Topology-aware Scheduling on Blue Waters with Proactive Queue Scanning and Migration-based Job Placement

Kangkang Li[1], Maciej Malawski[1,2], and Jarek Nabrzyski[1]

[1] University of Notre Dame, Center for Research Computing, IN, USA
[2] AGH University of Science and Technology, Department of Computer Science, Krakow, Poland {kli3,mmalawsk,naber}@nd.edu

Abstract. Modern HPC systems, such as Blue Waters, has a multidimensional torus topology, which makes it hard to achieve both high system utilization and scheduling speed. The low scheduling speed comes from the inefficient resource allocation scheme by using a pre-defined Shape Table, which is highly time consuming. The low system utilization is majorly caused by system fragmentation and node drainage. System fragmentation includes both internal fragmentation due to convex prism shape requirement, and external fragmentation resulted from contiguous allocation strategy. The node drainage is caused by lacking of free contiguous space for allocation upon large job's schedule. System drainage is due to the insufficient free contiguous space to schedule the incoming large jobs. In this paper, with the objective of resource utilization and scheduling speed improvement, we address the topology-aware scheduling problem on Blue Waters. To improve the scheduling speed, we propose an efficient free partition detection algorithm. To overcome the system drainage by large job, we propose to use proactive queue scanning to search for large job and gain a buffer period to prepare the system for acceptance. With the set of jobs in the scan window and a set of free partition in the system, we can transform the scheduling into off-line placement, which can be modeled as multiple knapsack problem. We design a migration-based job placement heuristic to make space for large job and improve the system utilization. Through extensive simulations of modeled trace data, we demonstrate that our approach works well in terms of improving the system utilization and scheduling speed.

Keywords: Topology-aware scheduling, multiple-knapsack, free partition detection, migration-based, job placement

1 Introduction

Many high performance systems have multidimensional torus topologies, for example Cray XE/XK has a 3D torus communication network [1], BlueGene/Q 5D [2], and the K computer a 6D torus [3] topology. However, this large-scale network also makes it hard to achieve both high system utilization and scheduling speed. For example, on Blue Waters, jobs are allocated one by one at a

fixed rate, which is one *schedule cycle*. A pre-defined Shape Table is adopted to allocate the required nodes for a job. In order to place the job, the scheduler has to exhaustively search the entire system to find the free space with a matching shape, which leads to a high time complexity and low scheduling speed. To optimize the application performance, the job shape has a form of a convex prism, which guarantees that two jobs running in parallel do not share the communication network and thus do not interfere with each other.

The low system utilization on Blue Waters is majorly caused by two factors: system fragmentation and node drainage. System fragmentation includes both internal and external fragmentation. The *internal fragmentation* results from the the convex prism shape allocation that allocates more nodes a job than it requests. The *external fragmentation* is caused by contiguous job shape allocation strategy that makes the free system resources separated into smaller non-contiguous blocks and interspersed by allocated resources. This will lead to the situation when sufficient number of free nodes cannot be contiguously allocated for a job.

System drainage is caused by the incoming *large jobs*, i.e. more than 1024 nodes, as typically referred to by system administrators. To accept a large job, the system has to drain (release nodes) until sufficient space is available for allocation. This drainage brings the system utilization down for a long period. There are two possible scenarios of system drainage:

- The first is that, the total free resources in the system are sufficient to accept the large job, however, due to the system fragmentation, the scheduler is not able to find a free contiguous space for the job.
- The second scenario is that, the remaining total free resources are not sufficient.

Since the impact of large jobs on system utilization is considerable, in this paper, we will focus on addressing the problem of efficient placement of large jobs in these two scenarios.

For the first scenario, suppose upon the large job schedule cycle, there is a sufficient contiguous block for allocation, there will be no obstacle to immediately place the large job and achieve a high utilization. This gives us the intuition that, if the system is *proactively prepared* with enough resources before the large job arrives at the head of the queue, we are able to avoid the system drainage and schedule the large job.

This *proactive preparation* of the system requires two measures to be taken. The first is to scan deep into the queue to *proactively* search for a large job. This will gain us a buffer period to make preparations before the large job arrival. Once a large job is detected, the second is to make preparations by implementing a job placement reconfiguration in the system, which allows us to make space to place the large job.

We will reconfigure the job placement in the system once a large job is detected at the tail of scan window. As all the running jobs are allocated with convex shape, we can consider them as a set of "allocated" bins for re-allocation.

Combing the set of free bins of partitions and the set of "allocated" bins, we try to place the jobs in this scan window including the large job. This can be transformed as a multiple knapsack problem, which is NP-hard. We propose a migration-based job placement algorithm to give a heuristic solution.

For the second scenario, we will adopt backfilling to maintain the system utilization once the start time of the large job is determined, which requires three steps to be taken. Firstly, in the time period before the large job, we perform proactive queue scanning and placing jobs in the scan window as normal. If there are new large jobs detected in this period, we will mark the first large job as the pause job point for backfilling, where we do not select the jobs ranked after this pause point as the candidate jobs for backfilling. Secondly, upon the schedule of large job, we determine its start time based on the computing of current system status. Thirdly, starting from the job immediate after the large job, we place it into the system as normal if it is a qualified candidate for backfilling. A qualified backfilling candidate is a job ranked before the pause point and has less walltime than the start time. Furthermore, in order to efficiently implement the migration and backfilling, we need to improve the scheduling speed. This requires us to design an efficient free partition detection algorithm for job placement.

The paper is organized as follows: the problem statement is presented in Section 2. In Section 3, we propose a scheduling scheme with queue scanning approach. In Section 4, we present the multiple knapsack model for job placement problem. Section 5 discusses our migration-based job placement on Blue Waters. In Section 6, we conduct simulations to validate the efficiency of our approach. The related work is discussed in Section 7 and we give our conclusions and future work in Section 8.

2 Problem Statement

The topology-aware job scheduling problem for the 3D torus HPC system is formulated as follows. The system consists of X, Y, Z dimensions with torus interconnect. E.g. in the case of Blue Waters, each dimension has 24 Gemini routers, making the system 24*24*24 torus interconnect topological structure. Every Gemini router contains two compute nodes. The links in the X and Z directions are two times faster than in the Y direction, leading to the the asymmetric communication speed in the system.

The scheduler is in charge of job prioritizing and resource allocation. All the jobs are assigned a priority rank and are waiting in the queue to be allocated into the system. At each schedule cycle, the processes of each job are placed into a subset of computing nodes of the system. Each job requires a convex prism shape to ensure the high performance by avoiding communication interference.

The objective is to design an efficient job scheduling strategy to achieve a high resource utilization of the entire system, namely the percentage of running nodes in the system. As we apply migration to reconfigure the job placement, we need also design a fast free space detection algorithm for an efficient implementation of our approach.

Algorithm 1 Proactive Queue Scanning

1: **Input:** the scanning window size D, ranked waiting queue Q, current head job priority rank r
2: **Output:** $J_r^D = \{J_r, J_{r+1}, ..., J_{r+D}\}$
3: **while** Q is not empty **do**
4: scan the queue with window size D
5: obtain the job set J_r^D at schedule cycle with head job rank r and scan depth D
6: $r = r + 1$

3 Scheduling with Proactive Queue Scanning

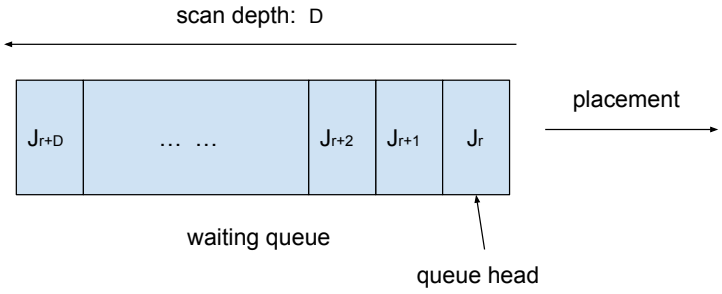

Fig. 1. Proactive Queue Scanning

In this section, we discuss the method of proactive queue scanning to search for large jobs and obtain a set of jobs for placement into the system, which can lead to a better resource allocation.

The scheduler maintains a scan window to proactively scan the queue at *each schedule cycle*. The size of the scan window is the depth of scanning from the head of the queue. This queue scanning produces a set of ranked jobs in the scan window, and we assume each job's walltime is known a priori. We summarize this process in Algorithm 1.

As described in Algorithm 1, starting from current head job ranking r, we scan the queue with depth D, and get the set J_r^D of D jobs. Once a large job is detected at the tail of the scan window, we will try to reconfigure the job placement for accepting the large job, which we will describe in details in Section V. If there is not large job detected, we just place the job r into the system, and queue head will be the job ranked $r = r + 1$ for next cycle schedule and scan.

4 Multiple Knapsack Model

The queue scanning produces a ranked set of jobs, which will be scheduled into the contiguous free space of the system. The system is fragmented and has

multiple partitions of contiguous rectangular free space. These partitions can be represented as a set of rectangular bins. These bins show, at the current placement cycle, the set of free partitions in the system.

Similar to the model used in [6], each bin can be considered as a knapsack and the jobs are the items waiting to be put into the knapsacks. Let $J_r^D = \{j_1, j_2, ..., j_D\}$ be a set of D jobs in current scan window with head job ranked r. Each job j_i has weight w_i, with profit p_i. The physical meaning of both weight w_i is the job size and p_i is determined by specific scenario. Let $K = \{K_1, K_2, K_3, ..., K_m\}$ be a set of m knapsacks. Each knapsack K_j with the initial capacity of C_j. We want to find a placement for the D jobs into the bins to maximize the total profit of them. The mathematical formulation is below:

$$Max : \sum_{i=1}^{m} \sum_{j=1}^{D} x_{ij} p_i \tag{1}$$

$$Subject\ to : \sum_{j=1}^{D} x_{ij} \leq 1, \quad \forall i = 1, 2, ..., m \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} w_i \leq C_j, \quad \forall j = 1, 2, ..., D \tag{3}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i = 1, 2, ..., m, \forall j = 1, 2, ..., D \tag{4}$$

$$C_j \geq 0, \quad \forall j = 1, 2, ..., D \tag{5}$$

where $x_{ij} = 1$ means job i is put into knapsack j, and $x_{ij} = 0$ means job i is not put into knapsack j. The remaining capacity of each knapsack can never be negative, but it will be reduced as more jobs are put into this knapsack.

Based on this multiple knapsack model, maximizing the system utilization can be transformed into maximizing the objective of Eq. 1, which is NP-hard and requires a heuristic solution, which we will describe in the following section.

5 Migration-based Job Placement

In this section, we propose a migration-based job placement scheme to solve the multiple knapsack problem. Firstly, we design an efficient free partition detection algorithm to find all the bins. Secondly, we present our migration-based job placement to solve the multiple knapsack problem. Thirdly, to deal with large jobs, we combine migration and backfilling to produce more opportunities to higher system utilization.

5.1 Free Partition Detection

Blue Waters currently uses a pre-defined Shape Table to allocate the required nodes for a job. In order to place the job, the scheduler has to exhaustively search

Algorithm 2 Direct Placement

```
1: Input: Bins ordered with decreasing preferences; incoming job
2: for all bins ordered by preferences do
3:    if remaining capacity ≥ incoming job size then
4:       place the job into this bin
5:    else
6:       try to place the job into next-highest priority bin
```

the entire system to find the free space with a matching shape, which is inefficient and leads to a high time cost.

Based on the multiple knapsack model, we need to find all the free rectangular contiguous partitions in the system for job placement. Therefore, we propose an efficient free partition detection algorithm for fast free space search.

The system is sliced into layers along the Y dimensions, as illustrated in Fig. 3. Along the Y dimension, using logarithmic scale, we can divide all the XZ layers by the power of 2. Using divide and conquer idea, we can see this system as a binary tree, where the leaf at the bottom is each XZ layer.

Each $2D$ layer needs $O(M^2)$ to get the largest free rectangular block. Therefore, it needs $M * O(M^2)$ to combine the matrices of every two single layer and calculate a free 2-layer rectangular block, which corresponds to the layer of height of 2 in the tree.

After that, going up along with tree, each 4-layer free partition need $\frac{M}{2} * O(M^2)$ to calculate a 4-layer rectangular block, which which corresponds to the layer of height of 4 in the tree. Through going to the tree root, we need the total is $(1+2+4+\ldots 2^{log(M)}) * O(M^2)$ to scan the whole system and get the largest rectangular block. It takes $2M * O(M^2)$, which is $O(M^3)$ and very efficient.

Once the free bins of partitions are detected, we will place the incoming job into one of these bins. However, there is a preference difference in placing into different bins. The extent of internal fragmentation (the number of idle nodes because of convex shape) is different if we place the jobs in different bins. We want to place each job into the best bin to minimize the internal fragmentation, and thus improve system utilization. Fig. 2 gives a 2D example.

As shown in Fig. 2, for this incoming job J_1 with 6 nodes and convex shape requirement, we should place it in the right bin instead of the left. Since the left bin will bring 4 idle nodes wasted if convex shape is required. However, the right bin brings no internal fragmentation. For 3D system, if the large occupy more than one XZ plane, we we divide the job number by the size of XZ plane to get the number of layers it will be allocated in the bin. In that case, we will know the number of idle nodes of the allocation if job is placed in this bin.

Based on the preference ranking of these bins, we will place the incoming job into the bin from the highest rank bin. If that bin cannot accept the job, we will continue trying the next priority bin until the job is allocated.

5.2 Migration-based Job Placement

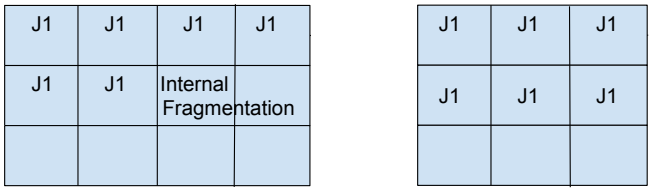

Fig. 2. Illustration of best bin for job placement

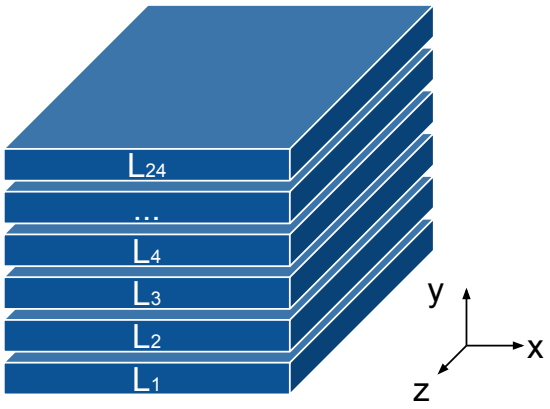

Fig. 3. Partitioning of the system into layers by Y axis

Due to the constraint of job priority, we have to allocate the jobs from the waiting queue one by one according to their ranking. Even if we greedily choose the best bin for placing each job, globally, it could not be the best allocation for the whole placement in this scan window. For instance, if the best bin one of the incoming job is full, we can directly place this job to the next-best bin with enough resource as our direct placement algorithm above. However, if we migrate one job from the best bin, and place it into another bin, we can save space for accepting this incoming job, as shown in Fig.4.

The key for implementing this migration is to find a qualified victim job to migrate, which is not always reachable. As each job has its own profit of placing into one bin. Migration a job will lead to a cost of profit decrease for this victim. Therefore, this migration process needs to have constraint. The migration constraint is that, the total profit gained through migration must be larger than

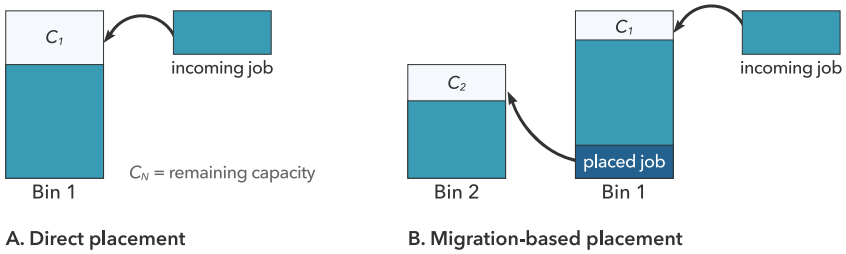

Fig. 4. An illustration of the job placement. When using direct (greedy) placement, the incoming job is assigned to bin 1 with the enough remaining capacity C_1. When using migration, we can find an already placed job ("a victim") to be migrated to another available bin with enough capacity, such as bin 2 in the picture above

Algorithm 3 Migration-based Job Placement

Input: a set of bins ranked from 1 to K, the incoming job
Output: a schedule of incoming job on the set of ranked bins
for each bin_i ranked from 1 to K **do**
  for each job_j on bin_i with rank i **do**
    if migration of job_j make sufficient space for placing incoming job **then**
      try to find another host for job_j
      if job_j can be replaced into another host **then**
        if migration constraint is satisfied **then**
          mark job_j as a qualfied victim
  if more than one qualified victim exists **then**
    select the best victim with the lowest migration constraint
    proceed migration-based job placement

that without migration process. The profit will have different meaning under different scenarios, and we will present one case in Section V.

In sum, There are three qualifications that such a qualified victim job must meet:

1. it can save enough resources for accepting the incoming job
2. it can find a new available host with enough remaining capacity to accept the victim job itself, and
3. the migration constraint must be satisfied.

If such a qualified victim job is found, then this migration process is viable. If more than one qualified victim jobs exists in that bin, we choose the victim job with the minimal idle nodes increase. However, there exists special circumstance for a particular incoming job:

– We cannot find a qualified victim job in the best bin. In that case, we try the next-best bin. If the incoming job still cannot be placed, we continue to

try the next-next-best bin. This search loop goes on until the incoming job is placed, or all the bins are tried.

With M jobs to be placed and N bins available in the system, assuming the average number of placed jobs on a bin is K, then, the time complexity of Algorithm 2 is $O(KN)$, which is no more than $O(M)$. In sum, the migration-based job placement costs only $O(M^2)$, which is very efficient.

6 Backfilling with Migration-based Job Placement

During the queue scanning, once a large job is detected and there is not sufficient space to schedule it in any of the free partitions, we need to drain the system and we will use backfilling to maintain high system utilization at the same time. However, before considering backfilling, we can try to reconfigure the job placement in the system if the total resources in the system are sufficient to accept the large job.

As jobs are scheduled at a fixed rate, based on prediction and system scanning, we can maintain a list of future running jobs in the system upon the the schedule of large job. This list of running jobs also can be seen as a set of "pre-allocated" bins for re-placement. Combining this set of pre-allocated bins and the free partition of bins, we can transform the placement to a multiple knapsack problem. The input is a set of running jobs and the set of jobs in the scan window. We can still use our migration-based heuristic to solve it and the objective of is to minimize the total migration cost, which can also regarded as the maximizing the reciprocal of migration cost.

Migration has cost of profit decrease of the victim job. The physical meaning of migration cost is the time delay for migration. To implement the migration, the job needs to checkpointed, transferred and re-started on different set of resources. Considering a shorter job which almost finish and a longer job which just starts, the former leads to more migration cost as the amount of work it requires to transfer is larger. Furthermore, a larger job leads to more migration cost than the smaller one, since the extra nodes bring in more amount of data for transmission. Therefore, We define migration cost of each job as $size * walltime * completion\ ratio$. In that case, a 24 hour walltime single node job which only runs for 1 hour has less migration cost than that of a 3 hour walltime two-node job which has run 2 hours.

For each job, we assign a priority for each of these bins. For the set of running jobs, the priority value is the reciprocal of migration cost for re-placement. For instance, for an allocated job J_m, its migration cost to place into the position of another allocated job J_n is the sum of the migration cost by these two jobs as they are both relocated. The priority of the bins of free partitions is the reciprocal of its own migration cost. On the other hand, for the set of jobs in scan window, the priority value of each bin is the reciprocal of its own migration cost as well. If a bin cannot accept a job, the bin's priority is 0. After all the bins priority value are determined, we will apply our migration-based job placement to find

a solution. If one viable placement is found, we perform the real migration and rearrange the job placement. If not, we will only allocate the jobs in the scan window before the large job into the system. After that, we use backfilling process to maintain system utilization.

In order to implement the backfilling and determine the start time of the large job, we need three steps.

1) Based on the lists of these running jobs, we sort them increasingly by their remaining completion time.

2) calculate the system status and the largest free partition of the system for each job's completion time starting from the large job schedule to the future, and put them in the timeline

3) Go through the timeline and stop until one sufficient largest free partition is found to to accept the large job, remark that time as the start time of the large job, and reserve that space.

With the start time determined and space reserved, we place the qualified backfilled jobs into the current free partitions on the system. The qualified backfilled jobs are those have walltime time less than the start time of the large job. This placement can be solved using our migration-based placement algorithm as well. However, if we encounter a backfilled job which is also large job and cannot be placed in the system, we will stop the backfilling process and wait for the original large job to start, avoiding redundant time cost on scheduling.

7 Performance Evaluation

In this section, we describe the simulations used to evaluate our approach to improve the system utilization. We use simulated data to show the improvement of scheduling with proactive queue scanning and migration-based job placement reconfigurations. To emulate an non-empty system, we start to record the result data after the system is the first time 80% occupied.

We conduct three groups of simulations to find out the impact of the size of scan window on the performance of our migration-based job placement.

As we can see from Fig. 5, our approach achieves the system utilization on the level of around 90 percent. This shows that, our migration-based job placement can improve the system utilization compared to only implementing backfilling.

Furthermore, as shown in Fig. 6, with difference size of scan window, our approach maintains similar results in terms of system utilization. This shows that, a large size scan window, although enables us to have a long time of planning, but does not necessarily lead to significant benefit to improve system utilization. On the other hand, a large scan window can also bring in time cost on scheduling computing, which is not recommended.

8 Related Work

Recent results of a collaboration between NCSA, Adaptive Computing, and Cray, on topology-aware scheduling on Blue Waters are presented in [1]. The

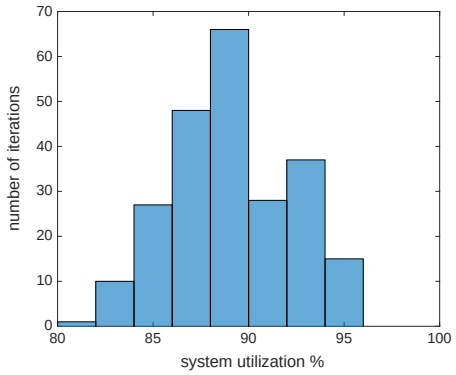

Fig. 5. histogram of our approach with scan window size of 100

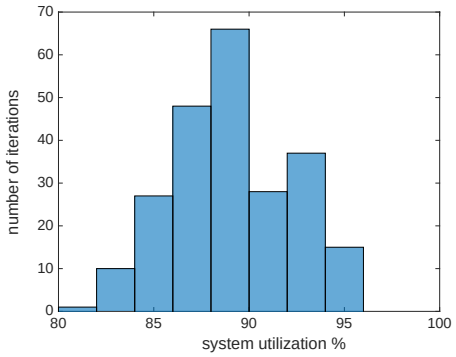

Fig. 6. histogram of our approach with scan window size of 500

first problem reported is runtime variability: e.g. PSDND application runtimes can vary up to 4 times. The improvements of the Moab scheduler in the first implementation enforce a strict policy prohibiting job-to-job interference. This policy allows only node allocations which guarantee that intra-job communication would not be routed over links used by any other job. The allocations thus have the shape of a cuboid or a rectangular prism. The experiments show that the runtime variability is reduced considerably. Another improvement is achieved by integrating Moab with the Topaware task mapping tool developed by Cray. Results for synthetic 3D programs and 4D benchmark application show 2x-4x improvement in performance.

To improve the task placement of applications with 2D, 3D and 4D Cartesian topologies and nearest-neighbor communication, a Topaware tool can be used [4]. There are tools such as Caypat for profiling an MPI application and to detect Cartesian grid communication patterns. This information can be used

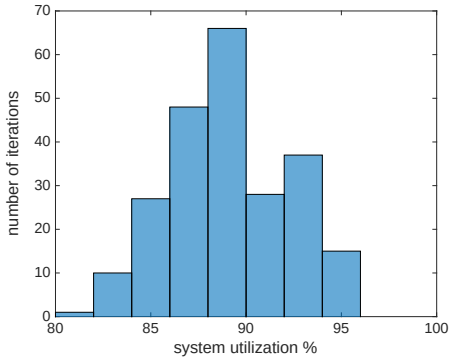

Fig. 7. histogram of our approach with scan window size of 1000

to provide runtime mapping of processes to the computing nodes using MPICH node ordering. The Topaware method requires the user to specify the required number of nodes along each torus dimension and finds the ordering by allocating nodes on subsequent XZ planes, taking into account the gaps resulting from service nodes. For mapping the 2D virtual topology to the 3D torus, a folding method is used. Topaware was evaluated using the WRF, VPIC, S3D and MILC applications.

An overview process of mapping techniques and algorithms for HPC systems is presented in [5]. It discusses algorithmic strategies for topology mapping, such as graph partitioning, mapping enforcement techniques (resource binding and rank reordering), as well as existing solutions and their implementations. This provides a formal definition of the mapping as an optimization problem, and discusses the metrics such as dilation or congestion.

The problem of scheduling jobs on BlueGene/Q system with a 5D torus interconnect is addressed in [2]. BlueGene allows allocating network links exclusively to the selected jobs, which can optimize their performance, but it can leave unused nodes within the system partitions, which leads to lower utilization. The shapes of partitions are the rectangular prisms. The authors use benchmarks to evaluate the application sensitivity to network configuration and propose a communication-aware scheduling policy that takes into account application characteristics. The scheduling policies are evaluated using Qsim, which is a simulator of the Cobalt resource management system used for BlueGene.

In another work related to BlueGene [6], window-based locality-aware job scheduling for torus-connected system is designed to balance job performance with system performance. While the goal is similar to ours, the torus topology is not fully studied in the paper and the scheduler takes all the input jobs as a one-dimensional item, along with the available slots in the torus. However, in our work, we fully investigate torus and job geometric characteristic and propose our topology-aware job mapping strategy.

Furthermore, bin packing [7, 8] is also a related topic. In the bin packing problem, we have a set of bins with fixed capacity limitation and a set of items with given size to be placed into these bins. The objective is to minimize the number of bins used. The offline version is NP-hard as well.

[**?**] presents the analysis and application of scheduling algorithms that augment a baseline first come first serve (FCFS) scheduler. The author presents simulation results for migration and backfilling techniques on BlueGene/L. These techniques are explored individually and jointly to determine their impact on the system. An efficient Projection Of Partitions (POP) algorithm for determining the size of the largest free rectangular partition in a toroidal system is developed. The results demonstrate that migration may be effective for a pure FCFS scheduler but that backfilling produces even more benefits. It is also shown that migration may be combined with backfilling to produce more opportunities to better utilize a parallel machine.

In our work [Singapore] we address the job mapping problem improving the utilization of a single queue HPC system with a multidimensional torus topology, while preserving the performance of jobs and avoiding the communication interference. We propose a simple but efficient resource allocation strategy based on the amount of available space in the system and job packing, which models the job mapping as a multiple knapsack problem. The main idea of our approach is to pack several jobs together from the queue to place into the system. To reduce the internal fragmentation, we propose a zigzag shape allocation method for the communication non-sensitive jobs and a bin selection algorithm for communication sensitive jobs. To reduce the external fragmentation, we propose a bi-directional allocation strategy to separate, in each bin, the spaces between communication sensitive and non-sensitive jobs.

9 Conclusions and Future Work

In this paper, we addressed the problem of improving utilization and scheduling speed of petascale systems with multidimensional torus topologies. To improve the scheduling speed, we propose a free partition detection algorithm. To improve the system utilization caused by large job drainage, we propose to use proactive queue scanning to search for large job and gain a buffer period to prepare the system for acceptance. With the set of jobs in the scan window and a set of free partition in the system, we can transform the scheduling into off-line placement, which can be modeled as multiple knapsack problem. We design a migration-based job placement heuristic to make space for large job and improve the system utilization. The simulations of modeled trace data demonstrate that our approach works well in terms of improving the system utilization. In the future work, we will conduct experiments on real trace data and evaluate the efficiency of our approach.

Bibliography

[1] J. Enos, G. Bauer, R. Brunner, and S. Islam, "Topology-Aware Job Scheduling Strategies for Torus Networks," in *Proceedings of the Cray User Group meeting.*, 2014. [Online]. Available: https://cug.org/proceedings/cug2014$_proceedings/includes/files/pap$182 − $file$2.*pdf*

[2] Z. Zhou, X. Yang, Z. Lan, P. Rich, W. Tang, V. Morozov, and N. Desai, "Improving Batch Scheduling on Blue Gene/Q by Relaxing 5D Torus Network Allocation Constraints," in *IPDPS 2015*, 2015.

[3] Y. Ajima, Y. Takagi, T. Inoue, S. Hiramoto, and T. Shimizu, "The Tofu Interconnect," in *2011 IEEE 19th Annual Symposium on High Performance Interconnects*. IEEE, aug 2011, pp. 87–94. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6041538

[4] R. A. Fiedler and S. Whalen, "Improving task placement for applications with 2D, 3D, and 4D virtual Cartesian topologies on 3D torus networks with service nodes," in *Proc. Cray User's Group*, 2013. [Online]. Available: https://cug.org/proceedings/cug2013$_proceedings/includes/files/pap$148.*pdf*

[5] T. Hoefler, E. Jeannot, and G. Mercier, "An Overview of Process Mapping Techniques and Algorithms in High-Performance Computing," in *High Performance Computing on Complex Environments*, E. Jeannot and J. Zilinskas, Eds. Wiley, jun 2014, pp. 75–94. [Online]. Available: https://hal.inria.fr/hal-00921626

[6] X. Yang, Z. Zhou, W. Tang, X. Zheng, J. Wang, and Z. Lan, "Balancing job performance with system performance via locality-aware scheduling on torus-connected systems," in *2014 IEEE International Conference on Cluster Computing, CLUSTER 2014, Madrid, Spain, September 22-26, 2014*, 2014, pp. 140–148. [Online]. Available: http://dx.doi.org/10.1109/CLUSTER.2014.6968751

[7] D. S. Johnson, "Near-optimal bin-packing algorithms," Ph.D. dissertation, 1973. [Online]. Available: http://opac.inria.fr/record=b1000391

[8] M. R. Garey and D. S. Johnson, "Approximation algorithms for np-hard problems," D. S. Hochbaum, Ed. Boston, MA, USA: PWS Publishing Co., 1997, ch. Approximation Algorithms for Bin Packing: A Survey, pp. 46–93. [Online]. Available: http://dl.acm.org/citation.cfm?id=241938.241940