On the Interconnect Energy Efficiency of High End Computing Systems

Abstract

High performance computing systems are moving towards the exaflops era. The tremendous increase in computational speeed is accompanied by enormous power consumption in these systems. It is necessary to harvest any potential opportunities to save power in these high end computing systems. The goal of this paper is to explore possibilities of power savings in the interconnects between the nodes. By careful scheduling of jobs in a 3D torus-connected cluster of nodes, we show that significant amounts of power can be saved by switching certain portions of the network elements to low power modes. We also present an estimation method that more accurately estimates the actual runtime of jobs from the user provided runtimes and enhances the performance of the scheduling scheme. We validate our results via detailed MATLAB simulations.

1. Introduction

The rising power/thermal issues in ICT equipment and the associated costs have led to intensive research efforts to improve the energy efficiency of data centers and large server farms. With the recent trends in computer architecture promising exaflop systems in the near future, the focus of high performance computing system architects has shifted from just performance maximization to achieving energy efficient computing. A compelling reason behind these research efforts is that with peta and exa–scale computing, the power consumed by these systems is expected to contribute a significant portion of the total cost of ownership of these systems. Power consumption would also severely impact the scalability of the exascale systems. Exascale systems are expected to consume anywhere between $60 - 120$ MW of power [1, 2]. Efficient power management of high performance systems can hence result in significant savings in energy costs. The primary goal of this work is to explore a few possible ways to save energy in a high performance computing environment and to suggest techniques to maximize energy savings.

We distinguish the high performance computing environment from other Internet data centers and cloud service providers. Our focus in this paper is on the high performance applications running on tightly coupled supercomputers. Internet data centers today are generally described as *always on* since the applications hosted in these data centers are expected to be up and running continuously. The demands of these applications are different during different times, and they are expected to be available always. The applications may run in virtual machines and a single physical machine hosts multiple virtual machines. The power demands of these applications vary continuously

depending on their utilization and the power management schemes need to account for the demand side variations. While the demand variations can be predicted to some extent, there can be short term spikes in demands that make power management of these systems difficult.

The nature of jobs running in supercomputers however, is different from those running in Internet data centers. In typical supercomputing environments, the jobs are long running and their behavior is usually predictable. They have fixed runtimes and get completed within the requested/allocated period of time. This is especially true in scenarios where the jobs are run in batches and are allocated groups of dedicated processors. Any communication happens within this allocated group of processors as long as the jobs are independent. Hence spatial and temporal allocation of jobs plays a key role in determining the power consumption of these HPC systems. For instance, if there are idle periods of communication in the interconnects between the processors, they can be exploited by putting these interconnects in low power modes. As a complementary measure, idle periods can be introduced in a few links and sustained for long periods by appropriately allocating jobs to processors.

Power management in tightly coupled HPC systems is a harder problem than other typical data center environments providing cloud computing and web hosting type of services, where the systems are more loosely coupled. One main reason is that in HPC clusters performance is a much more critical factor than in other systems. Adverse impacts on performance are too expensive to tolerate. The aim of this paper is to study and quantify the impacts of spatio–temporal allocation of jobs in HPC environments on the interconnect power consumption of supercomputers–in particular 3D torus connected machines. A 3D torus is a network topology in which each node is connected to six other nodes, two each in the x, y and z dimensions. We investigate potential power savings that can be achieved by ensuring that certain interconnect elements are idle and can be switched to low power modes. As mentioned before, this certainty is achievable in high end computing environments mainly because the jobs are long running and allocation of jobs to processors remains almost static for longer periods of time until the jobs complete. Moreover the networking topology of these machines have more regular structures (like 3D torus) that can be exploited to save power. The target environment of our work is such a tightly coupled system where multiple nodes are running the subtasks of the same job and are interconnected via high speed interconnects and typically located in the same geographical location. Examples are supercomputers like IBM Blue Gene and the Cray machines that have racks of nodes interconnected by high speed network interconnects.

In general, any technique to reduce power consumption directly means switching from a higher performance state to a low performance state. For instance the DRPM technique [3] reduces disk power consumption by dynamically modulating the hard disk rotation speed to achieve significant savings in energy. Dynamic Voltage and Frequency Scaling (DVFS) [4, 5] for processors also trades performance/speed for power. However on an orthogonal note, techniques like workload consolidation during very low utilization periods can result in energy savings with minimum impact on performance. Due to strict performance requirements of HPC applications, a fine grained power management scheme may be infeasible because of the latencies involved while transitioning from low power to high power modes and vice versa. We contend that

coarser grained techniques that operate at a larger time scale can still achieve significant savings in power without impacting performance in supercomputers. Especially in a supercomputer where communication bottleneck is the most significant area of concern, small windows of idle periods in interconnects cannot be exploited without significantly impacting performance. In this regard, we propose a scheduling scheme that minimizes the power consumption of interconnects in the high performance server clusters by careful allocation of processors to jobs. We evaluate the performance of the algorithm formally and via detailed simulations.

The rest of the paper is organized as follows. Section 2 discusses the related research works in the field of power management in high performance computing systems. Section 3 describes the target environment of our scheduling algorithm. Section 4 explains our problem modeling and the working of our scheduling algorithm. Section 5 details our runtime prediction algorithm. We provide our simulation results in Section 6. Finally, Section 7 concludes the paper.

2. Related Work

Kamil et al. [6] explore less invasive approaches to measure power in large scale supercomputers running HPC workloads. The authors compare the effectiveness of multiple power measurement methodologies and investigate the power requirements for various benchmark applications. Sonmez et al. [7] propose job placement policies on a grid to minimize wide area communications in parallel applications. Their test environment is a grid where parallel processes are coscheduled on clusters that are geographically distributed.

Virtualization is increasingly being used in server clusters due to its application isolation capabilities and ease of managemeent. Nagarajan et al. [8] propose a scheme that migrates virtual machines hosting MPI applications from a fault-prone node to a healthy node proactively. Verma et al. [9] investigate the power management in virtualized server clusters hosting HPC applications. They use their experimental results to build a framework for power - aware application placement in virtualized clusters. Their placement strategy takes both CPU usage and working set size into account. Mukherjee et al. [10] propose enhancements to the FCFS and Earliest Deadline First policies to make thermal aware job placement decisions in virtualized data centers.

Tang et al. [11] propose task assignment techniques that minimize the cooling requirements in large scale data enters. The authors formalize the problem of minimizing the cooling requirements as the problem of minimizing the peak inlet temperatures via task assignments. Niyato et al. [12] propose an optimal power management scheme for a server farm to minimize the power consumption while meeting the performance requirements. They formulate the problem as a constrained Markov decision process and then optimize the allocation of tasks to the servers.

Ranganathan et al. [13] propose an ensemble level power management technique for power management in blade servers. They exploit the trends in workloads across multiple systems and manage the power budgets at the enclosure level. This enables more flexibility and results in significant reductions in the power consumption and cooling costs. Moore et al. [14] propose a temperature aware workload placement

technique in datacenters. The goal of the work is to cut down on the cooling costs by placing the workloads in the data center in appropriate temperature zones.

Frachtenberg et al. [15] propose a flexible co-scheduling algorithm to avoid external and internal fragmentation. Their scheme identifies the jobs that require strict co-scheduling and co-schedules them while other jobs are scheduled to increase the utilization of the overall system. Tang et al. [16] propose scheduling schemes that are walltime - aware and attempt to reduce fragmentation of resources due to differences in runtime of neighboring jobs. The scheduling schemes are evaluated on IBM Blue Gene with real time traces from the Argonne National Laboratory [17].

Scheduling on parallel machines has been studied for a long time [18, 19, 20, 21, 22, 23, 24, 25, 26, 27]. The scheme by Qiao et al. [28] is one of the earliest works on scheduling in 3D torus connected systems. They propose a submesh detection scheme considering the size and orientation requirements of the job. Choo et al. [29] propose a scan and search scheduling algorithm for 3D torus connected systems. The authors manipulate the 3D torus information as 2D information. There is a logically contiguous virtual space of processors which is then mapped to the physical space. Aridor et al. [30] investigate architectural solutions to alleviate the fragmentation problem in mesh and torus connected topologies. They claim that a multi–toroidal topology like the IBM Blue Gene/L has better resource allocation and low resource fragmentation.

3. Interconnect Power Management

In this paper, we consider two well known supercomputers that have nodes connected by a 3D torus network–the IBM Blue Gene [31] and the Cray [32] machines. Flat networks like the 3D torus, are highly scalable and the number of network components does not increase super–linearly with an increase in the number of nodes [33]. Hence they are being widely used in large scale systems with thousands of cores. Communication between nodes is increasingly becoming critical in HPC clusters and interconnects play an extremely important role in tightly coupled supercomputers. The interconnects in HPC systems are designed to guarantee high bandwidth between any two nodes. Interconnects connect nodes within the racks and across the racks. The communication phases between nodes (e.g., MPI Send, Receive) can turn out to be bottlenecks if the interconnects cannot guarantee the necessary bandwidth. From the data that we managed to gather, a typical rack in IBM Blue Gene consumed 23 kW of power and the link cards consume around 0.35 kW of power per rack. A typical IBM Blue Gene installation consists of 40 to 64 racks and hence the power consumed by link cards is around 14 to 24 kW which is significant. With introduction of high speed links and fiber interconnects, the power consumption of networking components can become even higher. William J. Camp observes that the power consumed by interconnect elements in exascale computers will be of the order of serveral megawatts [2]. Hence a careful power management of interconnects in supercomputing systems could save significant power. However, to the best of our knowledge there are no research works that have attempted to analyze the potential power savings in network interconnects in supercomputers.

The nodes in IBM Blue gene are divided into partitions and the partitions are allocated to jobs. A single rack consists of two mid–planes with each of the mid–planes

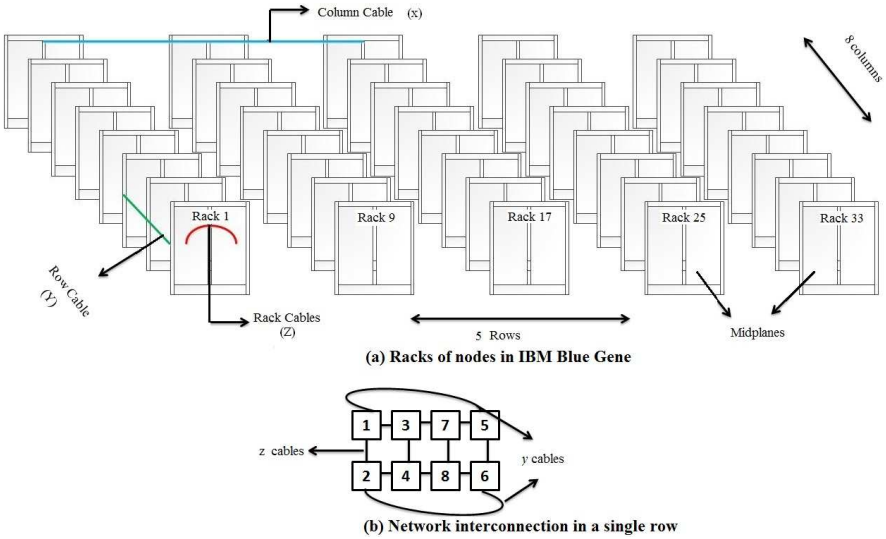

Figure 1: 3D torus network topology in IBM Blue Gene

having 512 nodes connected by a 3D torus network with dimensions 8 x 8 x 16. Multiple racks are connected by cables that maintain the torus structure. Typically the number of racks are between 40 to 64 and the number of nodes per rack is $1,024$. Cray machines have a similar structure but the 3D torus links are shared [34, 35] unlike the IBM machines where the network links are dedicated to partitions. The multi–core processors are connected using the Gemini interconnect ASICs. The Gemini ASICs have 48 ports per chip and provide extremely fast routing capabilities. The number of network interconnects are proportional to the number of nodes and the number of racks.

Figure 1(a) shows a typical IBM blue gene system installation with torus connected nodes organized as multiple rows of racks. A Cray installation would have a similar structure with the racks being as called cabinets and with different number of nodes per cabinet. To maintain consistency we follow the naming convention similar to that of IBM Blue Gene. We refer to the cables connecting different rows of racks as *x*–dimension cables (column cables), the cables connecting the racks within a row as *y*–dimension cables and the cables connecting mid-planes/cages in the same rack as *z*–dimension cables. The *z*–dimension cables also connect mid-planes in the same position on adjacent racks. Generally the *x*–dimension cables that connect different rows of racks are significantly longer than the cables along other dimensions. Hence they are considered expensive in terms of power and communication costs. Figure 1(b) shows the physical links in a row that form the $1 \times 4 \times 4$ slice of the entire $5 \times 4 \times 4$ 3D torus. There are 8 link cards per rack. Each link in Figure 1(b) represents 16 cables 2 each from every link card in the rack. When scheduling the jobs on processors, we deal in terms of allocation units of 512 processors to avoid unnecessary complexity. Even though this assumption might lead to internal fragmentation issues since requested

processors need to be in multiples of 512, it can be easily shown that the scheduling algorithm that we propose is efficient even if the assumption on minimum sizes of allocation units is removed. External fragmentation is also an issue for large jobs, since free processors cannot always be obtained contiguously. We assume that a non–contiguous allocation is done in these cases as in Cray machines.

3.1. Interconnect Power model

The inter–node communication interconnects used in supercomputers like IBM Blue Gene and Cray machines are still mostly copper. The network links provide around 1.4 Gbps unidirectional bandwidth. Other technologies like Infiniband and optic fibers are also increasingly being used for high speed I/O accesses and long distance communication. We assume the conventional model in which the power consumed by the network elements consists of two components–static and dynamic [36]. For instance in the IBM blue gene machines, each rack has link cards that connect the nodes together. There are 8 link cards per rack and each link card has 16 cables - 8 in the X direction and 4 each in the Y and Z directions. Hence the power model for the link card is given by,

$$P_{linkcard} = P_{chassis} + n * P_{link} \tag{1}$$

where n is the number of links that are active.

Hence if the communication along x–dimension is cut down completely, according to our model, the power saved in one link card will be $8 * P_{link}$. The same model can be applied to router ASICs (as in Cray machines) as well. The fewer the number of links in use the lower is the power consumption of the router card.

4. Power aware scheduling

4.1. Scheduling on parallel machines

Given a set of jobs and a set of available processors, a scheduling algorithm maps the jobs to processors. The scheduling algorithm can be optimized based on a number of factors including minimizing response time, minimizing fragmentation, maximizing resource utilization and so on. The goal of our scheduling algorithm is to optimize for minimum power consumption in interconnects. The objective of our algorithm is an optimal spatio–temporal allocation of tasks to nodes so that some of the communication links can be idle for long periods of time.

The following factors influence the design of any scheduling algorithm.

- *Performance impact :* Any power savings in interconnects comes from powering down the network links. Bringing the links back up involves latency. Having to bring back the links from low power modes often will affect the performance considerably. Hence the scheduling algorithm that is being designed has to avoid this.

- *Resource Fragmentation :* Any scheduling algorithm that does not take the topology of the network into consideration could potentially leave the resources fragmented. This not only wastes resources and increases the wait time of jobs but also results in poor energy efficiency. In this paper we consider 3D torus as a representative network topology. Another common topology worth exploring is the fat tree.

- *Resource Utilization :* The scheduling algorithm has to maximize the resource utilization and make sure that no jobs are waiting in the queue if sufficient resources are available.

- *Starvation:* The response time of jobs is a very common metric to evaluate a job placement algorithm. Any job scheduling algorithm has to make sure that the wait time of a job is not more than a certain starvation threshold.

Our scheduling algorithm tries to address all of the above mentioned issues. Power savings is achieved by careful scheduling of tasks to enable powering down of links that are redundant. Sufficient number of links are kept on so that the required connections are maintained between communicating processors. Even though our scheduling scheme does not explicitly consider adaptive routing schemes during hotspots, additional links can always be powered on during periods of congestion with a one time latency of powering them up. The appropriate cost functions in the algorithm can then be modified based on the links that are up and running. However given that jobs are allocated to dedicated processors and the communication is only between those processors, we assume that the hotspot scenarios arise rarely. Hence there is almost no impact on performance. External resource fragmentation is minimized by providing a contiguous allocation of processors to the jobs whenever possible. Only if contiguous allocation is not possible, a non–contiguous allocation is provided. Starvation is prevented by using appropriate strategies in the temporal allocation policy. In the next section we provide an analysis of the optimality of our scheduling algorithm.

4.2. Problem Modeling and Analysis

This section provides a formal analysis of the problem of allocating the jobs to the processors to minimize the number of expensive links.

4.2.1. Problem Modeling

We model the problem of allocating the jobs to processors as an instance of the *extensible bin packing problem*. The extensible bin packing problem can be defined as follows. Consider a set of m bins with sizes $b_1, b_2 .... b_m$. There are n items of sizes $s_1, s_2, s_3 .... s_n$. The bins can be extended and overloaded with items. The load of bin j is l_j and is defined as the total size of the items assigned to it. The size of the bin after allocation is defined as $max(l_j, b_j)$. The objective is to minimize the total size of bins after allocation. The problem instance is to allocate a set of jobs that are waiting to be queued to a set of available processors. The set of jobs that are considered should satisfy the minimum requirement that the total number of processors requested by the jobs is not be greater than the number of available processors.

4.2.2. Approximation algorithm

Olmo and Speranza [37] prove that the Longest Processing Time (LPT) algorithm yields a worst case performance bound of 1.173 OPT where OPT is the cost of an optimal algorithm for the extensible bin packing problem. The working of LPT algorithm is as follows. The bin sizes and the item sizes are in terms of processing times. The jobs are arranged in the decreasing order of the processing time. Then they are allocated one by one to the processor with the largest idle time space.

Our scheduling algorithm is based on LPT with the time domain being replaced by the spatial domain. In our case the bin sizes and item sizes are in terms of number of allocation units of processors. A group of allocation units along the same y–z plane form a bin. The jobs are allocated to the bins. An extension of the bin implies allocating processors across the planes. Each allocation unit is analogous to the time slots in the original LPT algorithm and the overloading of bins is expensive because it involves the use of links consuming excessive power. We consider a two level cost function. The links in the y–z plane have the same cost and the links along the x–dimension have a larger cost.

The assumption in [37] for the optimality bounds is that the optimality bound holds for cases where $max(s_i) \leq min(b_j)$. However *Lemma 1* shows that the solution to a problem instance not respecting this condition is still an approximation scheme comparable to a problem instance respecting this assumption.

Lemma 1: We can find an approximation scheme to a problem instance where $max(s_i) > min(b_j)$.

Proof: Consider a problem instance P where one item k has size $s > min(b_j)$. Let us create a problem instance P' with item k removed and bin $min(b_j)$ removed. The new problem instance P' is exactly in the form of the extensible bin packing problem defined in [37]. So the cost of the original problem instance P is given by,

$$cost(P) \leq cost(P') + s \tag{2}$$

where $cost(P')$ is the cost of the solution provided by any approximation scheme.

In a similar manner if more than one item has a size $> min(b_j)$ then we create a new instance P'' by removing one such item and the bin with size $min(b_j)$ at each step until there is only one bin remaining or the condition $max(s_i) > min(b_j)$ is satisfied. Note that when only one bin is remaining, the problem reduces to the trivial case of packing all the items into one bin. Then the cost of the approximation scheme

$$cost(P) \leq cost(P'') + S \tag{3}$$

where $S = \Sigma s_k, k = 1, 2, ..l$ and l is the number of items that were removed from problem instance P to form problem instance P''.

Lemma 2: The fact that the extension of the bins have to come from other bins does not affect the solution to the allocation problem.

According to our algorithm the basic condition to allot a set of jobs J to a group of processors P is that $size(J) \leq size(P)$. Only then at the least, a completely random non-contiguous allocation of jobs to processors is feasible. So, as long as this condition holds, finding an allocation unit for the extension of the bins is always possible. In choosing the processors for the extension, we further try to minimize the number of x–dimension links that need to be used.

4.3. Scheduling Algorithm

The main goal of our scheduling algorithm is to cut down on communication across as many expensive links as possible and enable switching them to low power modes. In machines like the IBM Blue Gene/P where the 3D torus network is dedicated to partitions, there is risk of fragmentation of resources since once the networking resources are allocated to partitions, they cannot be used by any other partition. In Cray machines [32], where the 3D torus network is shared, the communication links are not dedicated and hence we cannot switch them to low power modes unless we are absolutely sure that no communication will take place through those links. Our proposed spatio–temporal allocation algorithm works irrespective of whether the links are shared or dedicated. In our problem model, there are discrete power costs associated with link types and as more links are powered on, the associated costs increase proportional to the number of links that are up.

Algorithm 1 Power Aware scheduling

Input : A set of jobs $\{J\}$

Output : Allocation of jobs$\{J\}$ *to nodes*

Job ordering: Jobs are ordered in a queue based on some priority function. To schedule the jobs, the total number of processors required by the jobs cannot be more than the total number of processors available/free. Jobs that satisfy this condition are selected and moved to the ready queue.

Pre-processing: Sort the jobs in the ready queue in decreasing order based on number of processors

Step 1: Form bins of processors connected by links with minimum cost (y - z plane)

Step 2: Allocate the jobs in the ready queue in sorted order to the bin with maximum remaining capacity

Step 3: Allocate extensions to bins from non contiguous processors across multiple planes

The scheduling mechanism is shown in Algorithm 1. The algorithm is invoked upon job arrival or termination. Algorithm 1 tries to restrict communication along the same plane and hence enables the links that connect these planes with the other planes to be put in low power mode. It can be seen that there are two phases in the scheduling algorithm. The first phase is the temporal ordering of jobs and the second phase is the scheduling of jobs on the available processors using the extensible bin packing algorithm. For the temporal ordering of jobs, we tried two different strategies. The first strategy is the most commonly used FCFS scheduling with backfilling. The jobs are ordered on an FCFS basis. They are scheduled on processors as they become available. Small jobs that will not affect the scheduling of higher priority jobs are chosen out of order from the queue for backfilling. When the bins are formed, the processors that will be free in a time of τ seconds are also treated as free and are considered to be allocated to jobs. The price paid for this strategy is that the jobs to which these processors are allocated need to wait until the processors are freed. The configurable parameter τ

determines the trade–off between wait time of the jobs and optimal power consumption. The second strategy for prioritizing in the temporal domain is to order the jobs based on a utility function as defined in [38] and used in IBM Blue Gene machines. The jobs are arranged in decreasing order of a utility function which is defined below.

$$U_i = \left(\frac{queue\ time}{wall\ time}\right)^3 \times n_i \tag{4}$$

where n_i is the number of processors requested by job i. It is easy to see that the utility function favors jobs that have been waiting in the queue for a long time and hence prevents these jobs from waiting indefinitely. Also it avoids starvation of large jobs. The combination of the spatial allocation algorithm with the temporal priority schemes helps in achieving a scheduling strategy to minimize the interconnect power consumption.

5. Runtime Estimation

The workload logs that we use for evaluation purposes are collected from the Argonne National Laboratory from January to August 2009. The traces were collected from a 40 rack IBM Blue Gene/P machine with 40, 960 quad–core nodes. Each rack has two midplanes and each midplane has 512 nodes. The nodes in the IBM Blue Gene machine are connected by 3D torus networks [31]. The nodes are grouped into partitions and each job is assigned to a few partitions. The performance of a scheduling

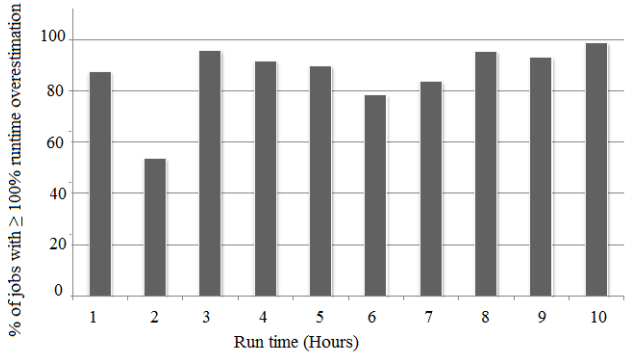

Figure 2: Percentage of jobs with more than 100% overestimation of runtime by users

algorithm on a space sharing machine like the IBM Blue Gene is dependent on the accuracy of runtime estimates. The impact of inaccuracy in user provided runtimes has been studied in a lot of research works [39, 40, 41]. Tang et al. [41] provide a scheme for estimating user runtimes based on the historical data of different users and projects. The impact of inaccuracies in user provided runtime estimates has been further explored analytically by Tsafrir [42]. The conventional F–model [40] associates a user provided runtime estimate e with the actual runtime r by the relation $e = r.F$ where F is a badness factor and $F > 1$. Tsafrir [42] observes that the conventional F

may be misleading to the conclusion that the performance is insensitive to estimation accuracy of F. However most of the previous works including [41] are based on the F–model. Tsafrir et al. [43] observed the modal nature of the user provided runtime estimates. The authors propose a model that aims to map the percentage of jobs to their most popular runtime estimates by users. Figure 2 shows the discrepancy between the user provided and the actual runtimes in the Argonne National Lab trace that we use. It can be seen that more than 90% of jobs with runtimes greater than 8 hours have an overestimation of more than 100%. Any scheduling algorithm that is based on user provided runtimes as such is hence bound to waste a lot of resources. We found that on the average the difference between user estimated and actual runtimes was around 4000 seconds. Figure 2 supports the intuitive notion that users always tend to provide an overestimate of the runtime of the jobs either to avoid the risk of the jobs being shut down because they exceeded the requested runtime or because of the nature of rounding to the nearest popular number (like multiples of $1,000$).

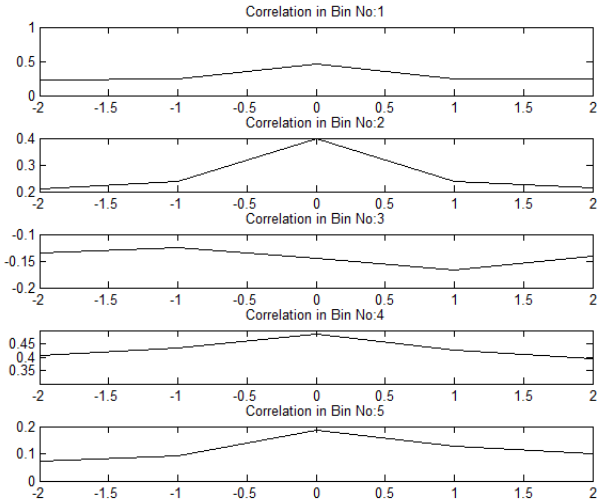

Figure 3: Cross correlation between user provided and actual runtimes in different bins (bin size = 4000 seconds)

The runtime estimation model that we propose exploits the modal nature of the user provided runtimes and tries to improve the accuracy of the estimates. We group the user provided runtime estimates into discrete bins. We then model each bin separately using regression. Modeling and estimating the actual runtimes for each bin separately helps to increase the accuracy of estimation. The error in user provided estimates has a significant impact on power savings. For instance if the user provided estimate of runtime of a job is highly overestimated, the scheduling algorithm may decide that a contiguous allocation is not possible and hence go for a non–contiguous allocation. This would turn out to be highly inefficient in terms of power savings. If the estimate was accurate, the scheduling algorithm might decide to wait and schedule after the processors are free.

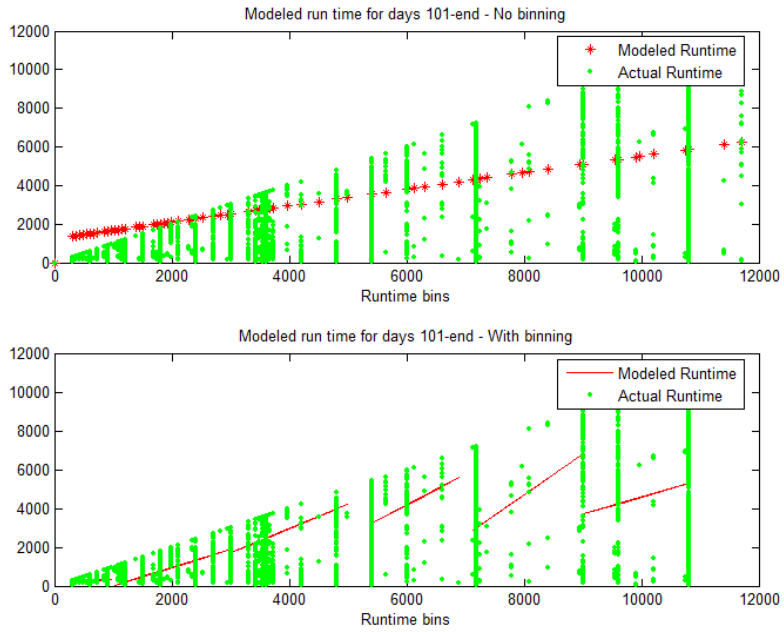

Figure 4: Modeled run time with and without binning for days 101 till the end (bin size = 4000 seconds

Figure 3 shows the correlation between user provided and actual runtime of jobs in different bins of size 4000 seconds. We see that the cross-correlation between user provided and actual run times are completely different in different bins. This explains the intuition behind our binning technique. Any modeling technique that ignores this fact would result in an averaging effect and hence an increase in error in the estimates. We use the trace data for the first 100 days to create our model and use the model to adjust the user provided runtimes in the rest of the days. Figure 4 shows the adjusted estimates when a single model is used for the entire runtime range and when different models are used for different bins. We see that the models are completely different for each of the bins and hence the mean square error in estimates drops from 2.3271×10^7 without binning to 6.4061×10^6 with binning.

6. Simulation

Figure 5 shows the number of jobs that are running concurrently per hour. The average number of jobs that are running per hour was around 19 and the 99th percentile was 43. The total number of jobs that were considered for scheduling at the end of the simulation was $68,936$ and the maximum number of available cores was $163,840$. Figure 6

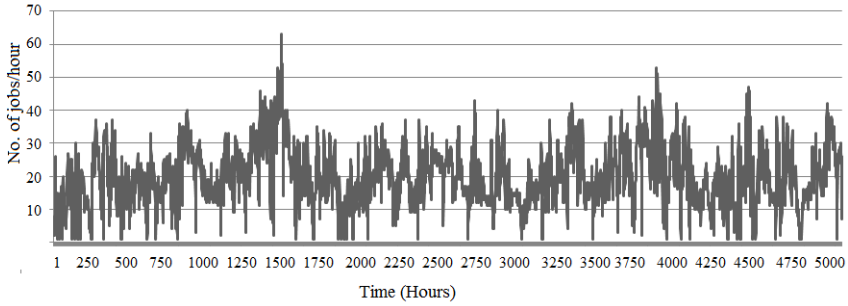

Figure 5: Number of concurrent jobs per hour

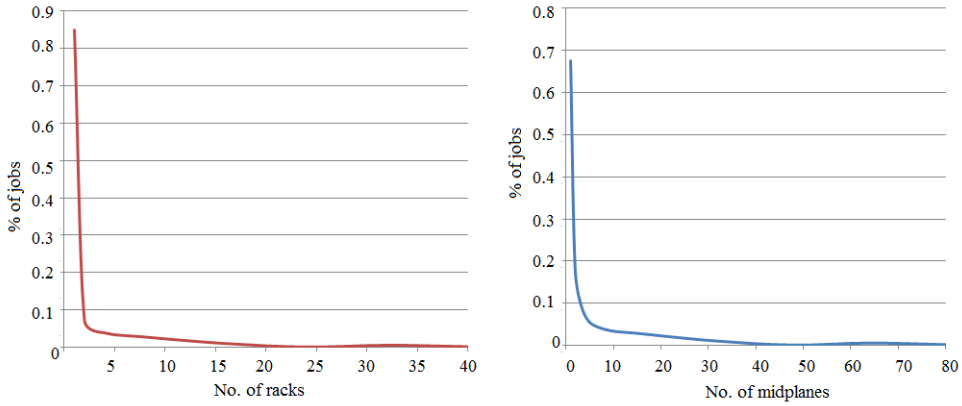

Figure 6: Number of racks and midplanes required by jobs

6.1. Simulation Results

We developed a simulator in MATLAB to simulate our scheduling scheme. We chose MATLAB as our simulation platform since it is relatively simpler to implement a 3D torus connected machine in MATLAB. Also since the simulation is at a coarse granularity complex functionalities provided by other high level programming languages were not necessary. The simulator takes as input the submit time, runtime and the number of processors required to run the jobs and creates a scheduling for the jobs. Jobs are in the wait queue and when a set of processors are available, an instance of extensible bin packing problem is solved. To avoid trivial instances of the allocation problem, the inter–arrival times of the jobs was adjusted to have at least 3 jobs in the wait queue when an instance of bin packing is solved. Even though some of the processors allocated to a job are idle until all processors allocated to the job are really free, this idle period does not exceed τ seconds. In our simulations we used the value of τ to be 1000 seconds. We assume that only the links that connect communicating nodes are up and the rest of the links are shut down completely. We use a conservative method to decide on what links should be up to ensure connection between commu-

nicating nodes. If two midplanes in the rack are allocated to the same job, we keep all the *z*–dimension links up. This ensures that all redundant links are up to connect the midplanes and hence represent the maximum number of links needed to connect the two midplanes. Note that in practice some of these links can be shut down and hence lead to even more power savings. Column cables are needed only when there are communicating nodes across racks. We assume a cost function of 2 : 1 for the ratio of power consumption of the *x*-dimension cables to cables along the other dimensions. As far as we know, the exact values of the power values are not available. A simple reason behind using this ratio is because in the link cards in IBM Blue Gene, the number of column cables are twice that of rack or row cables. The additional cables are the spare column cables. Note that these inter-row links are also expensive in terms of communication and hence the assumption that the cost of using these links is higher is valid not only from a power consumption point of view but also from a communication perspective. Hence the extensible bin packing problem formulation also helps to reduce the communication costs.

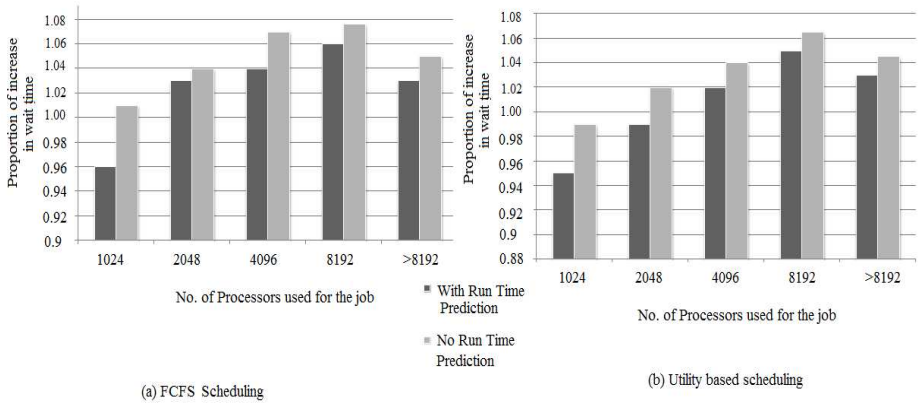

Figure 7: Increase in wait time for jobs normalized to the greedy first fit scheduling (with and without runtime prediction)

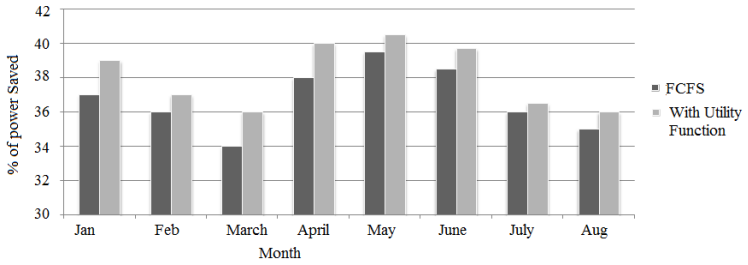

Figure 8: Power savings achieved by putting network elements to sleep over greedy first fit scheduling

In order to quantify the benefits of our scheduling algorithm we compare it with a greedy first fit scheduling algorithm combined with each of the two temporal scheduling strategies - FCFS with backfilling and utility based temporal ordering of jobs. As mentioned before, FCFS is the most commonly used scheduling strategy in supercomputers and utility based job ordering in the time domain is used in IBM Blue Gene for production jobs. Figure 7 shows the increase in wait time for jobs compared to a first fit scheduling scheme. We see that when the number of processors is low, the wait time is also low. This is intuitive because getting hold of a few free processors is easier than getting hold of large number of contiguous free processors. Note that when solving the extensible bin packing problem instace, processors that will be free in τ seconds are also considered as free. This leads to a slight increase in the wait time for the jobs. Nevertheless, this increase is never more than 7%. It can be seen that the wait time actually decreases significantly with run time prediction. This is because with a better accuracy in runtime prediction, the jobs can be assigned to processors based on the time when the processors will be free and hence resource fragmentation is reduced. This directly translates to a decrease in the wait time for jobs. Also the utility based scheduler consistently outperforms the FCFS scheme.

Figure 8 shows the average percentage of power that can be saved by our proposed scheduling scheme as against the simple first fit scheduling with the two temporal job ordering strategies. It can be seen that on the average around 35%-38% of power can be saved in the communication links by power aware scheduling of jobs. Our calculation of power savings is based on the dynamic power consumption of link cards. A 20% savings in power means that 20% of the links are shut down. Inlcusion of static power might offset these calculations a little and the power savings percentage would be lesser. However, even if we assume that the static power consumed is as high as 30%, the power savings by our algorithm would still be around 30%. Fine grained techniques can reduce the static power consumption of the network components as well. However it is highly likely that they will have a significant impact on latency. We do not have an explicit metric for evaluating the impact on the run time of jobs. The reasons are two fold. First, we do not have access to a fine grained trace with the communication phases of the jobs and the congestion periods. Second, our scheduling algorithm does not shut down links between communicating nodes. We just speculate the potential power savings in network links by careful allocation of jobs with almost zero impact on performance (and a slight increase in wait time). We believe that current systems do not have mechanisms to do this.

7. Conclusion and Future work

We have presented and evaluated a power aware scheduling scheme for a tightly coupled HPC system with 3D torus interconnects . We have shown that a power savings of 30-40% is possible in the interconnects by shutting down the links with almost no performance loss. We also show that with a more accurate prediction of run time of jobs an even better scheduling can be achieved that can lead to improvements in performance.

In this paper we do not focus on fine-grained power control techniques for the links. An accurate analysis of performance impact of the fine grained power control tech-

niques would require more detailed trace data which is difficult to obtain. A detailed analysis of communication patterns could help improve the power savings manifold. We would like to test the performance of our algorithm on a real time supercomputer and study further power saving options. An analysis of storage I/O in HPC environments is a field that has been relatively less focused upon. Taking into consideration the very high speed storage networks and interconnects would provide a more comprehensive analysis on power savings. The cost models that we have right now has two different costs for links. In future we would like to develop a more complex cost model with multiple cost functions. Exploring power saving options in other topologies like fat tree is a possible extension for the paper.

References

[1] H. D. Simon, T. Zacharia, R. Stevens, Modeling and simulation at the exascale for energy and the environment, Department of Energy Technical Report,.

[2] W. J. Camp, Exascale Ambitions : What, me worry?, `http://www.lanl.gov/orgs/hpc/salishan/pdfs/Salishan\%20slides/Camp2.pdf` (2008).

[3] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, H. Franke, Reducing disk power consumption in servers with drpm, Computer 36 (2003) 59–66.

[4] T. D. Burd, T. A. Pering, A. J. Stratakos, R. W. Brodersen, A dynamic voltage scaled microprocessor system, IEEE Journal of Solid-state Circuits 35 (2000) 1571–1580.

[5] D. Liu, C. Svensson, Trading speed for low power by choice of supply and threshold voltages, IEEE Journal of Solid-state Circuits 28 (1993) 10–17.

[6] S. Kamil, J. Shalf, E. Strohmaier, Power efficiency in high performance computing, in: IPDPS '08, 2008, pp. 1–8.

[7] O. Sonmez, H. Mohamed, D. Epema, Communication-aware job placement policies for the koala grid scheduler, in: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing, E-SCIENCE '06, IEEE Computer Society, Washington, DC, USA, 2006, pp. 79–.

[8] A. B. Nagarajan, F. Mueller, C. Engelmann, S. L. Scott, Proactive fault tolerance for hpc with xen virtualization, in: Proceedings of the 21st annual international conference on Supercomputing, ICS '07, ACM, New York, NY, USA, 2007, pp. 23–32.

[9] A. Verma, P. Ahuja, A. Neogi, Power-aware dynamic placement of hpc applications, in: Proceedings of the 22nd annual international conference on Supercomputing, ICS '08, ACM, New York, NY, USA, 2008, pp. 175–184.

[10] T. Mukherjee, A. Banerjee, G. Varsamopoulos, S. K. S. Gupta, S. Rungta, Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers, Comput. Netw. 53 (2009) 2888–2904.

[11] Q. Tang, S. K. S. Gupta, G. Varsamopoulos, Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach, IEEE Trans. Parallel Distrib. Syst. 19 (2008) 1458–1472.

[12] D. Niyato, S. Chaisiri, L. B. Sung, Optimal power management for server farm to support green computing, in: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 84–91.

[13] P. Ranganathan, P. Leech, D. Irwin, J. Chase, Ensemble-level power management for dense blade servers, in: Proceedings of the 33rd annual international symposium on Computer Architecture, ISCA '06, IEEE Computer Society, Washington, DC, USA, 2006, pp. 66–77.

[14] J. Moore, J. Chase, P. Ranganathan, R. Sharma, Making scheduling "cool": temperature-aware workload placement in data centers, in: Proceedings of the annual conference on USENIX Annual Technical Conference, ATEC '05, USENIX Association, Berkeley, CA, USA, 2005, pp. 5–5.

[15] E. Frachtenberg, D. G. Feitelson, F. Petrini, J. Fernandez, Adaptive parallel job scheduling with flexible coscheduling, IEEE Trans. Parallel Distrib. Syst. 16 (2005) 1066–1077.

[16] W. Tang, Z. Lan, N. Desai, D. Buettner, Y. Yu, Reducing fragmentation on torus-connected supercomputers, in: IPDPS '11, 2011.

[17] ANL Intrepid, Parallel workloads archive, `http://www.cs.huji.ac.il/labs/parallel/workload/` (2009).

[18] A. Oliner, R. Sahoo, J. Moreira, M. Gupta, A. Sivasubramaniam, Fault-aware job scheduling for bluegene/l systems, in: Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International, 2004, p. 64.

[19] S. Majumdar, D. L. Eager, R. B. Bunt, Scheduling in multiprogrammed parallel systems, in: Proceedings of the 1988 ACM SIGMETRICS conference on Measurement and modeling of computer systems, SIGMETRICS '88, ACM, New York, NY, USA, 1988, pp. 104–113.

[20] Y. Zhang, H. Franke, J. Moreira, A. Sivasubramaniam, An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration, Parallel and Distributed Systems, IEEE Transactions on 14 (3) (2003) 236 – 247.

[21] D. G. Feitelson, M. A. Jette, Improved utilization and responsiveness with gang scheduling, in: Proceedings of the Job Scheduling Strategies for Parallel Processing, IPPS '97, Springer-Verlag, London, UK, 1997, pp. 238–261.

[22] B. Yoo, C. Das, A fast and efficient processor allocation scheme for mesh-connected multicomputers, Computers, IEEE Transactions on 51 (1) (2002) 46 –60.

[23] J. Kim, C. Das, W. Lin, A top-down processor allocation scheme for hypercube computers, Parallel and Distributed Systems, IEEE Transactions on 2 (1) (1991) 20 –30.

[24] V. Lo, K. Windisch, W. Liu, B. Nitzberg, Noncontiguous processor allocation algorithms for mesh-connected multicomputers, Parallel and Distributed Systems, IEEE Transactions on 8 (7) (1997) 712 –726.

[25] I. Ababneh, Availability-based noncontiguous processor allocation policies for 2d mesh-connected multicomputers, Journal of Systems and Software 81 (7) (2008) 1081–1092.

[26] C.-Y. Chang, P. Mohapatra, Performance improvement of allocation schemes for mesh-connected computers, J. Parallel Distrib. Comput. 52 (1998) 40–68.

[27] K.-H. Seo, Fragmentation-efficient node allocation algorithm in 2d mesh-connected systems, in: Proceedings of the 8th International Symposium on Parallel Architectures,Algorithms and Networks, ISPAN '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 318–323.

[28] W. Qiao, L. M. Ni, Efficient processor allocation for 3d tori, in: Proceedings of the 9th International Symposium on Parallel Processing, IPPS '95, IEEE Computer Society, Washington, DC, USA, 1995, pp. 466–471.

[29] H. Choo, S.-M. Yoo, H. Y. Youn, Processor scheduling and allocation for 3d torus multicomputer systems, Parallel and Distributed Systems, IEEE Transactions on 11 (5) (2000) 475 –484.

[30] Y. Aridor, T. Domany, O. Goldshmidt, Y. Kliteynik, E. Shmueli, J. Moreira, Multitoroidal interconnects for tightly coupled supercomputers, Parallel and Distributed Systems, IEEE Transactions on 19 (1) (2008) 52 –65.

[31] A. Gara, M. A. Blumrich, D. Chen, G. L.-T. Chiu, P. Coteus, M. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, P. Vranas, Overview of the Blue Gene/L system architecture, IBM Journal of Research and Development 49 (2-3) (2005) 195–212.

[32] Cray XE6m, `http://www.cray.com/Assets/PDF/products/xe/CrayXE6Brochure.pdf` (2009).

[33] P. Balaji, H. Naik, N. Desai, Understanding network saturation behavior on large-scale blue gene/p systems, in: Proceedings of the 2009 15th International Conference on Parallel and Distributed Systems, ICPADS '09, IEEE Computer Society, 2009, pp. 586–593.

[34] D. Weisser, N. Nystrom, C. Vizino, S. Brown, J. Urbanic, Optimizing Job Placement on the Cray XT3, in: 48th Cray User Group Proceedings(CUG 2006), 2006.

[35] A. Bhatelé, E. Bohm, L. V. Kalé, A Case Study of Communication Optimizations on 3D Mesh Interconnects, in: Proceedings of the 15th International Euro-Par Conference on Parallel Processing, Euro-Par '09, Springer-Verlag, 2009, pp. 1015–1028.

[36] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, D. Wetherall, Reducing network energy consumption via sleeping and rate-adaptation, in: NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, 2008.

[37] P. Dell'Olmo, H. Kellerer, M. G. Speranza, Z. Tuza, A 13/12 approximation algorithm for bin packing with extendable bins, Information Processing Letters 65 (5) (1998) 229 – 233.

[38] W. Tang, Z. Lan, N. Desai, D. Buettner, Fault-aware, utility-based job scheduling on blue, gene/p systems., in: CLUSTER, IEEE, 2009, pp. 1–10.

[39] T. Suzuoka, J. Subhlok, T. Gross, Evaluating job scheduling techniques for highly parallel computers, Technical Report CMU-CS-95-149, School of Computer Science, Carnegie Mellon University.

[40] A. Weil, Utilization and predictability in scheduling the ibm sp2 with backfilling, in: Proceedings of the 12th. International Parallel Processing Symposium on International Parallel Processing Symposium, IPPS '98, IEEE Computer Society, Washington, DC, USA, 1998, pp. 542–.

[41] W. Tang, N. Desai, D. Buettner, Z. Lan, Analyzing and adjusting user runtime estimates to improve job scheduling on the blue gene/p, in: IPDPS, 2010, pp. 1–11.

[42] D. Tsafrir, Using inaccurate estimates accurately, in: Proceedings of the 15th international conference on Job scheduling strategies for parallel processing, JSSPP'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 208–221.

[43] D. Tsafrir, Y. Etsion, D. G. Feitelson, Modeling user runtime estimates, in: JSSPP, 2005, pp. 1–35.