Simulation and Optimization of HPC Job Allocation for Jointly Reducing Communication and Cooling Costs

Jie Meng[†], Samuel McCauley[*], Fulya Kaplan[†], Vitus J. Leung[‡], and Ayse K. Coskun[†]

Abstract—Performance and energy are critical aspects in high performance computing (HPC) data centers. Highly parallel HPC applications that require multiple nodes usually run for long durations in the range of minutes, hours or days. As the threads of parallel applications communicate with each other intensively, the communication cost of these applications has a significant impact on data center performance. Energy consumption has also become a first-order constraint of HPC data centers. Nearly half of the energy in the computing clusters today is consumed by the cooling infrastructure. Existing job allocation policies either target improving the system performance or reducing the cooling energy cost of the server nodes. How to optimize the system performance while minimizing the cooling energy consumption is still an open question. This paper proposes a job allocation methodology aimed at jointly reducing the communication cost and the cooling energy of HPC data centers. In order to evaluate and validate our optimization algorithm, we implement our joint job allocation methodology in the Structural Simulation Toolkit (SST) – a simulation framework for large-scale data centers. We evaluate our joint optimization algorithm using traces extracted from real-world workloads. Experimental results show that, in comparison to performance-aware job allocation algorithms, our algorithm achieves comparable running times and reduces the cooling power by up to 42.21% across all the jobs.

Index Terms—High-performance Computing, Data Center, Job Allocation, Joint Optimization, Cooling Energy, Communication Cost

1 Introduction

High performance computing (HPC) data centers deploy thousands of servers working closely together to solve complex problems. Even though the throughput of computing systems has increased tremendously over the last decade, the need for higher performance is on-going and will not disappear in the near future. Today's HPC data centers have highly parallel applications (such as scientific, financial or other applications) running on multiple data center nodes. The durations of such applications are usually in the range of minutes, hours, or even days. The running time is affected by the frequent communications between the threads of these parallel applications, which exchange data and messages through communication infrastructures such as the message passing interface (MPI). Thus, the performance of a communication-intensive application is highly dependent on the location of the individual computing units that are communicating with each other. Recent research has demonstrated that the communication cost of communication-intensive applications has a significant impact on overall system performance in HPC data centers (e.g., [1], [2], [3]). Leung et al. have shown that if two communication-intensive jobs are manually placed on the machine so that their communication paths overlap significantly, the running times of both jobs are approximately doubled [1].

In order to maximize the throughput and avoid bandwidth contention resulting from overlapping communication paths, threads of a job should be allocated on nodes that are close to each other (e.g., [4], [1], [5]). Existing performance-aware job allocation algorithms mostly focus on minimizing the average number of communication hops among the communicating nodes. For example, some researchers propose contiguous job allocation schemes, for which the allocated nodes are adjacent (e.g., [4], [6]). Other schemes allow discontiguous allocation to avoid fragmentation of available processors (e.g., [2], [3]). There are also performance-aware algorithms developed for transactional enterprise loads for satisfying the response time constraints imposed by service level agreements (e.g., [7]). Unfortunately, the workload and performance models of such algorithms are not applicable to parallel HPC applications with intensive communication among the tasks.

Along with the data center computational capacities,

- [†]J. Meng, F. Kaplan, and A. K. Coskun are with the Department of Electrical and Computer Engineering, Boston University. 8 St. Mary's Street, Boston University Boston, MA, 02215. E-mail: jiemeng@bu.edu, fkaplan3@bu.edu, acoskun@bu.edu.

  [*]S. McCauley is with the Department of Computer Science at Stony Brook University. Department of Computer Science, Stony Brook University, Stony Brook, NY 11794-4400. He is supported in part by NSF grants CCF 1114809, CCF 1217708, IIS 1247726, and IIS 1251137, and by Sandia National Laboratories. E-mail: smccauley@cs.stonybrook.edu.

  [‡]V. J. Leung is with Sandia National Laboratories, P.O. Box 5800, Albuquerque, NM 87185. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000. E-mail: vjleung@sandia.gov.

energy consumption of HPC data centers has increased tremendously. It has been reported that the worldwide data center electricity consumption increased by 56% from 2005 to 2010, which accounted for 1.3% of the total electricity use [8]. A recent review [9] shows that for every dollar spent on power of data center computing equipments, another dollar is spent on data center cooling infrastructures, which translates to an energy cost reaching up to millions of dollars and cooling costs reaching close to half of the overall energy cost [10], [11], [12]. Thus, maintaining cooling and energy efficiency is becoming one of the main constraints in the design and management of today's data centers.

In order to address the cooling challenge in data centers, prior work has proposed various cooling-aware job allocation policies (e.g., [13], [14], [15], [16], [17]). In efficient cooling, one of the key factors is to keep the computer room air conditioner (CRAC) supply temperature as high as possible [13]. Therefore, many techniques focus on decreasing the node inlet temperatures by allocating power to nodes according to their recirculation contribution in the data center or minimizing cooling power through minimizing the peak node inlet temperature (MPIT) (e.g., [13], [14]). There are also strategies that minimize the sum of server and cooling power in a data center by turning off some of the servers and chassis or deciding on the voltage/frequency setting for the active servers (e.g., [15], [16], [17]).

We observe that the performance-aware and cooling-aware optimization algorithms for job allocation in HPC data centers have been developed rather independently. How to find an optimization algorithm that is able to reduce the cooling energy cost and, at the same time, improve or achieve comparable performance on the HPC system is an open question. In order to address this problem, we have proposed a joint optimization heuristic in our recent work for reducing both the cooling power and the communication latency in an HPC data center at the same time [18].

This paper expands upon our prior work by providing an implementation and evaluation on a system-level simulation framework – the Structural Simulation Toolkit (SST), which is developed by Sandia National Laboratories for simulating the performance of large-scale data centers. Instead of solely using pairwise L_1 distance to estimate communication cost [18], we propose a performance model that computes the impact of communication cost on system performance based on experimental data collected from HPC applications running on real data centers. We target HPC data centers that are running multi-threaded workloads with heavy communication among the threads. Our main goal is to deliver the desired performance and reduce cooling energy as much as possible. The specific contributions of this paper are as follows:

- We propose a performance modeling approach to estimate the impact of communication cost on HPC system performance. The performance model is based on a combination of the L_1 pair-wise distance, experimental data from HPC applications in real data centers, and some randomization to account for the variations.
- We design and implement a job allocation method to jointly optimize the performance and the cooling energy consumption of a data center. We implement our performance model, cooling energy model, and joint optimization algorithm in SST, which has the ability to evaluate various scheduling and allocation algorithms for large-scale data centers.
- Experimental results show that, our joint optimization algorithm achieves comparable running times with those of performance-aware policies, while reducing the cooling power by up to 42.21% compared to performance-aware allocation.

The rest of the paper starts with a discussion of the related work. Section 3 presents the performance, server power, and data center cooling models. We introduce the performance and cooling optimization problems separately, and then, describe our joint allocation strategy in Section 4. We present the experimental evaluation in Section 6 and conclude in Section 7.

2 Related Work

A lot of recent research has been done to improve the performance and energy efficiency of data centers. In this section, we first review the related work on data center job allocation algorithms for optimizing the performance of HPC applications. We then discuss the existing methods that focus on reducing the data center cooling energy costs.

2.1 Performance-Aware Job Allocation

Most of the performance-aware job allocation algorithms for HPC data centers and supercomputers typically focus on minimizing the average number of communication hops among processors on which a job is running. Some of the job allocation algorithms allocate only *contiguous* (i.e., touching, in close proximity) sets of processors to each job (e.g., [4], [6], [19]), as contiguous node allocation provide significant reduction in execution time for communication-intensive parallel programs. For example, Bhattacharya et al. propose a heuristic for job allocation in a mesh-connected parallel processor [4]. They use a lookahead idea by analyzing the queue of waiting jobs and propose an algorithm to detect free submesh area for efficient allocation. However, such contiguous allocation algorithms may result in external fragmentation of available processors (i.e., available nodes that are separated from each other cannot be utilized) and reduce the achievable system utilization.

A number of recent proposed algorithms on communication-aware job allocation allow *discontiguous* allocation of processors (i.e., the processors given to a job do not need to be next to each other). For example, Mache et al. present the MC allocation strategy for mesh-connected parallel computers. Their method yields compact allocations by containing the jobs in the smallest rectangular area possible [2]. Motivated by the MC allocation strategy, Bender et al. propose an MC1x1 processor-allocation algorithm, in which the first sub-mesh is a 1X1 shell and subsequent sub-meshes grow in the same way as in MC [3]. Walker et al. discuss fast algorithms to allocate processors to compute jobs in mesh-connected clusters [5] by ordering processors according to some curve and using several buddy-system strategies. However, these existing performance-aware job allocation strategies solely target the performance and communication costs without considering the potential impact of job allocation on the cooling costs.

2.2 Data Center Cooling Management

As thermal management and reducing the cooling costs are among the dominant concerns for today's data centers, a number of thermal modeling and management techniques at data center level have been proposed recently. Kim et al. use a linear formula with input parameters of ambient room temperature, thermal resistance between die and air, and server power to find server temperatures [20]. However, their model does not consider the effect of recirculation on temperature. Wang et al. compute the temperature of a compute node as a combination of an RC-thermal model and a task-temperature profile [21]. This model assumes that the thermal map of the data center is available through input ambient sensors and on-board sensors. Moore et al. carry out computational fluid dynamics (CFD) simulations to conduct thermal evaluation, which cannot be used for real-time data center thermal management because the computation times are too long [13]. Heath et al. introduce a data center temperature emulation suite called Mercury that emulates temperatures based on the data center layout, hardware, and component utilizations [17]. Tang et al. propose a linear model to compute data center temperatures and cooling energy costs taking recirculation into account [14].

Utilizing the linear model for computing data center temperatures and cooling energy, Tang et al. introduce an optimization problem for minimizing the peak node inlet temperature (MPIT) through job assignment [14]. They use both a genetic algorithm and sequential quadratic programming to solve the problem. Moore et al. present two temperature-aware workload placement algorithms: the first assigns workloads to servers based on location-aware discretization heuristics and the second minimizes the heat recirculating within the data centers [13]. Pakbaznia et al. propose Minimum Total Data Center Power (MTDP) algorithm to minimize the total of server and cooling power in a data center by turning off some of the servers and chassis and deciding on the voltage/frequency setting for the servers [15]. These techniques focus on reducing temperature and cooling cost of data centers without considering the impact of the workload allocations on application performance. Sansottera et al. proposes the Greedy Least Power (GLP) algorithm to minimize the power consumption while satisfying response time constraints imposed by service level agreements [7]. Their main focus is not HPC applications with intensive communication; thus, their model does not consider communication latency during allocation.

2.3 Distinguishing Aspects from Prior Work

Our work differentiates from prior research as our job allocation policy optimizes both the application performance in terms of reducing the communication cost and the cooling energy cost of HPC data centers. Our policy confines the communicating nodes of a job in close proximity, while at the same time, selecting the most cooling-efficient locations possible. In comparison to our prior work [18], we propose a performance model that estimates the impact of communication cost on the performance of HPC applications by utilizing experimental data from a real data center. We implement the joint optimization algorithm in the SST simulation framework and compare the impact of prior performance-aware and cooling energy-aware job allocation policies on system performance and cooling energy. We demonstrate that for jobs involving intensive communication among the nodes, application performance becomes an important factor in determining the cooling energy.

3 System Modeling Methodology

In this section, we present the modeling methodology for our target HPC center. We first introduce the layout of the data center. Then we provide the performance simulation setup, which is used for computing the communication delays in the data center. The last part of the section describes the server power and cooling energy models.

3.1 Target Data Center System

Our target system in this paper is a small size data center with two rows of industry standard racks arranged in a layout shown in Figure 1. In this arrangement, rack inlets where the cool air is supplied are facing the outer aisles forming cold aisles at the sides. Rack outlets, where the hot air exits, are facing each other forming a hot aisle in between the two rows. Each row is composed of 5 racks and each rack has 4 compute *nodes*. In our experiments, we assume that each *node* includes 10 servers and each server has 2 processors. This layout corresponds to a total of 800 processors across the two rows of the data center. The proposed data center setup has been widely used in prior work and is a small-scale representative of today's data center configurations [7]. Our methodology and proposed policy are scalable to larger data centers.

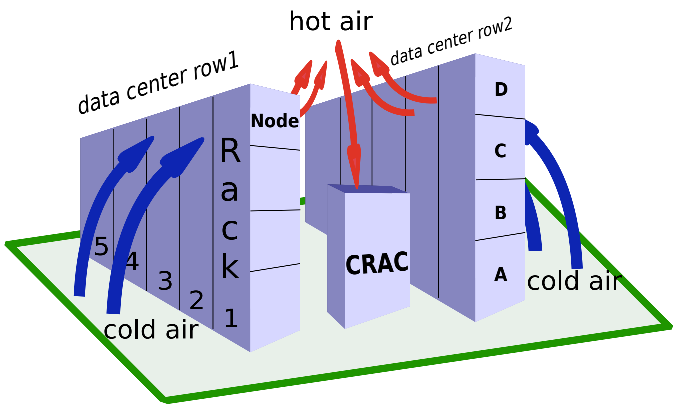

Fig. 1: Layout of the data center.

3.2 Performance Model

We design job allocation algorithms for mesh-connected HPC data centers and supercomputing systems. Mesh-connected networks for message passing are widely used in many experimental and commercial distributed-memory parallel computers, such as IBM BlueGene/L and Cray X systems.

We specify our workloads as jobs that request a number of nodes in the data center. This specification is in line with the practice in HPC data centers and has also been used in standard workload formats [22]. We base our communication cost computation on the average pairwise L_1 distance (Manhattan distance) across all the communicating nodes of a job running on the mesh-connected parallel system [3]. We employ L_1 distance as our metric as it has been demonstrated to correlate with application running time [1]. We call the total L_1 distance among the nodes of a job the *communication cost* and formulate it as follows:

$$CC_{job} = \frac{1}{n} \sum_{(s,t)\in(S,T)} [w_x(s,t) + w_y(s,t) + w_z(s,t)] \quad (1)$$

where n is the job size (the number of nodes a job requires), (s,t) is the pair of source and destination nodes of a message and (S,T) is the set of all the source and destination node pairs for all the messages. In this work, we assume $n > 1$ for all jobs. $w_x(s,t)$, $w_y(s,t)$ and $w_z(s,t)$ represent the distance between s to t along the x, y, and z-axes, respectively. Division by n provides the normalization with respect to job size; thus, CC_{job} gives the communication cost of a job per node.

In this paper, we assume *all-to-all* communication pattern for our workloads. *All-to-all* is a common communication pattern in HPC routines such as Fast-Fourier-Transform (FFT), which is part of several applications including molecular dynamics, quantum chemistry, and digital signal processing [23]. In an *all-to-all* pattern, each processor communicates with all the other processors that are running the threads of the same job.

In order to model the impact of communication cost on system performance, we use a combination of the L_1 pairwise distance CC_{job}, running time measurements from real-life traces, and some randomization to formulate a delay factor. In our model, the total running time of the application is composed of computation time and communication time. Communication time of the application is a function of job communication cost, which we call as delay factor and denote as $\tau(CC_{job})$. Equation (2) shows the total running time of an application:

$$t' = 0.7 \times t + 0.3 \times \tau(CC_{job}) \times t \quad (2)$$

where t is the running time of the application without considering the communication cost arising from job allocation decisions, $\tau(CC_{job})$ is the delay factor resulting from the communication cost and t' is the actual resulting running time of the job.

In order to extract the coefficients of the function τ, we experimented with miniGhost [24], an application with *all-to-all* communication pattern, on a real data center at Sandia National Laboratories. MiniGhost is modeled on the computational core of CTH [25], from which it inherits nearest neighbor and all-to-all communication patterns. We ran our experiments on Cielo [26], number 26 on the November 2013 Top500 list [27]. It is a Cray XE6 organized as a 3D torus and which uses non-contiguous allocation.

Figure 2 shows the relationship between the average hop distance and the delay factor τ. As seen from Figure 2, there is variation in the experimental results. We model this variation as r, which is a random number scaled by the L_1 pair-wise distance. By applying linear regression to fit the experimental data, we come up with

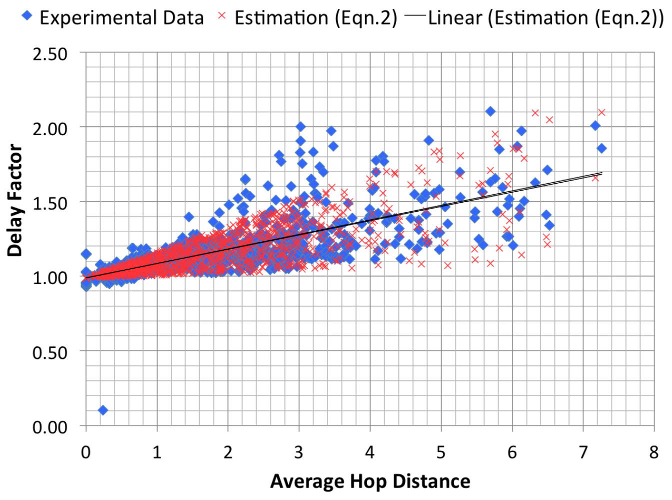

Fig. 2: The modeling of delay factor for the running time of applications as a function of the communication hop distance.

Equation (3) as follows:

$$\tau = 0.9875 + (0.0962 \times CC_{job}) + r \tag{3}$$

3.3 Data Center Server Power Model

The power consumption of data center severs is the dominant factor that determines the server temperatures and the cooling energy required to maintain safe temperatures. Power consumption of the servers varies with their activity. Based on reported power consumption values of several data center servers [7], [28], we assume that a server consumes 250W during computationally-intensive phases, 200W during communication-intensive phases, and 100W when idle.

Power consumption is lower during the communication phases due to the lower CPU activity and the time spent in I/O. We target large HPC jobs and assume that each job fully occupies a set of nodes (i.e., all of the servers in a node would be assigned to the same job). Therefore, for each node, each containing 10 servers, the total power of the node is 2500W, 2000W, or 1000W, depending on the activity.

We compute the total power of a node while executing a job as the weighted sum of the computation and communication power. We observe that HPC jobs spend a large percentage of time in communication, and the ratio of the communication time to total running time typically varies between 20% to 40% [29], [30]. We select the ratio of communication time to running time as 30% in this work, which represents an average case for HPC. Our experimental choices are set similarly with real-life scenarios in today's HPC clusters. Without loss of generality, the proposed techniques are applicable to data centers with different server power levels and applications with different communication to computation ratios. Section 6 discusses how the results vary as the power and communication activities change.

3.4 Cooling Energy Model

In the typical data center layout we use, racks and perforated vent tiles are placed on a raised floor. Cold air enters the room from the floor tiles, goes into the rack inlets from the sides, and gets hotter as it moves through the racks. Hot air exits the rack from the back into the center aisle and the exhaust air exits the room from the ceiling to be cooled again. This set-up is called hot aisle/cold aisle arrangement which avoids mixing cold supply air with exhaust air.

In order to compare different job allocation strategies, we need a fast and accurate data center thermal model. We use the model proposed and validated by Tang et al [31]. Their model combines a linear, low complexity heat recirculation model with a linear power model. This model is more practical than most other existing models as it requires a set of CFD simulations only once to characterize the data center. Once we have the measured data center specific parameters, the vector of inlet temperatures, $\boldsymbol{T_{in}}$, for all the nodes are computed using the following linear equation:

$$\boldsymbol{T_{in}} = \boldsymbol{T_{sup}} + \boldsymbol{DP} \tag{4}$$

$$\boldsymbol{D} = [(\boldsymbol{K} - \boldsymbol{A^T K})^{-1} - \boldsymbol{K^{-1}}] \tag{5}$$

where T_{sup} is the CRAC unit supply temperature vector, **D** is the heat distribution matrix and **P** is the node power vector. **K** is the thermodynamic constant matrix and **A** is the heat cross-interference coefficient matrix representing the recirculation phenomena. **K** is calculated as:

$$\boldsymbol{K} = diag(K_i) \tag{6}$$

$$K_i = \rho f_i c_p \tag{7}$$

where ρ=1.19 Kg/m^3 is the density of air, f_i=0.2454 m^3/s is the flow rate of node i (assumed fixed for all nodes), and c_p=1005 J/KgK is the specific heat of air [14].

Matrix **A** represents the fraction of output heat from each node that is recirculated to the inlet of other nodes. It is an $n \times n$ matrix for a system with n nodes with each term a_{ij} representing the fraction of heat at node i recirculating back into node j. It has been shown that elements of matrix **A** mostly depend on the data center layout rather than the power consumption of the nodes or the supply temperature [7]. Therefore, this matrix is obtained once for a data center. The matrix is calculated through CFD simulations in prior work [31]. If one has input ambient sensors mounted already, **A** can be obtained using sensor measurements instead of CFD simulations and following the same procedure in [31]. We use the coefficients for the data center given in [7].

Figure 3 shows the cross-interference coefficient matrix for the 40-node system in colormap and 3-D formats. We insert the colormap plot given in [7] into MATLAB to extract the coefficient value corresponding to each data point in the image. We map RGB values to indexes, which preserve the relationship between coefficients relative to each other. Next, we perform calibration by scaling the matrix according to the given temperature graph in [7]. For a data center with different layout and heat flow characteristics, matrix **A** differs. However, the equations to calculate the inlet temperatures are independent of the data center; in other words, the model applies to data centers in general.

The power values of data center nodes are calculated based on the job allocation. We calculate the inlet temperatures of data center nodes resulting from different allocation schemes using Equation (4). We perform node-level allocation in this paper, which is a reasonable hierarchical level for HPC data centers. Assume a given task of size n, which corresponds to the total number of nodes a task requires, and an integer variable x_i showing whether node i is assigned a job or not (i.e., it is either 1 or 0, respectively). Power consumption of node i can be expressed with a linear model as follows:

$$P_i = P_{idle} + x_i P_{util} \tag{8}$$

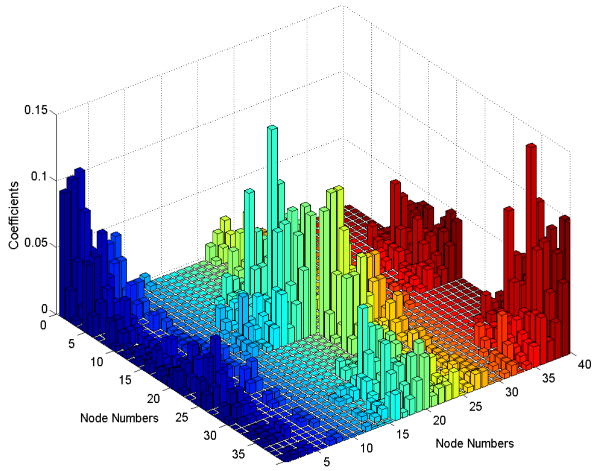

Fig. 3: Cross-interference coefficient matrix (A) for our data center. A higher bar at a coordinate (i, j) indicate stronger recirculation impact of node i to node j.

where P_{idle} is the node idle power and P_{util} is the power consumed by a node when running a task. This assumption of fixed node power when running a task is reasonable for HPC data centers, because even though there are fluctuations in the power, they are much smaller compared to the total power.

Once we get the node inlet temperatures using Equation (4), we need a cooling power model to estimate the power consumed by the cooling unit at various temperatures. This power depends on the efficiency of the CRAC unit. One of the most common metrics used for CRAC unit efficiency is the coefficient of performance (CoP). CoP is defined as the ratio of the heat removed from the system to the energy spent on cooling and has the following formula:

$$CoP = \frac{P_c}{P_{AC}} \tag{9}$$

where P_c is the total computing power (sum of the values in $\mathbf{P}$ vector) and P_{AC} is the cooling power. CoP increases with higher CRAC supply temperature (T_{sup}). In this work, we use the CRAC unit CoP model given in prior work [13] as follows:

$$CoP(T_{sup}) = 0.0068 {T_{sup}}^2 + 0.0008 T_{sup} + 0.458 \tag{10}$$

where T_{sup} is in Celsius. The upper limit on how much we can increase supply temperature (T_{sup}) depends on the difference between redline temperature (T_{red}), which is the highest allowed temperature at the node inlets, and maximum node inlet temperature ($T_{in,max}$). In other words, we can use this temperature slack to increase the supply temperature and operate at higher CRAC efficiency without violating the temperature constraints. A new supply temperature is found by adding this difference to T_{sup} and cooling cost is calculated as follows:

$$T_{sup'} = T_{sup} + T_{red} - T_{in,max} \tag{11}$$

$$P_{AC} = \frac{P_c}{CoP(T_{sup'})} \tag{12}$$

$$P_{AC savings} = \frac{P_c}{CoP(T_{sup})} - \frac{P_c}{CoP(T_{sup'})} \tag{13}$$

This model provides fast results as it does not require time consuming CFD simulations and it is able to capture the effect of recirculation, which has a significant contribution in server temperatures in current data centers.

4 Optimization Methodology

In this section, we first formulate and solve the cooling energy optimization and communication cost optimization problems individually. For cooling cost minimization, we use the Minimize Peak Inlet Temperature (MPIT) algorithm [14]. For communication cost minimization, we deploy the MC1X1 algorithm [3]. We then propose a job allocation algorithm, which takes both cooling efficiency and communication latency into consideration.

4.1 Cooling-aware Job Allocation Policy

The maximum cooling energy saving is achieved when the maximum inlet temperature in the data center is minimized [14]. Therefore, a cooling-aware allocation policy such as MPIT [14]assigns jobs to nodes so that the resulting max$\{\boldsymbol{T_{in}}\}$ will be minimum. We formulate the optimization problem of allocating a job to an idle data center with minimal cooling energy as follows:

$$\begin{aligned} \underset{X_{dcenter}}{\text{minimize}} \quad & max\{\boldsymbol{T_{in}}(\boldsymbol{X_{dcenter}})\} \\ \text{subject to} \quad & \sum_{i=1}^{N} x_i = n_{dcenter} \qquad x_i \in \{0,1\} \end{aligned} \tag{14}$$

where $\boldsymbol{X_{dcenter}}$ is a vector described as $\boldsymbol{X_{dcenter}} = \{x_1, x_2, ..., x_N\}$, where x_i $(i = 1, \ldots, N)$ represents whether node i is assigned a job or not. Vector $\boldsymbol{X_{dcenter}}$ shows all of the busy nodes in the data center corresponding to *currently* and *previously* allocated jobs that are still running. $n_{dcenter}$ is the sum of the sizes of all jobs running on the data center. $\boldsymbol{T_{in}}$ represents the vector of inlet temperatures of a data center, which is defined in Equation (15).

$$\boldsymbol{T_{in}}(\boldsymbol{X_{dcenter}}) = \boldsymbol{T_{sup}} + \boldsymbol{D} \cdot \boldsymbol{P_{idle}} + \boldsymbol{D} \cdot \boldsymbol{X_{dcenter}} \cdot \boldsymbol{P_{util}} \tag{15}$$

where T_{sup} is the CRAC unit supply temperature, D is heat distribution matrix. P_{idle} and P_{util} are the idle and dynamic power for the nodes. Note that while allocating each job, we use additional constraints to represent the currently busy nodes. For example, if nodes 1, 2 and 3 are busy at the time of allocation, we add the constraints x_1=1, x_2=1, x_3=1 to solve the problem.

As described in Section 3.4, cooling cost is highly dependent on the CRAC supply temperature T_{sup}. Therefore, if we can increase T_{sup} as much as possible without causing the nodes to exceed the redline temperature, we can reduce the cooling power accordingly. The maximum allowed T_{sup} increase, therefore, is limited by the maximum inlet temperature max$\{\boldsymbol{T_{in}}\}$.

In our MPIT implementation within SST, every time a new job arrives, we formulate and solve the optimization problem in C++ using the GNU Linear Programming Kit (GLPK) [32]. We use Equations 14 and 15 to obtain an integer linear program, then relax the constraints to allow a job to be assigned fractionally to some nodes. This results in a linear program which we solve using GLPK. GLPK returns a real number solution x_{real} and we use the discretization algorithm suggested in prior work [14] to convert it to the nearest integer solution x_{int} that obeys the constraints. This algorithm was shown to give the highest power savings among various other approaches [14]. For various allocations, we have compared the max$\{\boldsymbol{T_{in}}\}$ of both real and integer solutions and confirmed that the results are the same to the second decimal point.

4.2 Performance-aware Job Allocation

A performance-aware job allocation algorithm tries to maximize performance when it assigns processors to a job. This is done by grouping the assigned processors close together. Specifically, we try to minimize the average number of hops (i.e., the L_1 distance in a mesh) between each pair of allocated nodes.

Minimizing the average pairwise L_1 distance optimally is still an open problem, so several approximation algorithms have been developed. Two of these have already been implemented as allocation algorithms in SST. The first, the Manhattan Median, or MM algorithm, is a $2 - \frac{1}{2d}$ approximation in a d-dimensional mesh. The second, MC1x1, is a $(2-2/k)d$ approximation algorithm when k processors are desired [3]. While MM has better provable performance, it is much slower than MC1x1. Our paper focuses on the more feasible MC1x1 algorithm, which is what we will modify to obtain our hybrid algorithm.

At a high level, the MM algorithm works as follows. The algorithm considers any node as a potential center point. For each of these potential centers, it finds the k closest nodes by the L_1 metric, and calculates the average pairwise L_1 distance between them. It returns the set of k such points with the smallest average L_1 distance.

The MC1x1 algorithm uses the same high-level strategy as MM, but differs in several details. MC1x1 only considers free nodes as potential centers. Nodes are chosen by gradually building up *shells*, squares of increasing length centered on the given center node. This is equivalent to using the L_∞ metric instead of the L_1 metric in MM. Finally, the last shell may be partially empty; in fact the job may only need one or two nodes in the final shell. The MC1x1 implementation in SST attempts to cluster the processors in the final shell, which seems to improve the average pairwise L_1 distance in practice.

4.3 Joint Optimization Policy

In this section, we provide the details of our optimization policy, which aims at jointly optimizing the cooling energy and communication cost of applications running in an HPC data center.

As discussed in Sections 4.1 and 4.2, the existing performance-aware and cooling-aware job allocation policies minimize the communication latency and cooling power independently, which means that the resulting allocations may not be successful when both objectives are considered simultaneously. Cooling-aware job allocation needs to consider the layout of the data center as the recirculation effect changes depending on the location of the active nodes. In most cases, cooling-aware policy allocates jobs to the nodes located far from each other. For example, for a job of size 4, cooling-efficient allocation distributes the job equally among the data center rows in order to minimize the peak inlet temperature. This causes very high communication latency for cooling-aware policy. On the other hand, performance-aware MC1X1 policy confines the nodes of each allocated job into the smallest possible *shell*. It follows a regular pattern to allocate the jobs in the data center and arbitrarily breaks ties. It does not take into account whether an allocation results in high temperature as long as the allocated nodes are within the smallest shell possible, potentially causing inefficient cooling.

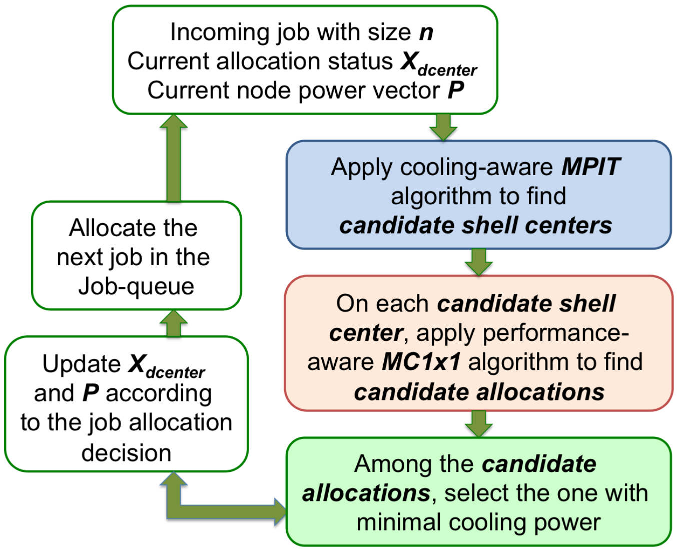
Incoming job with size ***n***
Current allocation status ***X***
Current node power vector ***P***
Apply cooling-aware ***MPIT*** algorithm to find ***candidate shell centers***
On each ***candidate shell center***, apply performance-aware ***MC1x1*** algorithm to find ***candidate allocations***
Among the ***candidate allocations***, select the one with minimal cooling power
Update ***X***dcenter and ***P*** according to the job allocation decision
Allocate the next job in the Job-queue

Fig. 4: A flow chart for illustrating our joint optimization algorithm.

In order to optimize the performance and cooling energy at the same time, we design a heuristic algorithm combining both policies. As illustrated in Figure 4, our algorithm first considers the cooling-aware job allocation solution. It then uses the resulting nodes as candidates for shell centers to apply the performance-aware job allocation policy. Then, we break the ties of possible performance-aware job allocations by selecting the allocation with minimal peak inlet temperature.

We modify the MC1X1 algorithm to make it open a shell centered at a given input node (possible shell center) accordingly. In MC1X1, opening a shell centered at a node refers to finding the smallest square-shaped area to include all nodes of a job. Starting from the smallest shell (1 square unit), the number of available nodes in the shell is checked. If the size of the job is larger than the available nodes, the shell is expanded.

When the available node count is met, the policy examines whether there are multiple allocation options within the shell area. For example, assume that we have a shell with 9 nodes, 3 of whom are busy, and we will assign a job of size 4 to the rest. In this case, we choose the 4 nodes with minimum communication cost possible. The resulting selection is the possible allocation corresponding to that *possible shell center*. For each *possible shell center*, the revised MC1X1 algorithm gives an allocation vector, *possible_X_dcenter*. Among those vectors, we select the most cooling efficient one (i.e., resulting in smallest peak inlet temperature). For example, assume that for a job i of size 3, the cooling-aware policy assigns the job to nodes 1, 4, and 5. We open shells centered at those nodes and select the one with the smallest inlet temperature. For the cases where two or more allocations result in the same inlet temperature but different communication costs, we find and choose the smallest communication cost allocation.

5 SIMULATION INFRASTRUCTURE

In this paper, we target communication-intensive parallel applications that use high-level message passing interfaces such as MPI. It is impractical to explore the vast design space of parallel resource management algorithms for such data centers without a detailed system-level simulation framework. Addressing the challenges of managing HPC centers running communication-intensive parallel applications requires design guidelines from simulation studies. This section introduces the simulation framework (SST) we use for evaluating our joint optimization algorithm managing real-world parallel workloads, as well as the implementations of job scheduler and allocation algorithms in SST.

5.1 Structural Simulation Toolkit (SST)

We implement our evaluation models and allocation optimization methods in SST, the Structural Simulation Toolkit. SST is an architectural simulation framework designed to assist in the design, evaluation, and optimization of HPC architectures and applications [33]. It is developed by Sandia National Laboratories to evaluate the performance of computer systems ranging from small-scale single-chip processors to large-scale parallel computing architectures.

The simulation in SST is based on component-based discrete events. The concept of component in SST refers to one of the simulating elements. For example, Gem5 [34] is a component that has been integrated in SST for simulating performance. SST synchronizes the events from different components through communication between components [35]. The scheduler is implemented as one of the components of SST for simulating the workload scheduling and job allocation in HPC data centers. In our evaluation of job allocation algorithms, we use the $EASY$ scheduler for all our experiments in SST. The $EASY$ scheduler also known as aggressive backfilling only gives a guaranteed start time to the first job in the queue at any time [36]. However, this is enough to prevent starvation for all jobs.

SST reads jobs from a simulation file and sends it to the scheduler. The scheduler runs each waiting job as processors become available; the exact start time of each job depends on the scheduler being used. The allocator then assigns a set of processors to run the job. SST figures out the running time of the job based on the data in the simulation file and the processors which are allocated to it. We have extended SST such that after the allocation decision is made, SST calculates the server inlet temperatures and cooling energy of the HPC center. Once the job finishes, SST frees the processors, leaving them available to run other jobs.

We update the timing based on both the original job length from the simulation and the quality of the allocation. Section 3.2 presents how the performance of an HPC application is modeled to reflect the impact of communication cost resulting from different allocation decisions.

5.2 Job Scheduler Implementation in SST

The *scheduler* component handles the placement of jobs onto a simulated system. To do this, it simulates scheduling, the decision of when jobs run, and processor allocation, the decision of which processors they should be assigned to.

The scheduler in SST was designed using an object-oriented approach. Figure 5 shows a simplified class diagram. The scheduler has a `Main` class which uses command line arguments to identify the desired simulation, parses a trace derived from the Parallel Workloads Archive [22] for characteristics of the jobs to run, and manages events for job arrivals and completions. `Main` uses objects from the abstract classes `Scheduler`, `Allocator`, and `Machine` to store most of the simulation state and to make decisions. Classes derived from these implement different versions of the necessary

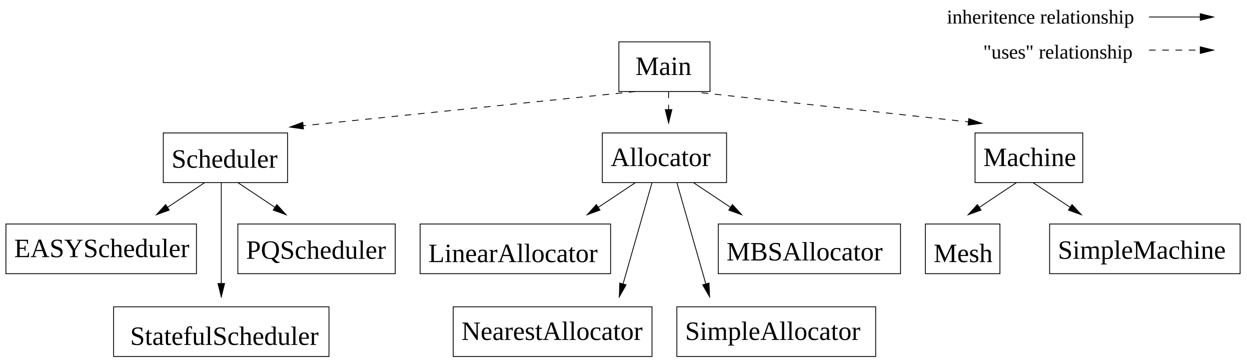

Fig. 5: Simplified class diagram for scheduling/allocation component/element.

functionality, with the abstract base classes enforcing a shared interface so that different combinations of schedulers and allocators can be used together as needed for different research projects.

Common schedulers and allocators have been implemented for use as baselines. Provided schedulers include EASY [36] (`EASYScheduler`), a generalization of Conservative [37] (`StatefulScheduler`), and a priority-based scheduler (`PQScheduler`). The latter uses a set of comparators (not shown in Figure 5) to select between first come first serve (FCFS), Shortest Job First, Widest Job First, and similar priority-based schemes. Most of the allocators are meant for mesh systems, with classes derived from `Allocator` implementing allocators based on linear orderings [38], [1], [5] (`LinearAllocator`), center-based allocators [2], [39], [3] (`NearestAllocator`), and buddy systems [38], [5] (`MBSAllocator`). In addition, to speed up simulations where scheduling is the focus and allocation is unnecessary, the system provides a "bag of processors" with no notion of locality (`SimpleMachine`) along with the appropriate allocator (`SimpleAllocator`).

To integrate the scheduler with the rest of SST, there is a class to represent the component as a whole and to handle the interface between the scheduling and allocation objects and the other components of SST. This interface includes using the XML to initialize the component and receiving job arrival events as SST messages. Starting jobs and learning of their completion involve interacting with *node* components, one for each node in the system. The Allocator classes notify the interfacing class, which sends the appropriate nodes an SST message to begin processing. As each node completes its part of a job, it sends an SST message back to the interfacing class, which gathers these until all nodes assigned to the job have completed, at which point the rest of the scheduler is told of these nodes' availability. Returning the nodes to availability all together mimics the behavior of real systems.

5.3 Allocation Algorithm Implementation in SST

We implement the joint optimization algorithm in SST in order to evaluate our allocation strategy and compare with the existing performance-aware and cooling-aware job allocation methods.

The main idea of our joint optimization policy is to achieve both cooling awareness and performance awareness while allocating jobs on HPC data centers. Since the performance-awareness has been handled by the scheduler component in SST as discussed in Section 5.2, we implement the joint optimization algorithm by adding the solver for the cooling-aware optimization algorithm. We also incorporate an interface into SST for the scheduler and cooling-aware optimization solver to call each other interactively. The cooling-aware optimization algorithm (MPIT) is implemented by using the GLPK solver to find the allocation decision that provides the lowest peak inlet temperature for our target data center.

Once a job arrives, the scheduler component first calls the cooling-aware optimization solver to check on which nodes the cooling-aware policy would allocate the job. These nodes are treated as *possible shell centers* as described in Section 4.3. The scheduler then feeds the locations of these *possible shell centers* to the performance-aware algorithm (e.g., MC1x1) to select possible allocations that provide the lowest communication-cost. If multiple allocation choices are provided, the scheduler will calculate the cooling power consumption resulted by each allocation decision and select the one with the lowest cooling temperature. The entire process will repeat for each job in the job queue until the workload is finished.

6 Experimental Results

In this section, we present the experimental results for our joint optimization algorithm and the existing job allocation algorithms. We first compare the performance and cooling power between performance-aware and cooling-aware job allocation algorithms using traces from real-world parallel workloads archive [22]. We then present the performance and cooling power evaluation results for our joint optimization algorithm, and compare the results against performance-aware and cooling-aware job allocation policies. All of our experiments are conducted in SST to accurately mimic the performance and energy behavior of HPC data centers.

Figure 6 presents the running time and cooling power consumption resulting from using the existing job allocation algorithms. We use $DAS-FS4$ traces from the parallel workloads archive as our workload, which include 32953 jobs in total. The number of processors required by jobs from DAS traces ranges between 1 and 40. The bars in Figure 6 represents the average running time and cooling power for all the jobs in the DAS trace. We observe that the cooling power consumption varies significantly across different job allocation algorithms. Among the five existing job allocation algorithms, cooling-aware job allocation results in the lowest cooling power consumption. Across all the 32953 jobs in DAS traces, MC1x1 job allocation algorithm causes 14.02% and 9.27% more cooling power on average, in comparison to the cooling-aware algorithm and random allocation respectively. On the other hand, MC1x1 provides better performance than random and cooling-aware allocations by reducing the communication cost between different threads. In comparison to the cooling-aware algorithm, MC1x1 improves performance by 3.16% on average, while it reduces job running time by an average of 6.78% compared to the random job allocation.

We conduct the same experiments using a different workload from parallel workloads archive: $NASA$, which includes 18239 jobs. As shown in Figure 7, we observe similar trends as in the results of DAS traces. MC1x1 job allocation algorithm, in this case, causes 4.56% more cooling power consumption in comparison to cooling-aware algorithm, and 3.20% more cooling power than random allocation. The detailed simulation results for both $NASA$ and DAS traces are listed in Table 1. From Table 1, we observe that cooling-aware job allocation algorithm provides much lower cooling power consumption than the other job allocation algorithms. For example, for $NASA$ traces, Genalg job allocation algorithm (i.e., the center-based algorithm of Krumke et al. [39]) results in 3.81% more cooling power consumption than cooling-aware allocation. On the other hand, the cooling-aware algorithm in general results in poor application performance compared to the other policies. For example, for DAS traces, the resulting job running time when using cooling-aware job allocation algorithm is about 3.14% longer than using Genalg algorithm. Such delays are not desirable in HPC data centers.

In order to better understand the thermal impact of job allocation algorithms in HPC data centers, we select 100 jobs from $NASA$ traces (job number 101 to 200) and illustrate the cooling power traces of the target data center running these jobs in Figure 8. We observe a significant difference between the cooling powers resulting from different allocation algorithms. For example, for the first seven jobs in the selected job queue, using MC1x1 job allocation algorithm (which only considers minimizing communication time), the data center spends about 80KW on cooling power while it only spends around 20KW by using cooling-aware job allocation algorithm. These results show that, solely considering performance is not sufficient in making energy-efficient job allocation decisions. It is important to have a joint optimization algorithm to consider cooling energy and performance at the same time when allocating jobs in data centers.

Motivated by the potential inefficiencies of solely performance or cooling-aware job allocation algorithms, our joint optimization algorithm considers performance and cooling energy consumption at the same time. In order to evaluate the running time and cooling power of our algorithm, we conduct experiments in SST using both $NASA$ and $DAS-FS4$ traces from parallel workloads archive, and compare the results against MC1x1 and cooling-aware allocation algorithms. The number of processors required by jobs from $NASA$ and DAS traces ranges between 1 and 40,

Figure 9 shows the total running time of $NASA$ and DAS traces when using our joint optimization algorithm compared with the MC1x1 and cooling-aware allocation algorithms. We see that our joint optimization algorithm provides comparable performance benefits as the performance-aware allocation policy. For $NASA$ traces,

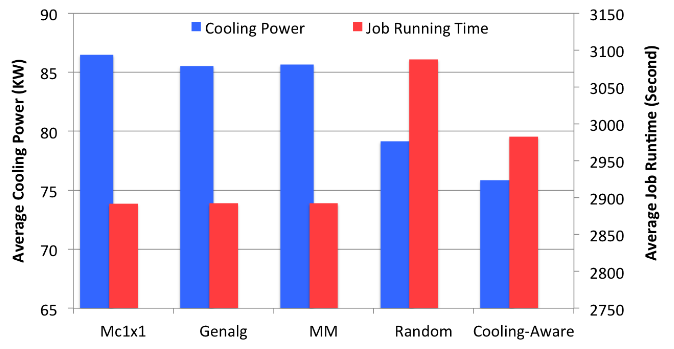

Fig. 6: Performance and cooling power comparisons between different job allocation algorithms for $DAS-FS4$ traces from parallel workloads archive.

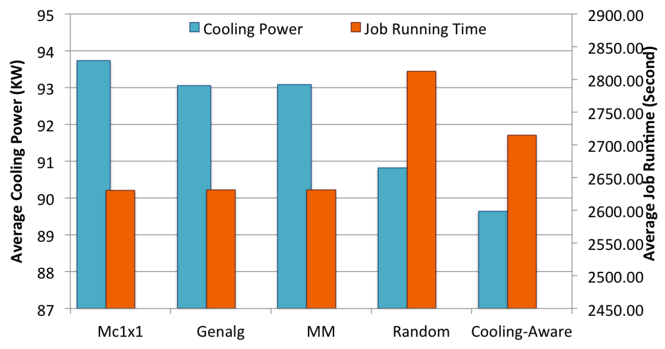

Fig. 7: Performance and cooling power comparisons between different job allocation algorithms for NASA traces from the parallel workloads archive.

TABLE 1: Performance and cooling power comparison results between different job allocation algorithms.

Allocation Algorithms		MC1x1	Genalg	MM	Random	Cooling-Aware
NASA	Cooling Power (KW)	93.74	93.06	93.08	90.83	89.64
	Runtime (Second)	2630.53	2631.08	2631.58	2812.50	2714.71
DAS	Cooling Power (KW)	86.48	85.53	85.65	79.14	75.85
	Runtime (Second)	2891.44	2892.03	2892.43	3087.34	2982.78

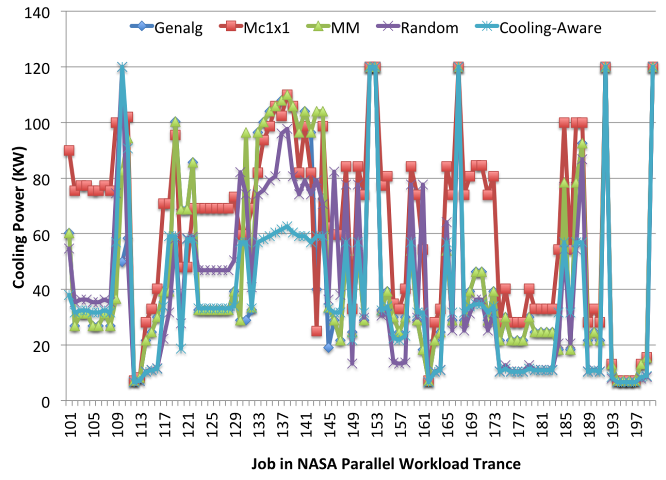

Fig. 8: Data center cooling power traces for job 101 to job 200 from $NASA$ traces with different job allocation algorithms.

our joint optimization algorithm saves job running time by 2.93% on average in comparison to cooling-aware optimization algorithm. For DAS traces, our algorithm improves performance by 2.92% on average compared to the cooling-aware algorithm.

We compare the resulting cooling power of $NASA$ and DAS traces using our joint optimization algorithm against the MC1x1 performance-aware job allocation algorithm and cooling-aware algorithm in Figure 10. We see that, for the target data center under 100% (i.e., all nodes are occupied, e.g., the first five jobs in the job queue) and 2.5% (i.e., only one node is occupied, e.g., the sixth job in the job queue) utilization, the algorithms give allocations resulting in the same cooling power. Across all the jobs of DAS traces, our joint optimization policy results in up to 42.21% less cooling power consumption in comparison to MC1x1 algorithm. Across all the jobs in $NASA$ traces, our joint optimization policy achieves 39.02% lower cooling power consumption than MC1x1, while retaining similar system performance.

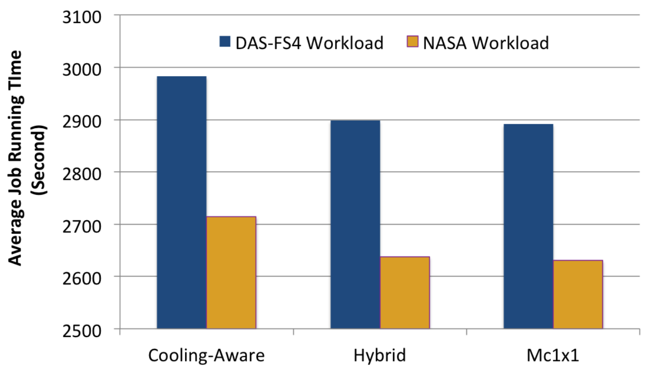

Fig. 9: Runtime comparisons between the 3 algorithms for $NASA$ and DAS traces from parallel workloads archive.

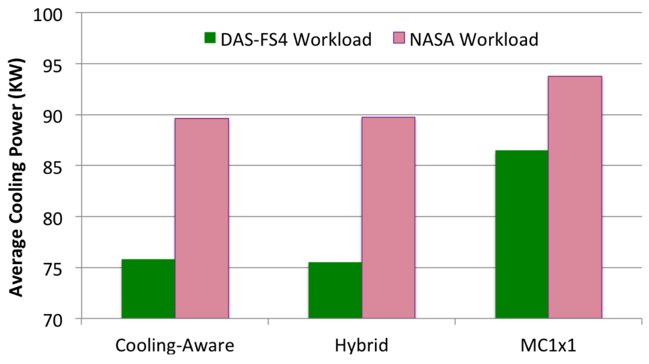

Fig. 10: Cooling power comparisons for the 3 algorithms for $NASA$ and DAS traces from the parallel workloads archive.

7 Conclusion

In this paper, we have proposed a joint job allocation policy to optimize both cooling power and communication latency in HPC data centers. Our policy first uses the MPIT algorithm to find the most cooling-efficient nodes to allocate a job. It then applies the modified MC1X1 algorithm to allocate the job on cooling-efficient nodes while keeping the average L_1 distance at a minimum. We have implemented the data center cooling energy model and our joint optimization algorithm in SST, a simulator for large data centers. Utilizing the simulation framework, we have compared the performance and cooling energy consumption of existing job allocation algorithms. We have also evaluated our joint policy under real-world workloads from parallel workload archive and observed that, for HPC applications with 30% communication, our policy decreases the cooling power by up to 42.21% in comparison performance-aware policies. At the same time, we achieve comparable performance to that of the performance-aware policy.

Acknowledgement

This work has been partially funded by Sandia National Laboratories.

References

[1] V. Leung, E. Arkin, M. Bender, D. Bunde, J. Johnston, A. Lal, J. Mitchell, C. Phillips, and S. Seiden, "Processor allocation on Cplant: achieving general processor locality using one-dimensional allocation strategies," in *IEEE International Conference on Cluster Computing*, pp. 296–304, 2002.

[2] J. Mache, V. Lo, and K. Windisch, "Minimizing message-passing contention in fragmentation-free processor allocation," in *International Conference on Parallel and Distributed Computing Systems*, pp. 120–124, 1997.

[3] M. A. Bender, D. P. Bunde, E. D. Demaine, S. P. Fekete, V. J. Leung, H. Meijer, and C. A. Phillips, "Communication-aware processor allocation for supercomputers: Finding point sets of small average distance," *Algorithmica*, vol. 50, pp. 279–298, Jan. 2008.

[4] S. Bhattacharya and W.-T. Tsai, "Lookahead processor allocation in mesh-connected massively parallel multicomputer," in *International Parallel Processing Symposium*, pp. 868 –875, apr 1994.

[5] P. Walker, D. Bunde, and V. Leung, "Faster high-quality processor allocation," in *Proceedings of the 11th LCI International Conference on High-Performance Clustered Computing*, 2010.

[6] V. Subramani, R. Kettimuthu, S. Srinivasan, J. Johnston, and P. Sadayappan, "Selective buddy allocation for scheduling parallel jobs on clusters," in *IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 107–, 2002.

[7] A. Sansottera and P. Cremonesi, "Cooling-aware workload placement with performance constraints," *Performance Evaluation*, vol. 68, pp. 1232–1246, nov 2011.

[8] J. Koomey, "Growth in data center electricity use 2005 to 2010." http://www.analyticspress.com/datacenters.html, August 1 2011.

[9] "UC San Diego's greenlight project to improve energy efficiency of computing." http://ucsdnews.ucsd.edu/newsrel/science/07-08GreenLightProj.asp.

[10] R. Sawyer, "Calculating total power requirements for data centers," *White Paper, American Power Conversion*, 2004.

[11] J. Rajic, "Evolving toward the green data center." http://http://stack.nil.si, 2009.

[12] D. J. Brown and C. Reams, "Toward energy-efficient computing," *Communications of the ACM*, vol. 53, pp. 50–58, March 2010.

[13] Moore *et al.*, "Making scheduling "cool": temperature-aware workload placement in data centers," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, pp. 5–5, 2005.

[14] 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 Trans. Parallel Distrib. Syst.*, vol. 19, pp. 1458–1472, Nov. 2008.

[15] E. Pakbaznia and M. Pedram, "Minimizing data center cooling and server power costs," in *International Symposium on Low Power Electronics and Design*, pp. 145–150, 2009.

[16] R. Ayoub, S. Sharifi, and T. S. Rosing, "Gentlecool: cooling aware proactive workload scheduling in multi-machine systems," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pp. 295–298, 2010.

[17] T. Heath *et al.*, "Mercury and freon: temperature emulation and management for server systems," in *ASPLOS*, pp. 106–116, 2006.

[18] F. Kaplan, J. Meng, and A. K. Coskun, "Optimizing communication and cooling costs in hpc data centers via intelligent job allocation," in *Proceedings of the International Green Computing Conference (IGCC)*, 2013.

[19] P.-J. Chuang and N.-F. Tzeng, "An efficient submesh allocation strategy for mesh computer systems," in *IEEE International Conference on Distributed Computing Systems*, pp. 256–263, May 1991.

[20] J. Kim, M. Ruggiero, and D. Atienza, "Free cooling-aware dynamic power management for green datacenters," in *International Conference on High Performance Computing and Simulation (HPCS)*, pp. 140 –146, july 2012.

[21] L. Wang, G. von Laszewski, J. Dayal, X. He, A. Younge, and T. Furlani, "Towards thermal aware workload scheduling in a data center," in *International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN)*, pp. 116 –122, dec. 2009.

[22] "Parallel workloads archive: raw workload logs from various machines around the world." http://www.cs.huji.ac.il/labs/parallel/workload/.

[23] S. Kumar, Y. Sabharwal, R. Garg, and P. Heidelberger, "Optimization of all-to-all communication on the blue gene/l supercomputer," in *International Conference on Parallel Processing*, pp. 320 –329, 2008.

[24] R. Barrett, C. Vaughan, and M. Heroux, "MiniGhost: A miniapp for exploring boundary exchange strategies using stencil computations scientific parallel computing," Tech. Rep. SAND2011-5294832, Sandia National Laboratories, 2011.

[25] E. Hertel, R. Bell, M. Elrick, A. Farnsworth, G. Kerley, J. McGlaun, S. Petney, S. Silling, P. Taylor, and L. Yarrington, "CTH: A software family for multi-dimensional shock physics analysis," in *Proc. 19th International Symposium on Shock Waves*, 1993.

[26] L. A. N. Laboratory, "High-performance computing: Cielo supercomputer." http://www.lanl.gov/orgs/hps/cielo/index.html.

[27] "Top 500 list - November 2013." http://www.top500.org/lists/2013/11/.

[28] C. Lively *et al.*, "Energy and performance characteristics of different parallel implementations of scientific applications on multicore systems," *J. High Perform. Comput. Appl.*, vol. 25, Aug 2011.

[29] M. Sayeed, H. Bae, Y. Zheng, B. Armstrong, R. Eigenmann, and F. Saied, "Measuring high-performance computing with real applications," *Computing in Science and Engg.*, vol. 10, pp. 60–70, July 2008.

[30] M. Crovella, R. Bianchini, T. Leblanc, E. Markatos, and R. Wisniewski, "Using communication-to-computation ratio in parallel program design and performance prediction," in *IEEE Symposium on Parallel and Distributed Processing*, pp. 238 –245, dec 1992.

[31] Q. Tang, T. Mukherjee, S. Gupta, and P. Cayton, "Sensor-based fast thermal evaluation model for energy efficient high-performance datacenters," in *International Conference on Intelligent Sensing and Information Processing, ICISIP.*, pp. 203 –208, 15 2006-dec. 18 2006.

[32] http://www.gnu.org/software/glpk/.

[33] A. Rodrigues, K. Bergman, D. Bunde, E. Cooper-Balis, K. Ferreira, K. Hemmert, B. Barrett, C. Versaggi, R. Hendry, B. Jacob, H. Kim, V. Leung, M. Levenhagen, M. Rasquinha, R. Riesen, P. Rosenfeld, M. del Carmen Ruiz Varela, and S. Yalamanchili, "Improvements to the structural simulation toolkit," in *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, (ICST, Brussels, Belgium, Belgium), pp. 190–195, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2012.

[34] N. L. Binkert *et al.*, "The M5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, 2006.

[35] M. Hsieh, J. Meng, M. Levenhagen, K. Pedretti, and A. K. Coskun, "Sst + gem5 = a scalable simulation infrastructure for high performance computing," in *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, pp. 648–655, 2012.

[36] D. A. Lifka, "The anl/ibm sp scheduling system," in *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 295–303, 1995.

[37] A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 6, pp. 529–543, 2001.

[38] V. Lo, K. J. Windisch, W. Liu, and B. Nitzberg, "Noncontiguous processor allocation algorithms for mesh-connected multicomputers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 7, pp. 712–726, 1997.

[39] S. Krumke, M. Marathe, H. Noltemeier, V. Radhakrishnan, S. S. Ravi, and D. J. Rosenkrantz, "Compact location problems," *Theoretical Computer Science*, vol. 181(2), pp. 379–404, 1997.