Cluster Computing manuscript No.
(will be inserted by the editor)

MILP Formulations for Spatio-Temporal Thermal-Aware Scheduling in Cloud and HPC Datacenters

Jean-Marc Pierson · Patricia Stolf · Hongyang Sun · Henri Casanova

Submitted: August 31st, 2018, Received: date / Accepted: date

Abstract This paper focuses on scheduling problems related to the execution of computational jobs in datacenters with thermal constraints. Mixed integer linear programming (MILP) formulations are proposed that encompass both spatial and temporal aspects of the temperature evolution under a unified model. This model takes into account the dynamics of heat production and dissipation in order to schedule jobs at appropriate times on appropriate machines. The proposed MILP formulations are applicable to both high-performance computing (HPC) and Cloud settings, and can target several objectives including energy and makespan minimization, while incorporating the cooling costs and dynamic voltage and frequency scaling (DVFS) capabilities of servers. The applicability and usefulness of our formulations are demonstrated via several HPC and Cloud case-studies.

Keywords HPC and Cloud datacenters · thermal modeling · thermal-aware scheduling · makespan · energy consumption · linear programming

1 Introduction

In this paper, we consider the problem of scheduling computational jobs in datacenters with both energy consumption and application performance objectives, while enforcing constraints on heat production. This represents an important problem in datacenter optimization as cooling constitutes a significant part of the total energy consumption in today's datacenters [7], [14]. Effective thermal management to prevent hotspots and server overheating also plays an critical role in ensuring the application performance [7], [27]. Although similar problems have been considered in the literature (see Section 2 for a review of related work), to the best of our knowledge, thermal-aware scheduling has not been formalized as a generally applicable constrained optimization problem, which is the goal of this paper.

While prior works have proposed thermal-aware scheduling algorithms on servers with individual and steady-state temperature constraints, we argue that a formulation based on both spatial and temporal thermal models at the entire datacenter level is needed. More specifically, a thermal model should account for heat recirculation within a datacenter (i.e., spatial dispersion of heat between servers) and temperature evolution throughout time (e.g., temperature increases as computation is being performed). Such behaviors have been modeled in the literature using air flows and spatial locations of servers (e.g., [19], [34], [1]), based on the physical characteristics (thermal capacitance and resistance) of the processors (e.g., [28], [29], [2]), as well as holistically linking both temporal and spatial properties (e.g., [32], [11]). Moreover, these models have been validated by several studies (e.g., [29], [34], [25], [21], [1]) using computational fluid dynamics (CFD) simulations on cyber-physical systems, and thus provide reliable means of modeling the temperature

Jean-Marc Pierson and Patricia Stolf
IRIT, University of Toulouse, CNRS, INPT, UPS, UT1, UT2J, France. E-mail: jean-marc.pierson@irit.fr, patricia.stolf@irit.fr

Hongyang Sun
Vanderbilt University, Nashville, TN, USA. E-mail: hongyang.sun@vanderbilt.edu

Henri Casanova
University of Hawai at Manoa, HI, USA. E-mail: henric@hawaii.edu

evolution of servers in datacenters and of studying related scheduling problems. The objective of this paper is to develop a formulation for the optimization problem of job scheduling based on these models.

Since most resource allocation problems can be framed as constrained optimization problems with both integer and real (in practice rational) variables, we formulate the problem using mixed integer linear programming (MILP). Integer variables are needed to encode the assignment of jobs to compute resources. Due to the presence of these integer variables, computing optimal solutions based on MILP formulations is often infeasible in polynomial time. Nevertheless, MILP formulations are still useful due to several reasons: (1) They can be solved for small problem instances, making it possible to assess the efficacy of a polynomial heuristic, which can then be used to solve large problem instances; (2) Most MILP solvers return an upper bound on the distance-to-optimal of returned solutions. Therefore, even though an optimal solution may not be found for a large instance, in practice a solver can return a solution that is close to optimal and quantifiably so; (3) MILP formulations can be relaxed by making all variables rational, and then solved in polynomial time in practice (even though computing the optimal solution is still NP-complete in theory). The solution to the relaxed formulation is not feasible, but it can serve as a starting point to construct a feasible solution, e.g., via rounding off rational variables to integers [30].

In this paper, by relying on a spatio-temporal thermal model and MILP formulations, we propose a unified framework for thermal-aware scheduling of computational jobs for both Cloud and HPC datacenters. This is by contrast with previously proposed formulations, which are typically valid in one particular setting [7]. The proposed formulation can be used to optimize a wide range of relevant performance and energy objectives under datacenter-wide thermal or placement constraints while incorporating cooling costs and dynamic voltage and frequency scaling (DVFS) capabilities of servers.

The following summarizes the main contributions of this paper:

- We employ an analytical thermal model of datacenters that takes into account both spatial and temporal temperature behaviors;
- We propose MILP formulations for several scheduling problems with various objective functions based on the thermal model;
- We demonstrate the applicability and usefulness of our formulations in several case-studies in both HPC and Cloud settings.

The remainder of this paper is organized as follows: Section 2 discusses related work. Section 3 gives a high-level description of our problem statement. Section 4 details our thermal models and how they capture relevant datacenter heat management concerns. Section 5 gives our MILP formulations for several relevant job scheduling problems. Section 6 presents quantitative results for an HPC and Cloud case study and presents a comparison between the optimal MILP solution and those from two heuristics. Finally, Section 7 concludes with a brief summary of our findings and perspectives on future work.

2 Related Work

In this section, we review the literature on scheduling for Cloud and HPC datacenters. We first review works that have proposed MILP formulations for energy-aware scheduling problems. We then discuss works that have proposed thermal models of datacenters. Finally, we review works that have proposed scheduling heuristics that take thermal considerations into account.

2.1 MILP Formulations for Energy-Aware Resource Allocation and Scheduling Problems

MILP formulation, due to its usefulness, has been proposed in many previous works to solve resource allocation and/or job scheduling problems.

Borgetto et al. [5] have studied energy-aware resource allocation for HPC jobs in datacenters. They have proposed MILP formulations for several multi-objective optimization problems, including maximizing job performance under power consumption constraints, minimizing power consumption under job performance constraints, as well as optimizing a linear combination of both objectives. Kantarci et al. [12] have considered Virtual Machine (VM) allocation for cloud datacenters interconnected via a backbone network. An MILP formulation has been proposed for backbone topology virtualization and VM placement with the objective of minimizing power consumption. Sharrock et al. [26] have proposed an MILP formulation for striking a desirable compromise between the energy cost of network equipment in a datacenter and the quality of service provided to applications. Gu et al. [9] have applied

an MILP formulation to the problem of minimizing the total energy cost in green cloud datacenters. Given an energy budget, requests are scheduled on different servers and sources by accounting for time-varying and location-varying electricity prices as well as renewable energy options. Barkat and Capone [3] have considered using energy storage technologies (i.e., batteries) as a green energy source to reduce carbon emissions from datacenters. They have considered geographically distributed cloud infrastructures using batteries for energy storage, and proposed an MILP formulation for computing an optimal configuration that considers service scenarios, storage capacities, as well as the energy consumed to route requests to different datacenter locations. Haque et al. [10] have proposed GreenPar, a scheduler for HPC jobs in datacenters partially powered by green renewable energy. Based on an MILP formulation, this scheduler executes the workload adaptively so as to maximize green energy consumption and minimize grid energy consumption, while respecting service-level agreements (SLAs). Metwally et al. [15] have used a two-phase MILP formulation to improve the resource utilization of cloud datacenters under the infrastructure-as-a-service (IaaS) paradigm. Mohamma Ali et al. [18] have presented a new datacenter design using disaggregated server (DS) that arranges resources in different physical pools, and developed an MILP model to optimize the VM allocation for DS-based datacenters.

2.2 Datacenter Thermal Models

Several authors have considered thermal-aware job scheduling in datacenters. While most works rely on thermal models that capture either the spatial correlation or the temporal correlation, very few consider both of them simultaneously.

A spatial thermal model characterizes the spatial correlation of the temperatures in different servers of a datacenter, leading to a "thermal map" of the datacenter. Moore et al. [19] have introduced the notion of heat recirculation to capture the thermal profile of a datacenter by taking physical layout and heat flow into account. Tang et al. [33] formally defined a heat-distribution matrix via an abstract heat flow model for the optimization of the cooling cost of a datacenter. This abstract spatial model has been successfully validated by several computational fluid dynamics (CFD) simulations [34], [25], [21], [1] that have shown high accuracy of the model within the natural range of temperature fluctuation. This model has subsequently been adopted in many thermal-aware scheduling research works (see Section 2.3). In contrast to a spatial model, a temporal thermal model accounts for the temperature of a single server over different time intervals. Ramos and Bianchini [24] have predicted the temperature of servers in a datacenter based on a simple temporal model governed by heat transfer laws. By exploring the duality between electrical circuits and heat transfer, Skadron et al. [28] have proposed the lumped-RC model to capture the transient temperature variation in processors. The model has been validated using a commercial, finite-element simulator of 3D fluid and heat flow for chips with measured errors below 6% and usually within 3% [29]. The authors have also developed HotSpot, a thermal modeling and simulation tool for microprocessor architectures [29]. Like the spatial model, the temporal model has also become widely adopted by thermal-aware scheduling researchers (see Section 2.3). Our prior work [32], [11] has considered a holistic thermal model that captures both spatial and temporal aspects of temperature distribution in datacenters. In this work, we build on this spatio-temporal model to formulate a variety of thermal-aware scheduling problems.

2.3 Thermal-Aware Scheduling Heuristics

Many existing works have proposed thermal-aware scheduling heuristics with the objective of minimizing cooling cost, energy consumption, and/or application performance. Here, we focus on works that consider the thermal models referenced in Section 2.2. For a more complete literature review, readers are referred to the survey [7] on thermal-aware scheduling for datacenters.

Based on a spatial thermal model, a simple job placement heuristic, often used as a baseline by researchers is "coolest first", which places a job on the server with the lowest (inlet) temperature. Moore et al. [19] have proposed several heuristics, and in particular MinHR, which assigns each job to the server that contributes minimally to the heat recirculation in the datacenter. Pakbaznia and Pedram [22] have proposed to reduce the total energy consumption of a datacenter by performing server consolidation in a way that accounts for heat recirculation. Mukherjee et al. [17] have considered a similar problem while taking the temporal job placements into account (but without a temporal thermal model). Sun et al. [31] have studied performance-energy tradeoff in heterogeneous datacenters while

considering heat recirculation effects, and proposed server placement strategies that minimize cooling cost. By assuming specific heat recirculation patterns, Mukherjee et al. [20] have designed approximation algorithms for a couple of related thermal-aware scheduling problems. Liu et al. [13] have designed scheduling algorithms for big-data jobs using DVFS under a similar spatial thermal model.

In terms of works that leverage a temporal thermal model, Wang et al. [36] have applied the lumped-RC model to predict the temperatures of the servers in a datacenter in order to make job placement decisions. Rajan and Yu [23] have relied on the same model to maintain the temperature of the system below a threshold by using DVFS while maximizing application throughput. Zhang and Chatha [39] have designed polynomial-time approximation schemes for the discrete version of the problem (assuming a discrete set of available DVFS levels) with the objective of minimizing application makespan. Yang et al. [37] have proposed intelligent ordering of the jobs based on their thermal characteristics for reducing the number of thermal constraint violations. Mhedheb and Streit [16] have considered thermal-aware VM management in Cloud datacenters to minimize energy using migration techniques. Van Damme et al. [35] have characterized the optimal workload distribution in a datacenter using KKT conditions with a thermal constraint.

In our prior work, we have considered thermal-aware scheduling while relying a spatio-temporal model. Specifically, Sun et al. [32] have proposed thermal-aware strategies to minimize the makespan of a set of HPC jobs by using DVFS to ensure that the temperature remains below a threshold in a homogeneous datacenter. Herzog and Pierson [11] have considered a similar problem but used a multi-agent based system approach for performing job assignment. In contrast to [32] and [11], which proposed heuristic solutions in an HPC setting to optimize makespan, we construct MILP formulations for both HPC and Cloud settings with both makespan and energy as objectives. As discussed previously, the solutions of our MILP formulation make it possible to assess the quality of solutions produced by these heuristics for small problem instances, and to guide the design of better heuristics for solving these thermal-aware scheduling problems for large instances.

3 Problem Statement

We consider a general thermal-aware scheduling problem: given a datacenter platform and a workload to execute on that platform, optimize a performance or energy objective subject to thermal constraints. We use the following assumptions:

- *Platform*: the platform is a set of (heterogeneous) servers, or *nodes*, in a datacenter with air cooling. We consider a typical datacenter layout with several rows of node racks organized in alternating cold and hot aisles. Cold air is provided by the CRAC (Computer Room Air Conditioning) unit and we assume that the air temperature from the CRAC is constant. Each node is defined by a maximum compute speed (i.e., a number of operations per second that can be performed at full utilization), as well as by parameters that determine its power and thermal behaviors (see details in Section 4).
- *Workload*: the workload is a set of single-node, independent jobs, each of which is characterized by an amount of computation to perform (i.e., a number of operations) and a maximum utilization of a node's compute capacity. We do not model other resource demands (e.g., memory, network bandwidth).
- *Objectives*: a scheduling problem can be framed to address different objectives, and we consider two main objectives:
  - *Makespan minimization*: This objective corresponds to a Service Level Agreement between datacenter providers and users. The makespan of a workload is defined as the time elapsed between the time when the workload enters the system and the time when its last job completes. Makespan minimization thus leads to users receiving job results quickly, but it also reduces the amount of time nodes are powered on and computing, which can also lead to energy consumption reduction.
  - *Energy consumption minimization*: This objective corresponds to the environmental impact of datacenters in terms of carbon emission as well as to operating costs. A key motivation to reduce energy consumption is to reduce the heat generated by the nodes, which can thus reduce the datacenter cooling cost.
- *Constraints*: The minimization of the objective functions above are subject to constraints on node temperatures. These temperatures should

always be below some datacenter specified threshold. Additional constraints (e.g., on the frequency of job migrations, on space sharing policies) can also be specified depending on the scenario at hand (i.e., HPC or Cloud). Finally, we always minimize energy consumption under a makespan constraint (otherwise an optimal solution could consist in not computing anything, since an infinite makespan has zero energy consumption).

4 Thermal Models

We consider a datacenter with N nodes, $\{n_1, n_2, \cdots, n_N\}$. Node i is characterized by the following parameters: thermal resistance R_i, thermal capacitance C_i, compute speed s_i, and idle power consumption P_i^{idle}. Time is discretized between time $t = 0$ and time $t = L$ with a time step Δt. When the objective is to minimize the makespan, L is the first timestep at which a feasible solution is reached; when the objective is to minimize the energy consumption subject to a makespan constraint M, then $L = M$.

$T_i^{in}(t)$, resp. $T_i^{out}(t)$, is the inlet, resp. outlet, temperature of node i at time t. We consider $T_i^{out}(t)$ to be the temperature of node i itself. The thermal constraint, to avoid overheating, is then that $T_i^{out}(t)$ should be below a threshold temperature T^{thresh}. T^{thresh} is typically determined based on the junction temperature of the chips [8]. $P_i(t)$ is the total power consumption of node i at time t. We assume that $T_i^{in}(t)$, $T_i^{out}(t)$, and $P_i(t)$ are constant over the interval $[t, t+\Delta t)$, thus the smaller the Δt the more realistic (i.e., approximately continuous) the model. Table 1 summarizes the notations used throughout this paper.

With the above definitions, the temperature evolution of node i is:

$$T_i^{out}(t+\Delta t) = P_i(t)R_i + T_i^{in}(t) + (T_i^{out}(t) - P_i(t)R_i - T_i^{in}(t)) \times e^{-\frac{\Delta t}{R_i C_i}} . \quad (1)$$

Given a workload allocation for node i at time t, i.e., a power consumption, the above model makes it possible to compute the temperature variation over the next time interval of duration Δt. This "RC model" is used in many previous works [28], [39], [36].

One of the challenges of thermal modeling is capturing the effects of air recirculation. Air recirculation causes the inlet temperature of a node to deviate from that provided by the CRAC unit, i.e., its temperature is higher due to the hot air recirculated from the outlets of other nodes in the datacenter. The heat produced by all nodes in the datacenter, including adjacent nodes, impacts the temperature of each node. The work by Tang et al. [33], [34] has made advances toward modeling air recirculation, but only in the context of steady-state execution without considering temporal evolution (i.e., $\Delta t = \infty$). Let T^{sup} be the temperature supplied to the datacenter by the CRAC unit. The work in Tang et al. [33], [34] gives:

$$\overrightarrow{T^{in}}(t) = \overrightarrow{T^{sup}}(t) + D \times \overrightarrow{P}(t) , \quad (2)$$

where $\overrightarrow{T^{in}}(t)$ is an N-dimensional vector whose components are the $T_i^{in}(t)$'s, $\overrightarrow{T^{sup}}(t)$ is an N-dimensional vector whose components are all equal to T^{sup}, $\overrightarrow{P}(t)$ is an N-dimensional vector whose components are the $P_i(t)$'s, and D is an N-by-N air recirculation matrix, which is constant and computed for a given datacenter configuration.

Combining the RC model and the air recirculation model, i.e., temporal and spatial temperature evolution, Sun et al. [32] compute all T^{out} values in matrix/vector form as follows:

$$\overrightarrow{T^{out}}(t+\Delta t) = \overrightarrow{P}(t) \times R + \overrightarrow{T^{in}}(t) + (\overrightarrow{T^{out}}(t) - \overrightarrow{P}(t) \times R - \overrightarrow{T^{in}}(t)) \times F , \quad (3)$$

where $R = \text{diag}(R_1, \ldots, R_N)$ and $F = \text{diag}(e^{-\frac{\Delta t}{R_1 C_1}}, \ldots, e^{-\frac{\Delta t}{R_N C_N}})$. Note that R and F are constant.

Let J be the number of independent jobs to be executed on the platform. Job j is defined by an amount of work (number of CPU cycles) w_j, and a maximum fraction α_j of a compute node's compute capacity that it can use. For instance, a job with $\alpha_j = 0.5$ will only utilize half of a node's compute capacity.

As discussed previously, we consider two objectives: the makespan, denoted by M, which is the time when all the jobs in the workload are completed, and the energy consumption, denoted by E, which is the total energy consumed during the execution of the workload. We consider the following two execution scenarios:

- **Scenario 1 – High Performance Computing (HPC):** In this scenario, only one job can be executed on a node at any time and no job migration is allowed, i.e., once a job has begun executing on a node it must finish execution on that node. Furthermore, no temporal interleaving of job execution on a node is allowed: a job scheduled on a node must wait for other jobs previously scheduled on that node to complete

Table 1: List of notations.

Notation	Meaning
N	Number of compute nodes
R_i	Thermal resistance of node n (W/°C)
C_i	Thermal capacitance of node n (J/°C)
s_i	Compute speed of node i (ops/s)
P_i^{idle}	Power consumption of node i when idle (W)
$P_i(t)$	Power consumption of node n at time t (W)
$T_i^{in}(t)$	Inlet temperature of node n at time t (°C)
$T_i^{out}(t)$	Outlet temperature of node n at time t (°C)
T^{sup}	Temperature supplied by the CRAC unit (°C)
T^{thresh}	Threshold temperature (°C)
$\overrightarrow{T^{in}}(t)$	N-dimensional vector of the T_i^{in}'s
$\overrightarrow{T^{out}}(t)$	N-dimensional vector of the T_i^{out}'s
$\overrightarrow{T^{sup}}$	N-dimensional vector with all components equal to T^{sup}
$\overrightarrow{T^{thresh}}$	N-dimensional vector with all components equal to T^{thresh}
R	$\text{diag}(R_1, \ldots, R_N)$
F	$\text{diag}(e^{-\frac{\Delta t}{R_1 C_1}}, \ldots, e^{-\frac{\Delta t}{R_N C_N}})$
D	air recirculation matrix
J	number of jobs
w_j	amount of work of job j (ops)
p_j	dynamic power consumption constant over time of job j
α_j	maximum node utilization of job j (%)
$\alpha_{i,j,t}$	the fraction of node i used by job j at time t
$e_{i,j}$	true if job j runs on node i
$e_{i,j,t}$	true if job j runs on node i at time t
$started_{i,j,t}$	true if job j has already started on node i at time t
$ended_{i,j,t}$	true if job j has finished on node i at time t
$Pon_{i,t}$	true if node i is switched on at time t
$\overrightarrow{CV}$	Vector of node fraction allocations for Cloud scenario
$\beta_{i,j,t,v}$	v-th value of $\overrightarrow{CV}$ chosen for job j on node i at time t
c_j	completion time of job j
M	Makespan
E	Energy

before beginning executing. However, a job can be temporarily suspended and resumed later, so as to allow the node's temperature to decrease.

– **Scenario 2 – Cloud:** In this scenario, several jobs can share the same node and job execution interleaving is also allowed. Task migration, however, is still not allowed. The rationale is that in real-world datacenters jobs are rarely migrated. Migrations only happen during scheduled resource consolidation phases, which we do not consider in this work.

Figure 1 shows the roadmap of the overall formulation and optimization process for both scenarios.

5 MILP Formulations

In this section, we present MILP formulations for the thermal-aware scheduling problems described in the previous section, detailing how they can be applied to both HPC and Cloud settings.

5.1 Task Placement Constraints

To express generic job placement constraints we define the following variables:

– $\alpha_{i,j,t}$: the fraction of node i used by job j at time t. We consider two cases. If $\alpha_{i,j,t}$ is declared as a binary variable, then only one job can be allocated to one node at a given time, which is in line with Scenario 1 (HPC). Once a job has begun executing on a node, the only option for decreasing the temperature of the node is then to temporarily suspend the job's execution. If, instead, $\alpha_{i,j,t}$ is declared as a rational variable, then several jobs can run on one node simultaneously, and the fraction of the node's compute capacity that is used by a job can be reduced in order to decrease temperature.

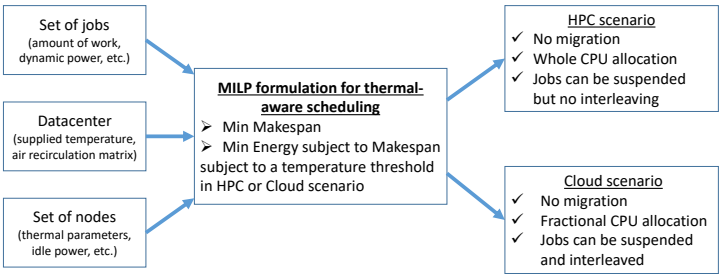


Fig. 1: Roadmap of our overall approach for both scenarios.

– $e_{i,j}$: a binary variable that equals 1 if job j runs on node i, and 0 otherwise.

Given $i \in \{1,\ldots,N\}$ (nodes), $j \in \{1,\ldots,J\}$ (jobs), and $t \in \{0,\ldots,L\}$ (timesteps), we have the following constraints:

$$\forall i,j \quad 0 \leq e_{i,j} \leq 1 \tag{4}$$
$$\forall i,j,t \quad 0 \leq \alpha_{i,j,t} \leq \alpha_j \tag{5}$$
$$\forall i,j,t \quad \alpha_{i,j,t} \leq e_{i,j} \tag{6}$$
$$\forall i,t \quad \sum_j \alpha_{i,j,t} \leq 1 \tag{7}$$
$$\forall j \quad \sum_t \sum_i \alpha_{i,j,t} s_i = w_j \tag{8}$$
$$\forall j \quad \sum_i e_{i,j} = 1 \tag{9}$$
$$\forall t \quad \overrightarrow{T^{out}}(t) \leq \overrightarrow{T^{thresh}} \tag{10}$$

– Constraint (4): the $e_{i,j}$ variables are binary;
– Constraint (5): a job cannot use more than its maximum resource usage;
– Constraint (6): if a job is not running on a node, then it is not using any of its resources;
– Constraint (7): the total compute capacity of a node is not exceeded;
– Constraint (8): the work of each job is fully executed;
– Constraint (9): a job runs only on one node (true for all timesteps, since there is no job migration);
– Constraint (10): the temperature threshold is respected.

In what follows we provide additional variables and constraints specific to HPC and Cloud settings.

5.1.1 HPC Setting

In an HPC environment, a node is generally dedicated to one application of one user, who has been allocated the node for their need. Even if the operating system (OS) uses some processing power, in this work we consider that the entire processing power of a node is allocated to the user's job and we ignore the impact of the OS.

Hence, in an HPC setting only one job runs on a node at a time, and we add the following binary variables accordingly:

– $e_{i,j,t}$: equals 1 if job j is running on node i at time t and 0 otherwise.
– $started_{i,j,t}$: equals 1 if job j has already started on node i at time t and 0 otherwise.
– $ended_{i,j,t}$: equals 1 if job j has completed on node i at time t and 0 otherwise.

We can then add the following constraints:

$$\forall i,t \quad \sum_j e_{i,j,t} \leq 1 \tag{11}$$
$$\forall i,j,t \quad started_{i,j,t} \geq \alpha_{i,j,t} \tag{12}$$
$$\forall i,j,t \quad started_{i,j,t+1} \geq started_{i,j,t} \tag{13}$$
$$\forall i,j,t \quad ended_{i,j,t+1} \geq ended_{i,j,t} \tag{14}$$
$$\forall i,j,t \quad e_{i,j,t} = started_{i,j,t} - ended_{i,j,t} \tag{15}$$
$$\forall i,j,t \quad started_{i,j,t} + ended_{i,j,t} + \alpha_{i,j,t} \leq 2 \tag{16}$$
$$\forall i,j,t \quad started_{i,j,t} \leq e_{i,j} \tag{17}$$
$$\forall i,j,t \quad ended_{i,j,t} \leq e_{i,j} \tag{18}$$
$$\forall i,j,t \quad e_{i,j,t} \leq e_{i,j} \tag{19}$$

– Constraint (11): only one job is running on a node at a given time;
– Constraint (12): a job can be started only if it is given sufficient resources;
– Constraint (13): when a job has already started at time t, it also has already started at time $t+1$;
– Constraint (14): when a job is finished at time t, it is also finished at time $t+1$;
– Constraint (15): a job runs only when it is started and not finished;
– Constraint (16): a job is not given resources once it is finished;

- Constraint (17): a job can only be started on a node where it is allocated;
- Constraint (18): a job can only be finished on a node where it is allocated;
- Constraint (19): a job can only run on a node where it is allocated.

Constraints (12) to (18) aim to compute the time interval during which a job is executing, possibly with some idle periods to decrease a node's temperature. In an HPC setting, constraint (11) prevents interleaving of job executions: another job cannot be executed during an idle period due to a running job being temporarily suspended.

5.1.2 Cloud Setting

In a Cloud computing environment, several virtual machines share the processor. Virtual cores are dedicated to virtual machines, and these virtual cores are mapped on physical processor cores, depending on the virtualization layer. In any case, several virtual machines share the node at the same time, and some virtual machines can be suspended temporarily to allow other virtual machines to access the physical resources. Similarly to the HPC setting, we ignore the usage of the processor by the OS and the virtualization layer.

In a Cloud setting, we allow interleaving of job executions on a node and allow several jobs per node at any time. Therefore Constraint (11) defined in the previous section is not necessary. All other constraints are maintained.

We introduce two variables to represent that jobs can use an arbitrary fraction of a node's compute capacity:

- $\overrightarrow{CV}$: is a vector of node allocations, which contains the fraction of node compute capacities that can be allocated to jobs;
- $\beta_{i,j,t,v}$: is a binary variable which is equal to 1 if the v-th value of $\overrightarrow{CV}$ is chosen for job j on node i at time t and 0 otherwise.

We then need a single constraints to guarantee that, at any time for one job on one node, only one $\overrightarrow{CV}$ value can be chosen:

$$\forall i,j,t \ \sum_v \beta_{i,j,t,v} = 1 \ . \tag{20}$$

Then, $\alpha_{i,j,t}$ is computed as follows:

$$\alpha_{i,j,t} = \overrightarrow{CV}[v] \times \beta_{i,j,t,v} \ .$$

For example, if fractions of a node's computational power are allocated in 10% increments, we would have:

$$\overrightarrow{CV} = (0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1) \ ,$$

and we could then express the fact that job $j = 0$ on node $i = 0$ at time $t = 0$ uses 90% of the node's computational power as:

$$\beta_{0,0,0,9} = 1 \ .$$

5.2 Objective Functions

5.2.1 Minimize Makespan

The makespan M is defined as the time when all job in the workload are completed. The completion time of each job j is computed as follows:

$$c_j = \sum_{i=1}^{N} (e_{i,j,0} + \sum_{t=1}^{L} \left(e_{i,j,t} + t \times (started_{i,j,t} - started_{i,j,t-1})\right)) \tag{21}$$

We can then write the following constraint, specifying that the makespan is greater than the completion time of each job:

$$\forall j \ M \geq c_j \ . \tag{22}$$

In other words, minimizing the makespan is equivalent to minimizing the maximum c_j.

5.2.2 Minimize Energy Consumption

The energy consumption comes from two sources: computing and cooling. The computing energy is given by the total computing power of all nodes integrated over time, which is usually composed of a static part (when nodes are powered on but idle) and a dynamic part (when nodes are on and computing). In [19], the cooling energy consumption is given as a function of the energy consumed by computing, which makes it possible to express the total energy consumption of the datacenter as:

$$E = \Delta t \times \sum_t \left(\sum_i P_i(t) \right) \times \left(1 + \frac{1}{CoP(T^{sup})} \right) \ , \tag{23}$$

where CoP is a quadratic function, i.e., $CoP(T^{sup}) = a(T^{sup})^2 + bT^{sup} + c$, with a, b and c depending on the performance of the cooling device. Note that when T^{sup} is a variable, the problem is no longer linear because E is not linear in T^{sup}. However, this difficulty can be resolved by approximating E as a step function of T^{sup}. If T^{sup} is a constant, as assumed in this work, the cooling energy turns out to

be a fixed overhead on top of the computing energy, so the total energy consumed by the system can be minimized by only considering the energy due to computing.

We consider that a node consumes no power when no job is running, i.e., we assume that when no job is running on the node then the node is either powered off or put into a suspended mode. The power consumption and the time to switch from the suspended mode to the running mode are neglected. If at least one job is running on node i, then a fixed P_i^{idle} power consumption is added to the power consumption due the job execution. In terms of our MILP formulation, we add the following binary variable

– $Pon_{i,t}$: equals 1 if node i is powered on at time t

and the two following constraints:

$$\forall i,t \quad Pon_{i,t} = \sum_j \alpha_{i,j,t} \text{ (for HPC setting)} \tag{24}$$

$$\forall i,t \; Pon_{i,t} \geq \sum_j \alpha_{i,j,t} \text{ (for Cloud setting)} . \tag{25}$$

– Constraints (24) and (25): at any time, if at least one job has been allocated resources on one node, the node is powered on. Note that Constraint (25) does not force $Pon_{i,t}$ to be equal to zero when the node is idle. This will be realized by the linear program solver when minimizing the energy consumption.

We consider that each job j has a power consumption p_j that is constant over time. Hence, the power consumption of node i at any time t is a function of the fractions of node i's compute capacity allocated to jobs running on node i, the power consumption of these jobs, and the idle power consumption of node i if it is powered on:

$$P_i(t) = \sum_j \alpha_{i,j,t} \times p_j + P_i^{idle} \times Pon_{i,t} . \tag{26}$$

A constraint on makespan can be added to formulate the "Optimize E subject to M" problem. It says that the makespan should be at most a factor of x larger than a target makespan value M, which could be a job's deadline or M_{opt} obtained by the "Optimize M" problem. A trade-off between performance (makespan) and energy consumption can be obtained:

$$\forall j \; c_j \leq x \times M . \tag{27}$$

5.3 Accounting for DVFS Capabilities

Some nodes may have DVFS capabilities, which makes it possible to dynamically manage their power consumption. The dynamic power consumption of a node is usually a convex function of its variable speed [6], [38]. In practice, nodes only provide a relatively small set of discrete speeds to choose from. The node speeds are variables represented by $s_{i,t}$ which corresponds to the DVFS level of node i at time t. The dynamic power consumption of a node executing a job j at speed $s_{i,t}$ can be approximated by $s_{i,t}^{\beta} p_j$, where $\beta > 1$ denotes the power parameter (usually 2 or 3 for CMOS-based processors), and the execution time of the job is $w_j / s_{i,t}$.

Accounting for DVFS capabilities in our MILP formulation can be done by modeling the power consumption of node i at time t as follows:

$$P_i(t) = \sum_j \alpha_{i,j,t} \times s_{i,t}^{\beta} \times p_j + P_i^{idle} \times Pon_{i,t} . \tag{28}$$

With this modification, however, the formulation for the energy consumption would no longer be a linear function. As a result, the optimization problem is no longer a linear program. One option is, again, to approximate the energy as a step function. In all experiments that follow, we do not consider this DVFS model and instead use the constant dynamic power model in Equation (26).

6 Experimental Case-Study

To evaluate the correctness and usefulness of our proposed MILP formulations, we solve formulations for small problem instances in the HPC and Cloud settings. We use the commercial Gurobi solver (version 6.5) with a Gurobi gap of 0.01%, which represents the difference between the computed solution and a computed bound on the optimal solution. In the rest of this section, we simply call this value the "gap". Note that the evaluation of the spatio-temporal model of temperature evolution is not part of this work, since they have been already evaluated in previous works [19], [34], [29]. We run the experiments on an Intel Xeon E5-2603 processor with a 1.60GHz CPU and 32 GB of RAM.

6.1 Experimental Setup

Workload – We consider a set of 12 jobs to be executed on 6 homogeneous nodes. Because the nodes are homogeneous, for each node i, $R_i = R$ and $C_i = C$. Thermal parameters are based on those in [29], [37]: we set $R = 0.7$ and the values of F are constant and defined to be $e^{-\frac{\Delta t}{RC}} = 0.5$ (which determines the value of C). For each job j, we generate values for the work (w_j), resp. the dynamic power

consumption (p_j), via sampling from a uniform probability distribution with range 1-10 timesteps, resp. 50-120 Watts according to typical node power consumption [40]. In this manner, we generate 40 different instances for which we solve the MILP formulations in Section 5.

We set $\alpha_j = 1$ for both HPC and Cloud contexts, meaning that jobs can fully utilize nodes. In the HPC context $\alpha_{i,j,t}$ can be 0 or 1. In the Cloud context $\alpha_{i,j,t}$ takes discrete values between 0 and 1 in 0.1 increments (as specified by the $\overrightarrow{CV}$ vector), which represents realistic partial node allocation schemes in typical clouds.

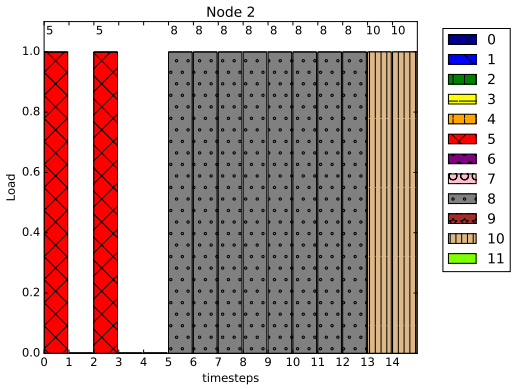
Datacenter – We adopt a classical air recirculation matrix configuration (denoted by D) that is representative of typical datacenters [4]. The maximum node temperature T^{thresh} is typically between 85°C and 100°C [8]. We opt for $T_{thresh} = 100$°C for the output temperature of all nodes. The static power of a node, P_i^{idle}, varies among architectures but is typically in the range of 10-50 Watts. We assume it contributes 15°C to the temperature of each processor [8]. Since power and temperature are linked by Equation (3) (thermal model), with the chosen thermal resistance R and capacitance C, a temperature increase of 15°C gives an idle power of 42 Watts. Hence, we set $P_i^{idle} = 42$ Watts.

6.2 Node Resource Allocation Examples

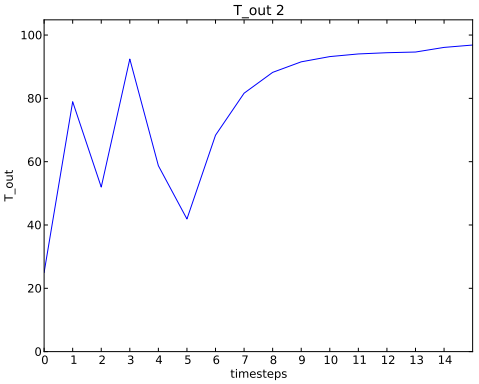
Before describing objective functions and results obtained when solving our 40 problem instances, we illustrate patterns of job executions and node temperature variations in typical solutions both in HPC and Cloud settings.

6.2.1 HPC Setting

The top part of Figure 2 depicts the load (vertical axis) of a typical node for 15 time steps (on the horizontal axis). At each time step, the node executes a single job, as dictated by the HPC setting. Among the 12 jobs in the workload, 3 are executed on this node (job IDs are shown above each bar). Job 5 is the first to execute at time step 0. Due to the increase in temperature, the node is suspended during the next time step, to avoid exceeding the temperature threshold. Job 5 then resumes at time step 2, followed by two idle time steps, again to allow for the node temperature to decrease. Then jobs 8 and 10 execute in sequence without the node being suspended due to its temperature being below threshold (e.g., because nearby nodes have lower temperature than at earlier time steps). According to the constraints in the HPC setting, jobs execute one after the other and the compute capacity of a node is either allocated entirely to a job or not allocated at all. The bottom part of Figure 2 shows the node temperature (vertical axis) at each time step (horizontal axis). As expected we see valleys corresponding to time steps in which the node is suspended, and we see temperature increases as the node computes (e.g., from time step 5 onward).



(a) Job execution



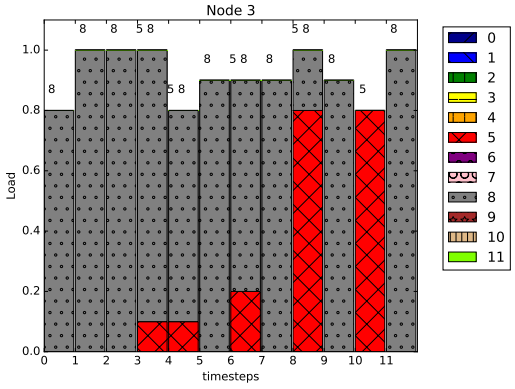
(b) Node temperature

Fig. 2: Example workload execution and temperature variation on a node in the HPC setting.

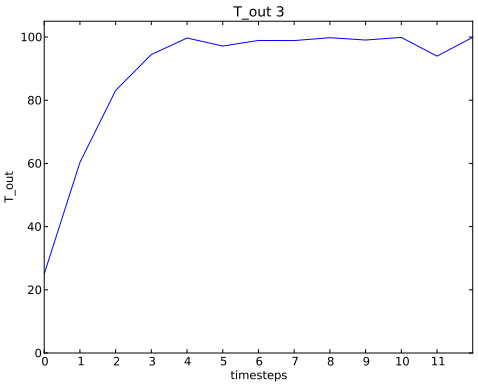
6.2.2 Cloud Setting

Figure 3 shows a similar example as in the previous section, but for the Cloud setting. Jobs are allowed to share the node during the same time step. For example, at time step 3, jobs 5 and 8 share the compute capacity of the node (job 5 uses 10%, job 8 uses 90%). Also, the node's compute capacity is not necessarily used fully at each time step.

For instance, at time step 0, job 8 uses only 80% of the capacity. This makes it possible to manage the node's temperature without introducing idle periods but instead by merely reducing the node usage. As a result, the evolution of the node's temperature (bottom of Figure 3) is smoother than in the HPC setting (bottom of Figure 2).



(a) Job execution



(b) Node temperature

Fig. 3: Example workload execution and temperature variation on a node in the Cloud setting.

6.3 Optimization Objectives

We consider the following two optimization problems, for which we have given MILP formulations in Section 5:

- Minimize the makespan M;
- Minimize the energy E, subject to a makespan constraint.

Both optimization problems are subject to thermal constraints. The reason why we impose an additional makespan constraint for optimizing E is as follows. Recall that we consider that when a node is idle it can be powered off/suspended, thus making its power consumption zero. Optimizing solely the energy without considering the makespan can thus lead to arbitrarily many idle time steps (since this does not increase energy consumption and in fact helps with thermal constraints). But in practice, this can lead to unacceptably large makespans. To the extreme, if the only objective is the energy, then one should keep all nodes powered off.

To illustrate this point, for one of our instances, we ran our solver to optimize the makespan and to optimize the energy without any makespan constraint, in an HPC setting. Makespan and energy Results are shown in Table 2. These results show that, when optimizing the makespan (the "Optimize M" column), our solver produces a solution with a makespan of 8 and an energy consumption of 4107.28. When optimizing the energy (the "Optimize E" column), our solver leads to the same energy, but a makespan of 20 (more than twice longer than the optimal makespan). When running the solver to minimize the energy but adding the constraint that the makespan should be below or equal to 8, we obtain the same energy consumption of 4107.28 (the "Optimize E subject to M" column). In this particular case, all optimization problems have solutions with the same energy.

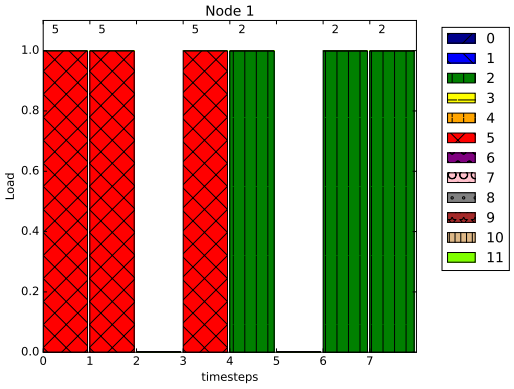
To better understand the above we pick a node and plot its load and power consumption throughout the execution in the HPC setting for the "Optimize M" (Figure 4), "Optimize E" (Figure 5), and "Optimize E subject to M" (Figure 6) approaches. In all three figures the node is either fully utilized by a single job or idle, which is consistent with the assumptions of the HPC setting (and the fact that we set $\alpha_{i,j,t} = 1$ for each job j). In Figure 4 we see that out of the 8 timesteps that make up the makespan, the compute node is kept idle during 2 timesteps so as to reduce temperature. During each non-idle timestep, the node executes job 5 then job 2. Figure 5 shows a very different picture, in which there are many more idle timesteps. And yet, it is possible to have fewer idle timesteps while respecting thermal constraints (as in Figure 4). The reason for this behavior is that since there is no incentive to finish earlier in the "Optimize E" problem, the solver finds an "easy" solution with a much higher makespan. The energy spent during the unnecessary time steps is zero based on our assumption when no jobs runs on a node. The results in Figure 6, for "Optimize E subject to M", show that it is possible, in this case, to optimize the energy and achieve the same energy consumption (see Table 2)

Table 2: Example makespan and energy values when solving optimization problems in an HPC setting.

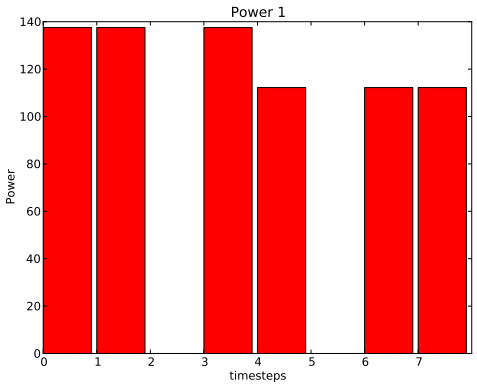
	Optimize M	Optimize E	Optimize E subject to M
Makespan (M)	8	20	8
Energy (E)	4107.28	4107.28	4107.28

with much fewer time-steps. The produced solution is different from that in Figure 4 for "Optimize M" (different jobs are executed). But it results in similar makespan and energy consumption.

The power depends on the jobs power consumption. Each job as a dynamic power consumption (p_j). In HPC, the power consumed during a time-step depends on which job is executed. When a job is executing it has the whole CPU so the power is equal to job's consumption. In the cloud, if a job only has a ratio $\alpha_{i,j,t}$ of the CPU of a node, the power on that node due to that job is equal to the $\alpha_{i,j,t}$ ratio of the job consumption.

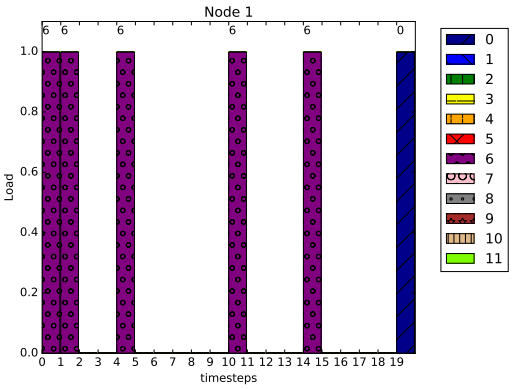


(a) Job execution

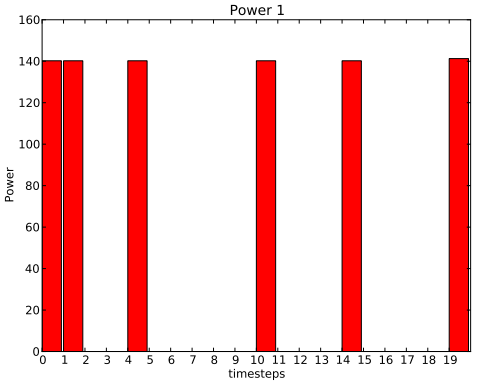


(b) Node power

Fig. 4: Workload execution and power consumption on a node when optimizing makespan (HPC setting).



(a) Job execution

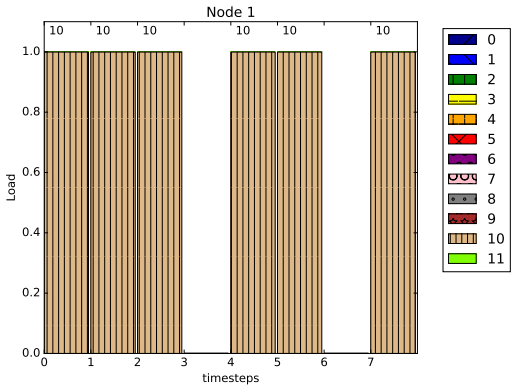


(b) Node power

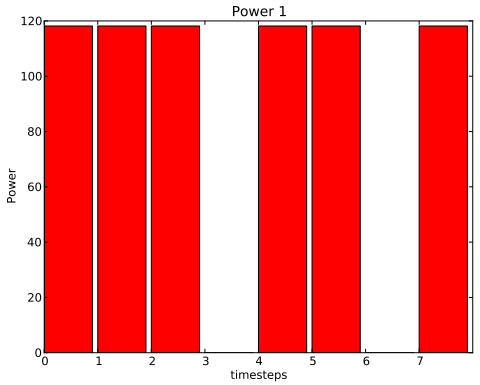
Fig. 5: Workload execution and power consumption on a node when optimizing energy (HPC setting).

Figure 7 shows results for "Optimize E" but in the Cloud setting. We see that the node is idle for only one timestep, but the node is shared by different jobs during two time-steps. This flexibility, in contrast to the HPC setting, makes it possible to perform more computation per time unit while managing the temperature so as to closely respect thermal constraints. In this case, it turns out that a better makespan (of 7) can be achieve in the Cloud setting than in HPC setting.

Equation (27) expresses a general constraint for the "Optimize E subject to M" problem. The makespan should be at most a factor x larger than the makespan obtained by solving the "Optimize M" problem. Picking a particular value of x would then achieve a particular trade-off between application performance

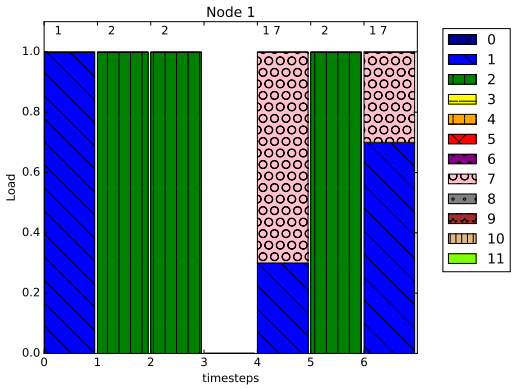


(a) Job execution

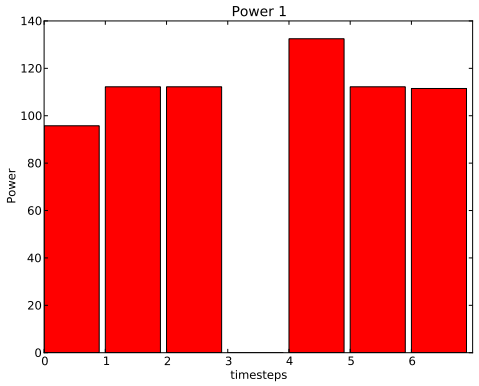


(b) Node power

Fig. 6: Workload execution and power consumption on a node when optimizing energy subject to makespan (HPC setting).



(a) Job execution



(b) Node power

Fig. 7: Workload execution and power consumption on a node when optimizing energy subject to makespan (Cloud setting).

and energy consumption. For instance, picking $x = 2$ would mean that one is willing to reduce the energy consumption at the cost of having a makespan twice as large as the optimal makespan. In all our experiments in this work, we use $x = 1$, which corresponds to not tolerating any increase in makespan for the sake of saving extra energy. The rationale for this strategy is that makespan and energy are tied anyway (a shorter makespan can reduce energy consumption because nodes are used for a shorter period of time). As in the example above, our results show that energy can be minimized with this stringent makespan constraint (i.e., inserting additional idle steps increases the makespan but does not reduce energy consumption in our results).

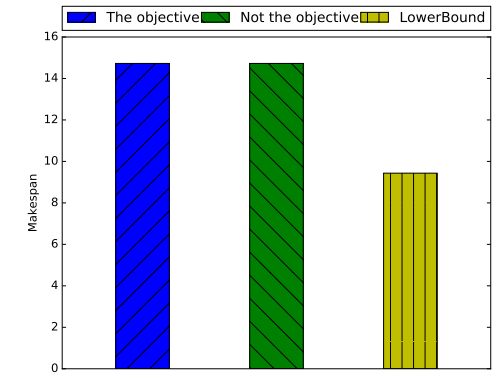
For both optimization problems, we consider theoretical (i.e., ideal) bounds on the optimal solution. A lower bound on the makespan is obtained by assuming that the workload is perfectly balanced across the nodes and that all jobs are executed at full speed ($\alpha_{i,j,t} = \alpha_j$ for each timestep t during which a job is executed). This lower bound cannot be achieved because load balancing is not perfect and because of thermal constraints. An upper bound on the energy is obtained by assuming that all nodes are powered on during the execution of the workload.

6.4 Case-Study Results

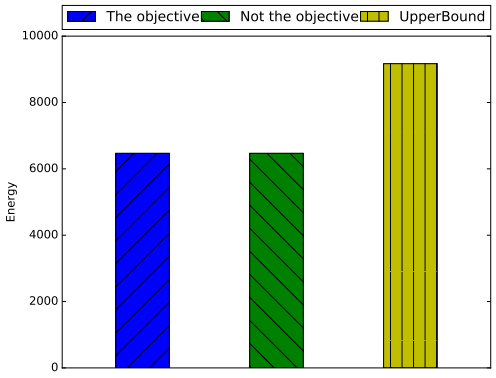
6.4.1 HPC Setting

Figure 8 shows average results over the 40 instances as obtained by the solver in the HPC setting, with makespan results in Figure 8(a) and energy results in Figure 8(b). In this figure, "The objective" refers to the case where the shown metric coincides with the optimization objective and "Not the objective" is when it does not. For instance, in Figure 8(a), the "Not the objective" bar is the average makespan

values obtained when optimizing the energy. First, we observe that the makespan is the same in both optimization problems. This is because the energy is optimized under the makespan constraint discussed earlier. Moreover, the energy is also the same in both optimization problems. It turns out that, in HPC settings, makespan and energy are equivalent objectives. This is because we have homogeneous nodes and a node's compute capacity is allocated either fully to a job or not at all, in which case the node is powered off. The allocation of jobs to nodes could vary when optimizing one objective or the other, but the energy is always directly proportional to the makespan (see Equation (23)).



(a) Makespan



(b) Energy

Fig. 8: Makespan and energy for different optimization problems in the HPC setting.

On average, the makespan found by the MILP solver is greater than the lower bound by 35.9%, which is expected since the lower bound cannot be achieved due to the scheduling and temperature constraints. Conversely, the energy found by the solver is lower than the upper bound by 41.7% on average.

6.4.2 Cloud Setting

In the Cloud setting, solving the linear program takes longer than in the HPC setting. This is because the search space is larger: Fractions of node compute capacity can be allocated to jobs (from 0 to 1 in steps of 0.1). Furthermore, unlike in the HPC setting, the energy takes continuous values, which renders the energy minimization more time consuming. When minimizing the energy, for most instances, a solution with a gap of 0.01% is not produced within 10 hours. Figure 9 plots the gap for each instance, as well as the average value. For 11 out of the 40 instances, the gap is below 1%. The maximum gap among all instances is just under 5%, and the mean gap is 1.87% with a fairly low standard deviation below 0.32%.

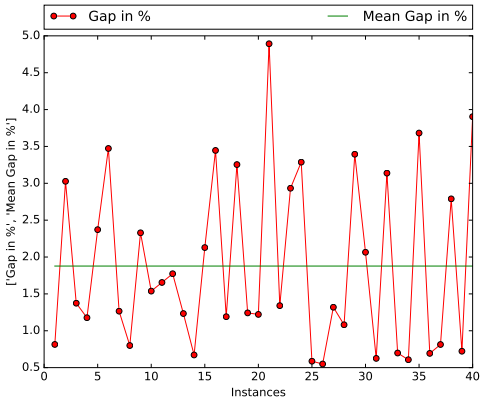
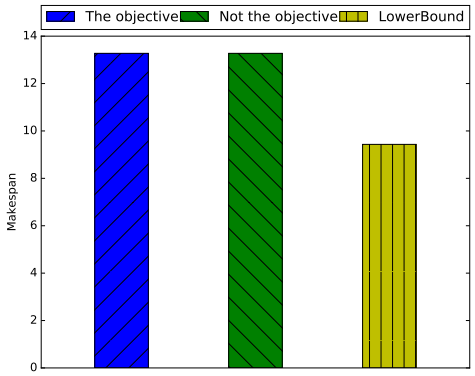


Fig. 9: Gap of each instance and mean gap for energy optimization results in the Cloud setting.

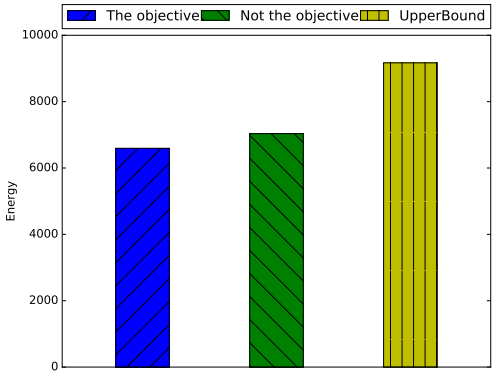
Figure 10(a) shows makespan results. Like in Figure 8(a), the makespan value is the same when the optimization objective is the makespan or the energy (due to the use of a makespan constraint when optimizing energy). Here the makespan is greater than the lower bound by 28.9% on average.

Figure 10(b) shows energy results. The achieved average energy is lower when the optimization objective is the energy. In other words, when optimizing for the makespan one ends up consuming more energy than the minimum achievable energy consumption. This is because the energy is not directly proportional to the makespan, since in the Cloud setting a node can be partially allocated to a job (or several jobs), thus leaving some compute capacity

unused. On average, the optimized energy value is lower than the upper bound by 39% while the energy obtained by optimizing the makespan is lower than the upper bound by only 30%.



(a) Makespan



(b) Energy

Fig. 10: Makespan and energy for different optimization problems in the Cloud setting.

Figure 11 shows, for each instance, the difference between the energy when it is the optimization objective and the energy when the optimization objective is the makespan. It also plots the gap for each instance since the energy optimization in the Cloud setting gives results with non-zero gap values. The maximum difference for the energy metric in both optimizations among all instances is above 12%, and the mean energy difference is 6.7%. The mean gap is around 2%, meaning that it does not explain the above differences. The results show that for the optimal makespan it is possible to achieve better energy consumption, which was not possible in the HPC setting. In the Cloud setting there is more flexibility to allocate resources: the execution can benefit from sharing the same node among multiple jobs so that energy consumed due to static power can be saved by optimizing the resource sharing. Thus, optimizing energy under the makespan constraint produces the best solutions in this context.

6.4.3 HPC vs. Cloud Settings

In both HPC and Cloud settings, the aim is to optimize either the makespan or the energy, while respecting a temperature threshold. The Cloud setting is more general since an HPC solution is essentially a more constrained Cloud solution (zero or full node allocation, no job execution interleaving). In this section, we compare the makespan and the energy obtained when optimizing either objective in both HPC and Cloud settings.

Figure 12 shows results for each metric (makespan or energy) when that metric is the optimization objective, in both HPC and Cloud settings. Figure 12(a) shows the makespan results. We can see that the makespan is lower (by 11%) in the Cloud setting. This is because being able to allocate less than 100% of a node's compute capacity makes it possible to complete all tasks earlier while respecting temperature constraints. Indeed, utilizing fractions of a node allows to decrease the temperature without ever powering down the node, whereas in an HPC setting idle periods are needed to decrease the temperature. Moreover, in the Cloud setting, job interleaving is allowed, which also helps to handle the nodes' temperatures without introducing idle periods (i.e., jobs that consume less power can be selected for execution). Figure 12(b) is similar to Figure 12(a), but shows average energy results. In the Cloud setting, the energy is slightly higher (by 1.9%). This is due to the gap in the energy optimization solution, which is 0% in an HPC setting but has an average value of 2% in the Cloud setting.

Figure 13 shows results for each metric (makespan or energy) when that metric is not the optimization objective, in both HPC and Cloud settings. Figure 13(a) shows makespan results. The results are identical to those in Figure 12(a). Again, this is because the energy is always optimized subject to a makespan constraint. Figure 13(b) shows that when the objective is the makespan, the energy achieved in the Cloud setting is 8.7% higher than the one achieved in the HPC setting. Note that the makespan is lower in the Cloud setting than in the HPC setting when the makespan is the optimization objective (Figure 12(a)). A better makespan is achieved in the Cloud setting by allocating fractions of the nodes' compute capacities, which leads to temper-

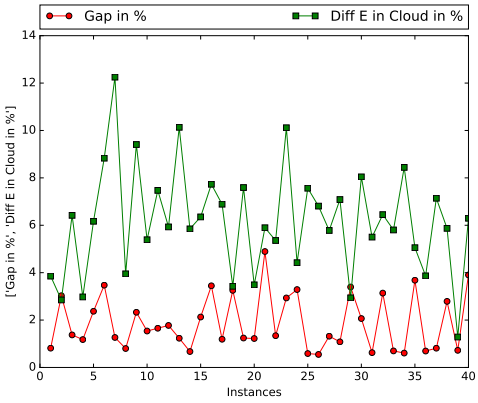
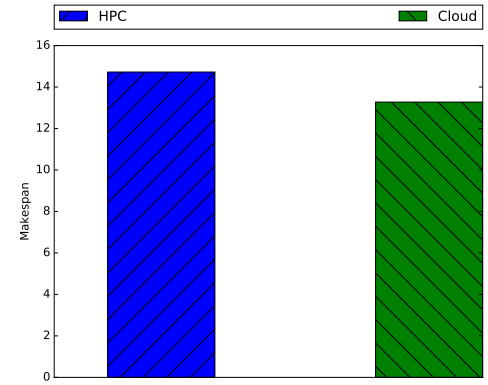
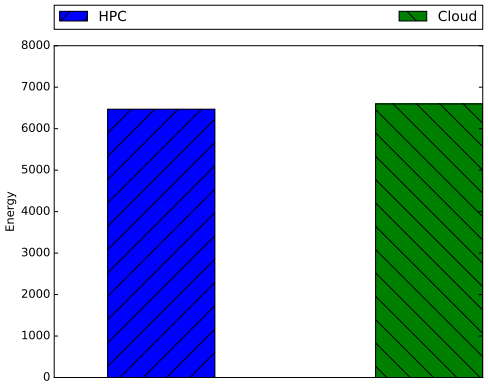


Fig. 11: Gurobi gap and the difference of energy consumption between optimizing energy and optimizing makespan in the Cloud setting.



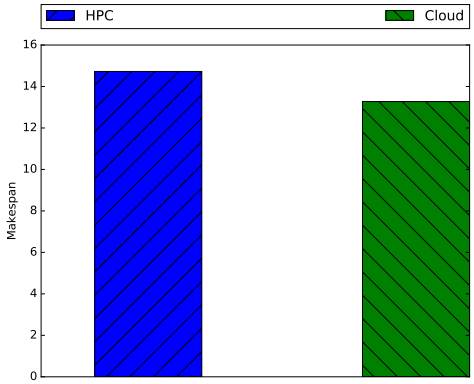
(a) Makespan optimization



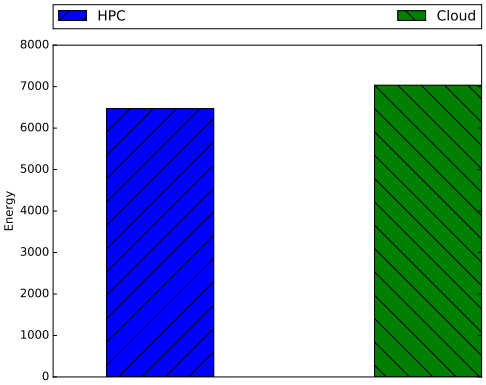
(b) Energy optimization

Fig. 12: Makespan and energy in both HPC and Cloud settings with matching objectives.

ature decreases. This is in contrast to the HPC setting, where idle periods are introduced, which saves energy (e.g., a node's static power consumption) but increases the makespan significantly. Overall, the results confirm the intuition that, in the Cloud setting, the best approach is to optimize energy subject to a makespan constraint.



(a) Energy optimization



(b) Makespan optimization

Fig. 13: Makespan and energy in both HPC and Cloud settings with opposite objectives.

6.4.4 MILP vs. Heuristics

As mentioned in Section 1, one use of an MILP formulation for a resource allocation problem is to assess, on small instances, the effectiveness of polynomial-time heuristics in an absolute sense. In this section, we demonstrate this use by evaluating two heuristics that have been proposed in the HPC setting (to the best of our knowledge no usable thermal-aware heuristic has been proposed in the Cloud setting). Specifically, we consider the Coolest heuristic [19], [33] and the Spatio-Temporal heuristic [32]:

- Coolest: a simple thermal-aware scheduling heuristic that places a job on the node with the lowest temperature at the time of assignment. This heuristic is not designed to be aware of the temperature threshold. To ensure that the threshold is not exceeded, we augment the heuristic so that it suspends a node when further execution would make the node's temperature exceed the threshold, and resumes it as soon as it is safe to do so.
- Spatio-Temporal: a thermal-aware scheduling heuristic that aims at minimize the makespan subject to a temperature threshold while taking both spatial and temporal temperature evolution into account. A job is placed on a node to balance the loads of all nodes in a thermal-aware manner (e.g., with potential idle steps included to avoid violation of the temperature threshold). The nodes' temperatures are regulated via DVFS, in a concerted manner based again on their thermal-aware loads. Nodes are ensured to remain below the threshold temperature by choosing at which frequency each job should run or, in the case without DVFS capabilities, when a node should be temporarily suspended. In our experiments, we do not consider DVFS capabilities.

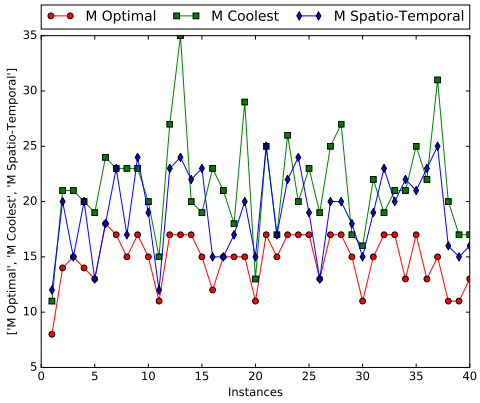
Neither heuristic aims to minimize energy, and Coolest does not even aim to minimize makespan, but both heuristics aim to schedule the workload from a thermal-aware perspective: Coolest aims to have a homogeneous thermal map in a datacenter while Spatio-Temporal aims to maintain the temperature below a threshold. Here, we only report on makespan results, since the energy obtained by the MILP and the two heuristics is the same. This is because, in the HPC setting, each job consumes a fixed amount of energy regardless of the schedule (due to using 0% or 100% of a node's compute capacity).

Figure 14(a) plots the makespan of the solution obtained by the MILP and those achieved by the two heuristics for each of our 40 problem instances, while Figure 14(b) shows average makespans. We can see that the results of the two heuristics are close: Spatio-Temporal is about 9.5% better than Coolest on average. Spatio-Temporal performs slightly better than Coolest because it is makespan-aware and regulates the temperatures of all nodes in a concerted way as compared to the distributed temperature regulation employed by Coolest. MILP is better by 21% compared to Spatio-Temporal and by 28.6% compared to Coolest (the average makespans computed by MILP, Coolest and Spatio-Temporal over the 40 instances are 15, 21 and 19, respectively). Neither heuristic attempts to solve the problem optimally: Coolest chooses the node with the lower temperature while Spatio-Temporal makes load balancing based on the concept of thermal-aware load [32]. These results quantify the "room for improvement" for both heuristics, at least on small instances. For these particular heuristics, the room for improvement is non-negligible, at about 21%, suggesting that striving for better heuristics may be a worthwhile endeavor.

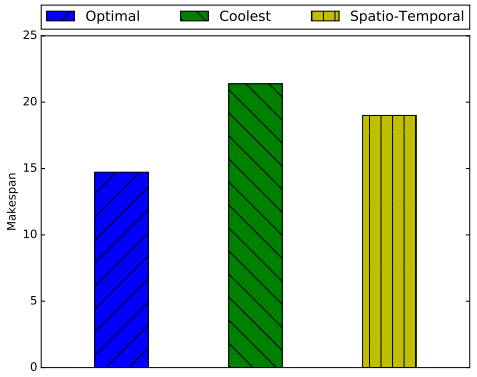
7 Conclusion and Future Work

In this paper, we have proposed MILP formulations for datacenter resource allocation problems with thermal constraints. These formulations are the first to take into account both spatial and temporal aspects of heat production and dispersion. Through several case-studies, we have shown the usefulness of these formulations for both makespan and energy optimizations in HPC and Cloud settings. Although the size of the problem instances is limited by the capabilities of our linear program solver and large instances are out of reach, the proposed MILP formulations are valuable for several reasons. In this paper, we have compared the optimal solutions obtained by MILP to the solutions computed by two polynomial-time heuristics [19], [32] that were previously proposed for makespan minimization in HPC setting. Our comparison provides an absolute measure of the efficacy of these heuristics.

Our main future direction is to work on improved mathematical formulations in order to find shortcuts and prune the search tree of the Gurobi solver (possibly assisted by computational intelligence design frameworks that are being utilized in smart design process), so as to be able to solve significantly larger problem instances. Designing improved heuristics, possibly inspired and informed by the MILP formulations (e.g., using relaxation



(a) Makespan for each instance



(b) Average makespan

Fig. 14: Makespan obtained by solving the MILP and by the Coolest and Spatio-Temporal heuristics.

techniques), is another future direction that is worth investigating.

References

1. Y. Bai, L. Gu and X. Qi. Comparative Study of Energy Performance between Chip and Inlet Temperature-Aware Workload Allocation in Air-Cooled Data Center Energies, 11(3):669, 2018.
2. N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. Journal of the ACM, 54(1):3:1–3:39, 2007.
3. A. Barkat and A. Capone. Effective management of green cloud data centers using energy storage technologies. In Proceedings of the 23rd International Conference on Software, Telecommunications and Computer Networks (SoftCOM), 2015.
4. BlueTool. http://impact.asu.edu/BlueTool/
5. D. Borgetto, H. Casanova, G. Da Costa, and J.-M. Pierson. Energy-aware service allocation. Future Gener. Comput. Syst., 28(5):94–125769–779, 2012.
6. D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. IEEE Micro, 20(6):26–44, 2000.
7. M. T. Chaudhry, T. C. Ling, A. Manzoor, S. A. Hussain and J. Kim. Comparative Study of Energy Performance between Chip and Inlet Temperature-Aware Workload Allocation in Air-Cooled Data Center ACM Computing Surveys, 47(3):39:1–39:48, 2015.
8. K. Ebrahimi, G. F. Jones, and A. S. Fleischer. A review of data center cooling technology, operating conditions and the corresponding low-grade waste heat recovery opportunities. Renewable and Sustainable Energy Reviews, 31(C):622–638, 2014.
9. C. Gu, L. Zhang, Z. He, H. Huang, X. Jia. Minimizing energy cost for green cloud data centers by using ESDs. In Proceedings of the 34th IEEE International Performance Computing and Communications Conference, 2015.
10. M. E. Haque, I. Goiri, R. Bianchini, and T. D. Nguyen GreenPar: Scheduling parallel high performance applications in green datacenters In Proceedings of the 29th ACM International Conference on Supercomputing (ICS), 2015.
11. C. Herzog and J. Pierson. A Generic Learning Multi-agent-System Approach for Spatio-Temporal-, Thermal- and Energy-Aware Scheduling In Proceedings of the Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), 2018.
12. B. Kantarci, L. Foschini, A. Corradi, H. T. Mouftah. Inter-and-intra data center VM-placement for energy-efficient large-scale cloud systems. In Proceedings of the First International workshop on Management and Security technologies for Cloud Computing, 2012.
13. H. Liu, B. Liu, L. T. Yang, M. Lin, Y. Deng, K. Bilal and S. U. Khan. Thermal-Aware and DVFS-Enabled Big Data Task Scheduling for Data Centers. IEEE Transactions on Big Data, 2(4):177–190, 2018.
14. G.I. Meijer. Cooling Energy-Hungry Data Centers. Science, 5976(328):318–319, 2010.
15. K. Metwally, A. Jarray and A. Karmouch. MILP-Based Approach for Efficient Cloud IaaS Resource Allocation. In Proceedings of the IEEE 8th International Conference on Cloud Computing, 2015.
16. Y. Mhedheb and A. Streit. Energy-efficient Task Scheduling in Data Centers. In Proceedings of the 6th International Conference on Cloud Computing and Services Science, 2016.
17. T. Mukherjee, A. Banerjee, G. Varsamopoulos, S. K. S. Gupta, and S. Rungta. Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers. Computer Networks, 53(17):2888–2904, 2009.
18. H. M. Mohammad Ali, T. E. H. El-Gorashi, A. Q. Lawey and J. M. H. Elmirghani. Future Energy Efficient Data Centers With Disaggregated Servers. Journal of Lightwave Technology, 35(24):5361–5380, 2017.
19. J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling "cool": temperature-aware workload placement in data centers. In USENIX Conference, 2005.
20. K. Mukherjee, S. Khuller and A. Deshpande. Algorithms for the thermal scheduling problem. In Proceedings of the IEEE International Parallel & Distributed Processing Symposium (IPDPS), 2013.
21. S. A. Nada and M. A. Said. Effect of CRAC units layout on thermal management of data center Applied Thermal Engineering, 118:339–344, 2017.
22. E. Pakbaznia and M. Pedram. Minimizing data center cooling and server power costs. In Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED), 2009.

23. D. Rajan and P. S. Yu. Temperature-aware scheduling: When is system-throttling good enough? In Proceedings of the International Conference on Web-Age Information Management (WAIM), 2008.
24. L. Ramos and R. Bianchini. C-Oracle: Predictive thermal management for data centers. In Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA), 2008.
25. A. Sansottera and P. Cremonesi. Cooling-aware workload placement with performance constraints. Performance Evaluation, 68(11):1232-1246, 2011.
26. R. Sharrock, T. Monteil, P. Stolf O. Brun. Autonomic computing to manage green Core networks with Quality of Service. In Proceedings of the Energy Efficiency in Large Scale Distributed Systems conference (EE-LSDS), 2013.
27. O. Sarood, P. Miller, E. Totoni, and L. V. Kale. "Cool" load balancing for high performance computing data centers. IEEE Transactions on Computers, 61(12):1752–1764, 2012.
28. K. Skadron, T. Abdelzaher, and M. R. Stan. Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. In Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA), 2002.
29. K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan. Temperature-aware microarchitecture: Modeling and implementation. ACM Transactions on Architecture and Code Optimization, 1(1):94–125, 2004.
30. M. Stillwell, D. Schanzenbach, F. Vivien and H. Casanova. Resource Allocation Algorithms for Virtualized Service Hosting Platforms J. of Parallel and Dist. Comp., 70(9):962-974, 2010.
31. H. Sun, P. Stolf, J.-M. Pierson, and G. Da Costa. Energy-efficient and thermal-aware resource management for heterogeneous datacenters. Sustainable Computing: Informatics and Systems, 4(4):292–306, 2014.
32. H. Sun, P. Stolf, and J.-M. Pierson. Spatio-temporal thermal-aware scheduling for homogeneous high-performance computing datacenters. Future Gener. Comput. Syst., 71C:157-170, 2017.
33. Q. Tang, S. K. S. Gupta, and G. Varsamopoulos. Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach. IEEE Transactions on Parallel and Distributed Systems, 19(11):1458–1472, 2008.
34. Q. Tang, T. Mukherjee, S. K. S. Gupta, and P. Cayton. Sensor-based fast thermal evaluation model for energy efficient high-performance datacenters. In Proceedings of the Fourth International Conference on Intelligent Sensing and Information Processing (ICISIP), 2006.
35. T. Van Damme, C. De Persis and P. Tesi. Optimized Thermal-Aware Job Scheduling and Control of Data Centers. IFAC-PapersOnLine, 50(1):8244-8249, 2017.
36. L. Wang, S. U. Khan, and J. Dayal. Thermal aware workload placement with task-temperature profiles in a data center. Journal of Supercomputing, 61(3):780–803, 2012.
37. J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin. Dynamic Thermal Management through Task Scheduling. In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and software (ISPASS), 2008.
38. F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS), 1995.
39. S. Zhang and K. S. Chatha. Approximation algorithm for the temperature-aware scheduling problem. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2007.
40. V. Villebonnet, G. Da Costa, L. Lefevre, JM. Pierson, and P. Stolf. Dynamically Building Energy Proportional Data Centers with Heterogeneous Computing Resources (short paper). In IEEE International Conference On Cluster Computing (CLUSTER 2016), Taipei, Taiwan, 2016.