number of machines, they are all either turned off or saturated; then, not knowing which machine will be turned off first is not penalizing. In the opposite case, there are enough machines available to support interruptions and re-executions. The workflow type generally has a limited impact on the relative performance of the heuristics, both for synthetic traces or for Borg. There are some minor relative performance details, e.g. TargetStretch is at the same level or even better than TargetASAP for SyntheticLogScale and Borg, while it is below FirstFitAware and FirstFitUnaware for SyntheticUniform. Altogether, the differences in goodput between the heuristics differ from a workflow type to another, from around 2% for SyntheticUniform and Synthetic3types and up to almost 10% for Borg, showing that the potential gain is substantial. The heuristics preserving long jobs should improve the Goodput in many practical scenarios.

**2. Varying the period $\phi$ of machine variation** Figure 2 shows the results of all five heuristics for the Goodput metric when the period $\phi$ of machine variation varies. Once again, we observe in this figure that the impact of the workflow type seems limited since the four figures are generally similar, except for the scale of the y-axis. For FirstFitAware and FirstFitUnaware, the goodput can reach values lower than 75% for SyntheticUniform, while it remains above 84% for Synthetic3types and Borg.




(a) SyntheticUniform (b) SyntheticLogScale

(c) Synthetic3types (d) Borg

**Figure 1: Goodput when varying the number of machines $M_{avg}$.**

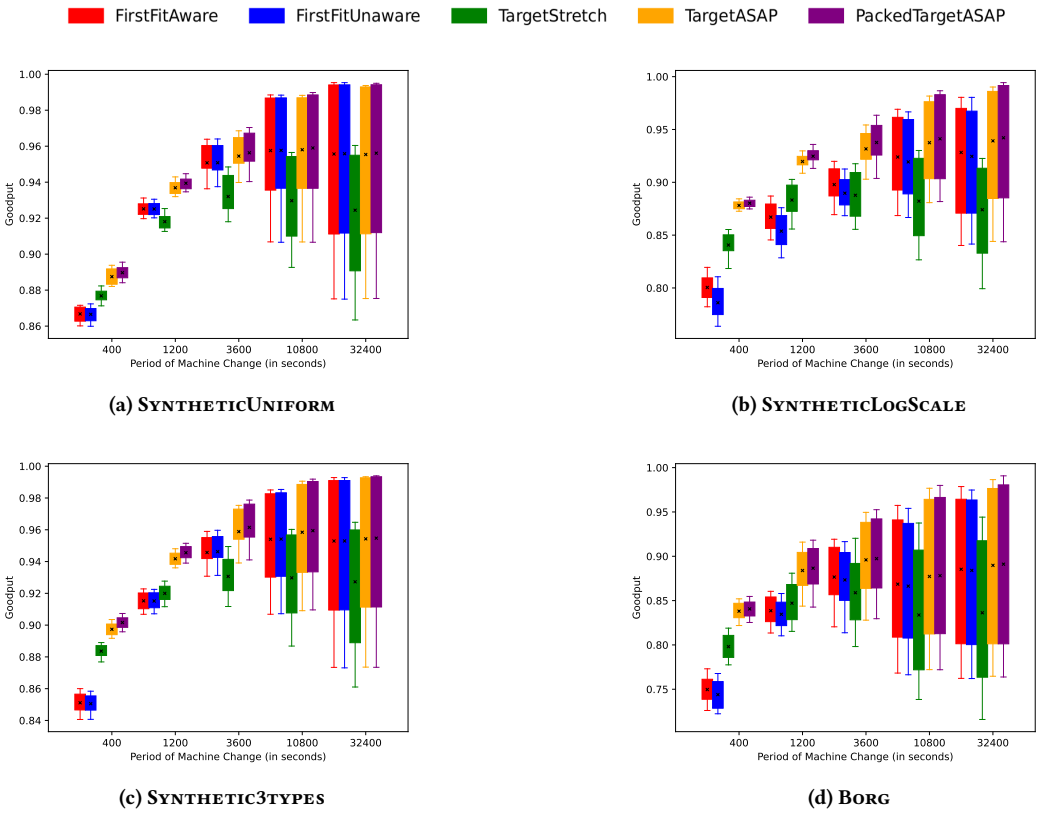
The goodput generally increases with the period $\phi$. This is logical since the fewer machine changes, the lower the risk of interruption, even if the average number of machines remains unchanged. We also observe a strong increase in variability for a given heuristic and set of parameters when the period is high. This is due to the fact that the goodput is scaled by the total available cores. The longer the period, the fewer states the random walk of the number of machines will take, and therefore the average of this specific random walk will be further away from its statistical average. In other words, there is a high variability in total available core hours: if the random walk stays with high values of machine numbers, all jobs can be executed; but since the system load is fixed, the goodput will be reduced. Finally, we note that the relative performance of TargetStretch improves when the period is low. Indeed, the lower the period, the more interesting it is to be safe and preserve long jobs. Conversely, the higher the period, the more problematic its shortcomings on the overall quality of the schedule.

**3. Varying the range $M_{ra}$ of machine variation–** Figure 3 shows the results of all five heuristics for the Goodput metric when the range of machine variations $M_{ra}$ vary. Of course, the higher the machine range $M_{ra}$, the lower the goodput, since more jobs are interrupted. We observe this for all types of workflows and for all heuristics, which generally maintain their relative performance. TargetStretch is slightly less impacted by the increase in range, again because it is scheduling more safely than the other heuristics.

**4. Exploring other metrics–** Figure 4 shows the results of all five heuristics for Borg when the range $M_{ra}$ of machine variation varies, and for different metrics: Goodput (reproduced from Figure 3 for convenience), MaximumStretch, AbortedVolume, and AverageAbortedTime. First and as expected, the results for all metrics are degraded by increasing the range $M_{ra}$ for each of the five heuristics. While TargetStretch is clearly worse than TargetASAP and PackedTargetASAP for the goodput, it turns out to be the best heuristic for MaximumStretch. Indeed, it is the heuristic that best preserves long jobs, which is reflected both in the total volume aborted AbortedVolume and in the average time lost per interrupted job AverageAbortedTime, where TargetStretch achieves the lowest values. Obviously, FirstFitAware and FirstFitUnaware, which have no consideration for job length, perform poorly for these metrics.

We study these metrics in more details in [11]. Overall, TargetStretch is not always better than TargetASAP and PackedTargetASAP for the maximum stretch: this depends on the experiments performed. On the contrary, the basic heuristics FirstFitAware and FirstFitUnaware are always the ones that perform the worst for all other metrics. We observe that the difference in maximum stretch is not at all negligible, with a factor of 2 to 3 compared to TargetASAP, PackedTargetASAP, and TargetStretch.

It is also worth studying the sum of Goodput and AbortedVolume. For FirstFitAware, FirstFitUnaware, TargetASAP, and




(a) SyntheticUniform (b) SyntheticLogScale

(c) Synthetic3types (d) Borg

**Figure 2: Goodput when varying the period $\phi$ of machine variation.**

PackedTargetASAP, this sum is close to 1 generally, which means that the machines are often used to their maximum capacity. This is not the case for TargetStretch, which, for example, is well below TargetASAP and PackedTargetASAP for both metrics, which means that there is a frequent under-utilization of machines, even though it better preserves long jobs.

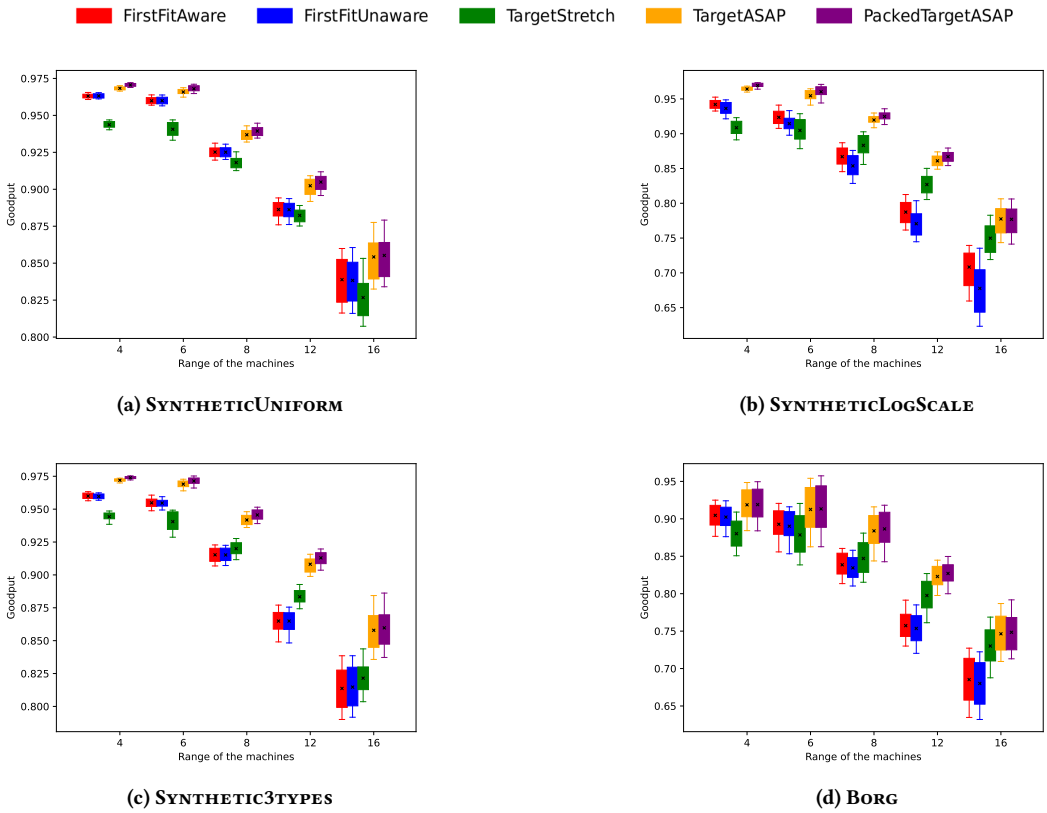
**Summary–** In summary, there are scenarios for which TargetStretch can be a decent heuristic, for instance when the number of machines is low or the variability is high. In this case, it is necessary to be very careful in sorting jobs by length in order to preserve the longest ones and not loose too much work due to interruptions. However, the lack of flexibility of TargetStretch can be problematic when the impact of machine variation is not critical, which is the case for most of the experiments we have done. Then, although it was designed to preserve the maximum stretch MaximumStretch, its performance is generally comparable to TargetASAP and PackedTargetASAP for this metric.

Furthermore, TargetASAP and more particularly PackedTargetASAP (which is slightly better) offer a greater flexibility. Hence, even if they are slightly less precise for preserving the longest jobs, they allow for an excellent overall utilization of machines and a good preservation of the longest jobs, which makes them the best heuristics in almost all scenarios. They are better than their standard competitors FirstFitAware and FirstFitUnaware, in all cases. The superiority of TargetASAP and PackedTargetASAP is significant: up to 10% increase in goodput and a maximum stretch two to three times smaller. We conclude that the use of FirstFitAware and FirstFitUnaware should be reconsidered with variable resources.

## 7 CONCLUSION AND FUTURE WORK

With growing variation in power cost, availability, and carbon-intensity, driven by integration of more renewable generation to the power grid, the incentives to operate datacenters as variable loads, producing variable computing capacity are growing. Resource management (schedulers) must be advanced to handle large-scale and perhaps increasingly frequent capacity variation, yet achieve high utilization of the available capacity. The primary challenge is that when capacity decreases, running jobs may need to be terminated to meet the required power load reduction.

We present online risk-aware scheduling strategies to preserve performance in this variable capacity environment. Specifically, we design novel risk-aware scheduling and mapping algorithms that assign the right job to the right machine, optimizing for the system's goodput. Our algorithms employ a variety of techniques to mitigate the impact of resource variation, including maintaining a risk index per machine, mapping longer jobs to safer machines, maintaining local queues at machines, re-executing interrupted jobs on new machines, and redistributing pending jobs as resource capacity




(a) SyntheticUniform (b) SyntheticLogScale

(c) Synthetic3types (d) Borg

**Figure 3: Goodput when varying the range $M_{ra}$ of machine variation.**

increases. Our assessment using workload trace from Google's Borg system and three synthetic traces shows significant gains over first-fit algorithms with up to 10% increase in goodput, with no loss in complementary metrics, such as the maximum and average stretch. We conclude that standard first-fit algorithms are insufficient for future variable capacity environments and require re-design in order to maintain the expected level of performance.

Directions for future work are bountiful; they include exploration of different workloads, different job execution models (e.g. migration or deferral), different variation models, and the recent interesting direction of malleable workloads.

## ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful reviews. This work is supported in part by NSF Grants CMMI-1832230, OAC-2019506, CNS-1901466, and the VMware University Research Fund. We gratefully acknowledge the support of the UChicago FACCTS program, that encourages and supports collaboration between researchers at the University of Chicago and the nation of France.

## REFERENCES

[1] Farag Azzedin and Muthucumaru Maheswaran. 2002. Integrating trust into Grid resource management systems. In *31st International Conference on Parallel Processing (ICPP)*. IEEE Computer Society, 47– 54.

[2] Jim Basney and Miron Livny. 1999. Improving Goodput by Coscheduling CPU and Network Capacity. *IJHPCA* 13, 3 (1999), 220–230.

[3] Anton Beloglazov and Rajkumar Buyya. 2010. Energy Efficient Resource Management in Virtualized Cloud Data Centers. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. 826–831. https://doi.org/10.1109/CCGRID.2010.46

[4] Michael A. Bender, S. Muthukrishnan, and Rajmohan Rajaraman. 2002. Improved Algorithms for Stretch Scheduling. In *Proc. of the 13th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'02)*. SIAM, 762–771.

[5] United States National Science Foundation. 2022. Design for Environmental Sustainability in Computing (DESC). https://beta.nsf.gov/funding/opportunities/design-environmental-sustainability-computing-desc.

[6] Google team. 2019. Borg cluster workload traces. https://github.com/google/cluster-data.

[7] Green ICT. 2009. Follow the Sun, Wind, Moon. https://www.vertatique.com/cloud-computing-starting-follow-sunwindmoon.

[8] Udit Gupta, Mariam Elgamal, Gage Hills, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2022. ACT: Designing Sustainable Computer Systems with an Architectural Carbon Modeling Tool. In *49th Int. Symp. Computer Architecture (ISCA)*. ACM.

[9] Nicola Jones. 2018. How to Stop Data Centres from Gobbling up the World's Electricity. *Nature* (September 2018).

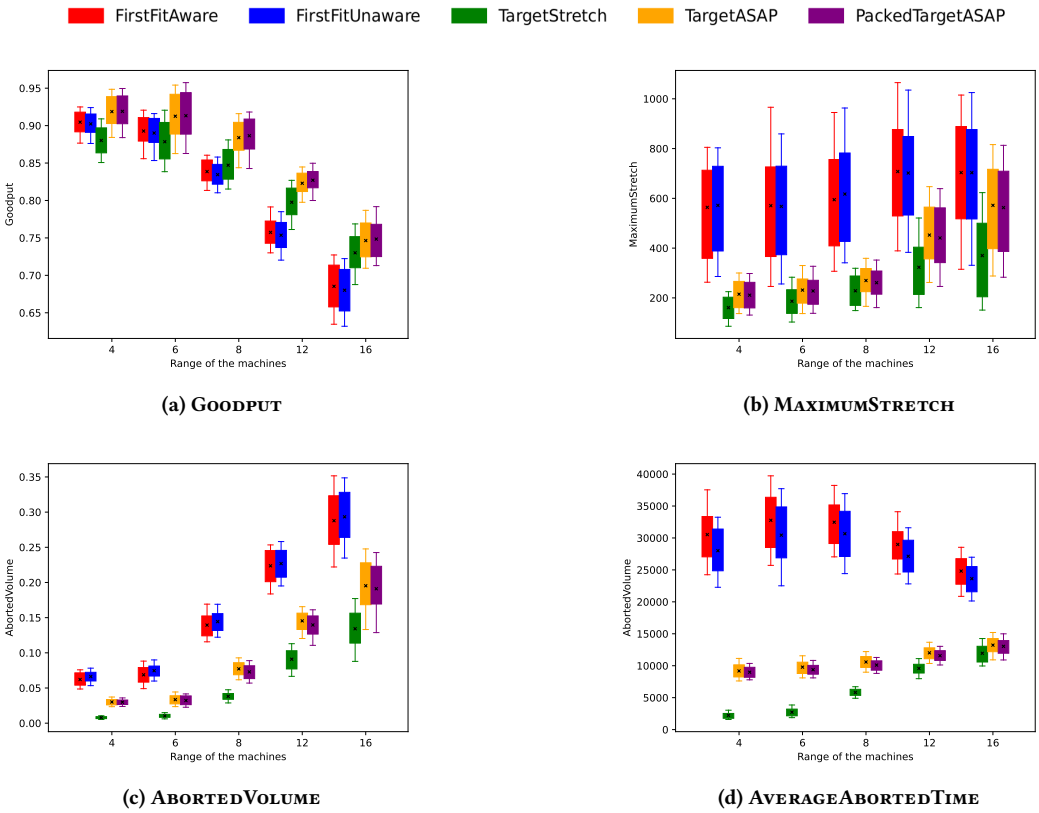
[10] Tirthak Patel, Zhengchun Liu, Raj Kettimuthu, Paul Rich, William Allcock, and Devesh Tiwari. 2020. Job characteristics on large-scale systems: long-term analysis, quantification, and implications. In *SC'20*. IEEE.

[11] Lucas Perotin, Chaojie Zhang, Rajini Wijayawardana, Anne Benoit, Yves Robert, and Andrew A. Chien. 2023. Extended version and source code. https://zenodo.org/record/8139439.

[12] T Pesch, S Schröders, H J Allelein, and J F Hake. 2015. A new Markov-chain-related statistical approach for modelling synthetic wind power time series. *New Journal of Physics* 17, 5 (2015).

[13] Ana Radovanovic. 2020. Our data centers now work harder when the sun shines and wind blows. https://blog.google/inside-google/infrastructure/data-centers-work-harder-sun-shines-wind-blows.

[14] S. Song, Kai Hwang, and Yu-Kwong Kwok. 2006. Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling. *IEEE Trans. Comput.* 55, 6 (2006), 703–719. https://doi.org/10.1109/TC.2006.89




(a) Goodput (b) MaximumStretch

(c) AbortedVolume (d) AverageAbortedTime

**Figure 4: Different metrics to analyze the results on Borg for varying range of machine variation $M_{ra}$.**


[15] Jason Sonnek, Abhishek Chandra, and Jon Weissman. 2007. Adaptive Reputation-Based Scheduling on Unreliable Distributed Infrastructures. *IEEE Transactions on Parallel and Distributed Systems* 18, 11 (2007), 1551–1564. https://doi.org/10.1109/TPDS.2007.1094

[16] Tal Rosenberg. 2023. The Computer that will Change Everything. *Chicago Magazine* (January 2023). https://www.chicagomag.com/chicago-magazine/february-2023/the-computer-that-will-change-everything/.

[17] United Nations Framework Convention on Climate Change. 1997. Kyoto Protocol. http://unfccc.int/kyoto_protocol/items/2830.php.

[18] United Nations Framework Convention on Climate Change. 2015. Paris Climate Change Conference. http://unfccc.int/meetings/paris_nov_2015/meeting/8926.php.

[19] Douglas J White. 1993. A survey of applications of Markov decision processes. *Journal of the operational research society* 44, 11 (1993), 1073–1096.

[20] Fan Yang and Andrew A. Chien. 2016. ZCCloud: Exploring Wasted Green Power for High-Performance Computing. *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2016), 1051–1060.

[21] Fan Yang and Andrew A. Chien. 2017. Large-scale and Extreme-Scale Computing with Stranded Green Power: Opportunities and Costs. *IEEE Transactions on Parallel and Distributed Systems* 29, 5 (December 2017).

[22] Reihaneh Haji Mahdizadeh Zargar and Mohammad Hossein Yaghmaee Moghaddam. 2020. Development of a Markov-Chain-Based Solar Generation Model for Smart Microgrid Energy Management System. *IEEE Transactions on Sustainable Energy* 11, 2 (2020), 736–745.

[23] Chaojie Zhang and Andrew A. Chien. 2021. Scheduling Challenges for Variable Capacity Resources. In *Job Scheduling Strategies for Parallel Processing*. Springer, 190–209.