Cooperative Batch Scheduling for HPC Systems

Xu Yang*, Zhiling Lan*
*Department of Computer Science, Illinois Institute of Technology, Chicago, Illinois, USA 60616
{xyang56}@hawk.iit.edu, lan@iit.edu

***Abstract*—The batch scheduler is an important system software serving as the interface between users and HPC systems. Users submit their jobs via batch scheduling portal and the batch scheduler makes scheduling decision for each job based on its request for computing sources, i.e. core-hours. However, jobs submitted to HPC systems are usually parallel applications and their lifecycle consists of multiple running phases, such as computation, communication and I/O. Thus, the running of a job could involve different kinds of system resources, such as power, network bandwidth, I/O bandwidth, storage, etc. Today's batch schedulers rarely take these resource requirements into consideration for making scheduling decisions, which has been identified as one of the major culprits for system wide performance loss.**

In this work, we propose a cooperative batch scheduling framework for HPC systems. The motivation of our work is to take these ignored factors such as job power consumption, job communication pattern, resource locality and network topology in consideration, and make scheduling decisions in a coordinated way. Our major contribution consist of a set of scheduling models and algorithms for addressing some chronic issues such as high power cost, job interference and system fragmentation for HPC systems. The proposed models and algorithms in this work have been evaluated by the means of simulation using real workload traces and application traces collected from production HPC systems. Preliminary experimental results show that our models and algorithms can effectively improve batch scheduling with regard to system performance, quality of service for applications and operating cost.

1. Motivation

The high performance computing (HPC) systems comprise hundreds of thousands compute nodes connected by large scale interconnected networks. The insatiable demand for computing power continues to drive the involvement of ever-growing HPC systems. The HPC systems with such unprecedented scale are expected to face many challenges. Some of the most prominent challenges are including, energy and power, network contention and job interference, concurrency and locality. These challenges demand great technical breakthroughs in many aspects of the HPC system's software and hardware stack.

The objective of this work is to provide solutions for these challenges from the perspective of batch scheduling. The batch scheduler, serving as the interface between HPC systems and users, is an irreplaceable layer in the HPC system software stack. The batch scheduler is responsible for system resource management and jobs scheduling. Users submit applications (jobs) via the batch scheduling portal. The job scheduling module makes scheduling decisions for each job based on many factors, such as job size, job run time, job priority, etc. In the meanwhile, the resource management module takes feedback from the system, keeps resource availability updated and allocates resources to each job. A good batch scheduler should utilize the limited system resources in a coordinated way and guarantee the quality of service to all jobs.

A traditional HPC batch scheduler makes scheduling decisions based only on the number of required nodes and expected run time of each job. However, the lifecycle of a running job involves various kinds of resource, such as compute nodes, energy consumption, network and I/O bandwidth, etc. The traditional batch scheduler rarely consider requirements of these resources in job's different running phases, which sometimes turn out to be major culprit of system wide performance loss. For example, when the batch scheduler assigns computing nodes to jobs without any knowledge about their communication patterns, these jobs may compete for the network bandwidth in an uncoordinated way, causing interference between each other, thus resulting serious performance degradation. Hence we argue the current batch scheduling framework is in desperate need of a more cooperative methodology.

We list three major challenges for the HPC systems. For each each of them, our research provides solutions with new batch scheduling models and algorithms.

1.1. Energy Cost

As HPC systems continue to grow so does their energy consumption. The electricity bill is now a leading component of total cost of ownership(TCO) of HPC systems. The typical current petascale system on average consumes 2-7 MW of power [1]. Case in point, the Argonne Leadership Computing Facility (ALCF) budgets approximately $1M annually for electricity to operate its primary supercomputing

resource. Based on current projections, exascale supercomputers will consume 60-130 MW, which will prove to be an unbearable burden for any facility. Therefore, energy cost savings are crucial for reducing the operational cost of extreme scale systems.

1.2. Fragmented Allocation

As the scale of supercomputers increases, so do their interconnected networks. Torus interconnection is widely used in HPC systems, such as Cray XT/XE and IBM Blue Gene series systems, due to their linear per node cost scaling and their competitive overall performance. A growing network means an increasing network diameter (i.e., the maximum distance between a pair of nodes) and a decreasing bisection bandwidth relative to the number of nodes. Consequently, applications running on torus-connected systems suffer great performance variability caused by the increasing network scale.

The partition based job placement policy adopted by IBM Blue Gene series systems is in favor of application's performance by preserving locality of allocated nodes and reducing network contention caused by concurrently running jobs sharing network bandwidth. However, this strategy can cause internal fragmentation (when more nodes are allocated to a job than it requests) and external fragmentation(when sufficient nodes are available for a request, but they can not be allocated contiguously), therefore leading to poor system performance (e.g., low system utilization and high job response time) [2]. The non-contiguous job placement policy adopted by Cray XT/XE series systems could eliminate internal and external fragmentation as seen in partition-based systems, thereby leading to high system utilization. Nevertheless, it introduces other problems such as scattering application processes all over the system. The non-contiguous node allocation can make inter-process communication less efficient and cause network contention among concurrently running jobs [3], thereby resulting in poor job performance especially for those communication-intensive jobs. As systems continue growing in size, a fundamental problem arises: *how to effectively balance job performance with system performance on torus-connected machines?*

1.3. Network Contention

The scale of supercomputers continually grows at a breakneck pace to accommodate the computing power requirements for scientific research. Supercomputers have tens of thousands of nodes and serve as an irreplaceable research vehicle for scientific problems with increasing size and complexity. Supercomputers are usually employed as a shared resource to accommodate many parallel applications (jobs) running concurrently. These parallel jobs share the system infrastructure such as network and I/O bandwidth, and inevitably there is contention over these shared resources. As supercomputers continue to evolve, these shared resources are increasingly the bottleneck for performance.

A prominent problem with network sharing is the resulting contention. Network contention between concurrently running jobs on both torus and dragonfly HPC systems is a primary cause of performance degradation [4]. This performance degradation can propagate into the queueing time of the following submitted jobs, thus leading to lower system throughput and utilization [5]. Optimizing job allocation and avoiding network sharing are hence crucial to alleviate the potential performance degradation. In order to do so effectively, an understanding of the interference among concurrently running jobs, their communication patterns, and contention in the network is required.

2. Accomplished Work

We have accomplished scheduling models, algorithms and comprehensive analyses for each of the aforementioned challenges.

2.1. Energy Cost-aware Scheduling

We hypothesize that it is possible to save a significant amount on an electric bill by exploiting a dynamic electricity pricing policy. To date, dynamic electricity pricing policies have been widely adopted in Europe, North America, Oceania, and parts of Asia. Under dynamic pricing, the power grid has on-peak time (when it bears a heavier burden and consequently the electricity price is higher) and off-peak time (when there is less demand for electricity and the price is lower) alternatively in a day. We develop a job power aware scheduling mechanism [6] [7]. The novelty of this scheduling mechanism is that *it can reduce system's electricity bill by scheduling and dispatching jobs according to their power profiles and the real time electricity price, while causing negligible impact on the system's utilization and scheduling fairness.* Preferentially, it dispatches the jobs with higher power consumption during the off-peak period, and the jobs with lower power consumption during the on-peak period.

We evaluate our job power aware scheduling design via extensive trace-based simulations and a case study of Mira. We present a series of experiments comparing our design against the popular first-come, first-serve (FCFS) scheduling policy, with backfilling done in three different aspects (electricity bill saving, scheduling performance, and job performance). Our preliminary results demonstrate that our design can cut electricity bill by up to 23% without an impact on overall system utilization. Considering HPC centers often spend millions of dollars on even the least expensive energy contracts, such savings can translate into a reduction of hundreds of thousands in terms of TCO.

2.2. Locality-aware Scheduling

We presented a *window-based locality-aware scheduling design* [8] [9]. Our design is based on two key observations.

First, existing scheduling system makes decisions in a per-job manner. Each job is dispatched to system resources without considering subsequent jobs. While making isolating job decision may provide a good short-term optimization, it is likely to result in poor performance in the long term. Second, existing scheduling system maintains a list of free nodes for resource allocation. While special processor ordering (e.g., using a space filling curve) is often adopted for preserving node locality in the list, the node list becomes fragmented as time goes by and subsequent jobs inevitably get dispersed nodes allocation due to the lack of contiguous node list.

Rather than one-by-one job scheduling, our design takes a "window" of jobs (i.e., multiple jobs) into consideration for making prioritizing and allocation decision, to prevent the short-term decision from obfuscating future optimization. Our job prioritizing module maintains a "window" of jobs, and these jobs are placed into the window to maintain job fairness (e.g., through FCFS). Rather than allocating jobs one by one from the head of the wait queue as existing schedulers do, we make scheduling decision for a "window" of jobs at a time. Our resource allocation module takes a contiguous set of nodes as a *slot* and maintains a list of such slots. These slots have different sizes and each may accommodate one or more jobs. The allocation of the jobs in the window onto the slot list is conducted in such a way as to maximize system utilization. We formalize the allocation of a window of jobs to a list of slots as a 0-1 Multiple Knapsack Problem (MKP) and present two algorithms, namely Branch&Bound and Greedy, to solve the MKP.

We evaluate our design via extensive trace-based simulations. In this paper, we conduct a series of experiments comparing our design against the commonly used FCFS/EASY backfilling scheduling that is enhanced with processor ordering. Our preliminary results demonstrate that our design can reduce average job wait time by up to 28% and average job response time by 30%, with a slight improvement on overall system utilization.

2.3. Topology-aware Job Scheduling

We choose three representative HPC applications from the DOE Design Forward Project [10] and analyze the interference among them when running on both torus and dragonfly networks. Our study is based on simulation with CODES, a high-fidelity, flit-level HPC network simulation toolkit [11].

2.3.1. Torus Network. In the current generation supercomputers, torus topology is widely used [12], [13], [14]. These supercomputers usually adopt two job placement policies. The partition-based placement adopted in the Blue Gene series of supercomputers [15] favors application performance through exclusive resource allocation and the locality that implies. However, partition-based placement could cause intrajob interference that resulting application performance loss. On the other hand, *noncontiguous placement* adopted by the Cray XT/XE series [16], assigns free nodes to jobs regardless of contiguity, although of course efforts are made to maximize locality. The noncontiguous placement introduces interjob interference due to the interleaving of the jobs' nodes,

In our work [17], we provide in-depth analysis of intra- and interjob communication interference with different job allocation strategies on torus connected HPC systems. Based on our simulation study, we make following observations.

- Compact allocation may not be necessary for every application.
- A good rank-to-node mapping strategy can greatly improve an application performance when a specific allocation is given.
- An optimal size for allocation units should be determined according to an application's dominant communication pattern. In general, a unit size should be large enough to accommodate neighboring communication in the application.
- Interjob interference is inevitable with a noncontiguous allocation. However, choosing the proper allocation unit size with communication pattern awareness can help alleviate the resulting negative effects.

2.3.2. Dragonfly Network. High-radix, low-diameter dragonfly networks will be a common choice in next-generation supercomputers. The dragonfly topology can lower the overall cost of the interconnect, improve network bandwidth and reduce packet latency [18], making it a very promising choice for building supercomputers with millions of cores. Even with such powerful networks, intelligent job placement is of paramount importance to the efficient use of dragonfly connected systems [19], [20]. Recent studies suggest that random node allocation for parallel jobs, coupled with adaptive routing, can alleviate local congestion, eliminate hot-spots and achieve load-balancing for dragonfly networks [20], [21], [22]. These studies explore the possible job placement and routing configurations that could optimize the overall network performance without examining how different configurations impact the progress of individual applications. *We study the implications of contention for shared network links in the context of multiple HPC applications running on dragonfly systems when different job placement and routing configurations are in use* [23]. Our analyses focus on the overall network performance as well as the performance of concurrently executing applications in the presence of network contention. We make the following observations through extensive simulations.

- Concurrently running applications on a dragonfly network interfere with each other when they share network resources. Communication-intensive applications "bully" their less intensive peers and obtain performance improvement at the expense of less intensive ones.
- Random placement of application processes in the dragonfly can improve the performance of communication-intensive applications by enabling

network resource sharing, though it introduces interference causing performance degradation to the less intensive applications.

- Contiguous placement can be beneficial to the consistent performance of less communication-intensive applications by minimizing network resource sharing, because it reduces the opportunities for traffic from other applications to be loaded on links that serve as minimal routes for the less intensive application. However, this comes with the downside of reduced system performance due to load imbalance.

We envision that future HPC schedulers will adopt a flexible job allocation mechanism which combines the best of both contiguous and non-contiguous allocation strategies. Such a flexible mechanism would take shared resource needs of jobs into account when making allocation decisions (e.g., network). With knowledge and analysis of job communication patterns, it can be identified which jobs require network isolation and locality, and to what degree. Then, rather than allocating each job in a "know-nothing" manner, one may specialize allocation so that, for example, only the jobs with stringent network needs are given compact, isolated allocations, resulting in maximized utilization and minimized perceivable resource contention effects.

References

[1] C. Patel, R. Sharma, C. Bash, and S. Graupner, "Energy aware grid: Global workload placement based on energy efficiency," in *ASME 2003 International Mechanical Engineering Congress and Exposition*. American Society of Mechanical Engineers, 2003, pp. 267–275.

[2] P. Krueger, T.-H. Lai, and V. A. Dixit-Radiya, "Job scheduling is more important than processor allocation for hypercube computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 5, pp. 488–497, May 1994.

[3] J. A. Pascual, J. Navaridas, and J. Miguel-Alonso, "Job scheduling strategies for parallel processing," E. Frachtenberg and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. Effects of Topology-Aware Allocation Policies on Scheduling Performance, pp. 138–156. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04633-9_8

[4] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs, "There goes the neighborhood: Performance degradation due to nearby jobs," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 41:1–41:12. [Online]. Available: http://doi.acm.org/10.1145/2503210.2503247

[5] A. Jokanovic, J. Sancho, G. Rodriguez, A. Lucero, C. Minkenberg, and J. Labarta, "Quiet neighborhoods: Key to protect job performance predictability," in *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, May 2015, pp. 449–459.

[6] X. Yang, Z. Zhou, S. Wallace, Z. Lan, W. Tang, S. Coghlan, and M. E. Papka, "Integrating dynamic pricing of electricity into energy aware scheduling for hpc systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 60:1–60:11. [Online]. Available: http://doi.acm.org/10.1145/2503210.2503264

[7] S. Wallace, X. Yang, V. Venkant, S. Coghlan, M. E. Papka, and Z. Lan, "A data driven scheduling approach for power management on hpc systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '16, 2016.

[8] 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 *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*, Sept 2014, pp. 140–148.

[9] 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 *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, May 2015, pp. 439–448.

[10] Department of Energy, "Exascale Initiative," Accessed April 2, 2016, available online http://www.exascaleinitiative.org/design-forward.

[11] M. Misbah, D. C. Christopher, B. R. Robert, and C. Philip, "Enabling parallel simulation of large-scale hpc network systems," in *TRANSACTIONS ON PARALLEL AND DISTRIBUTED COMPUTING,*. IEEE, 2015, pp. VOL. X, NO. X, 2015.

[12] D. Chen, N. Eisley, P. Heidelberger, R. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. Satterfield, B. Steinmacher-Burow, and J. Parker, "The ibm blue gene/q interconnection fabric," *IEEE Micro*, vol. 32, no. 1, pp. 32–43, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1109/MM.2011.96

[13] Y. Ajima, T. Inoue, S. Hiramoto, Y. Takagi, and T. Shimizu, "The tofu interconnect," *IEEE Micro*, vol. 32, no. 1, pp. 21–31, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1109/MM.2011.98

[14] ORNL, "Titan system overview," Accessed October 14, 2015, available online https://www.olcf.ornl.gov/kbarticles/titan-system-overview.

[15] A. Gara, M. A. Blumrich, D. Chen, G.-T. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay *et al.*, "Overview of the blue gene/l system architecture," *IBM Journal of Research and Development*, vol. 49, no. 2.3, pp. 195–212, 2005.

[16] C. Albing, N. Troullier, S. Whalen, R. Olson, J. Glenski, H. Pritchard, and H. Mills, *Scalable Node Allocation for Improved Performance in Regular and Anisotropic 3D Torus Supercomputers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 61–70. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24449-0_9

[17] X. Yang, J. Jenkins, M. Mubarak, R. B. Ross, and Z. Lan, "Study of intra- and interjob interference on torus networks," in *submitted to The 22nd IEEE International Conference on Parallel and Distributed Systems*, ser. ICPADS, 2016.

[18] J. Kim, W. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, June 2008, pp. 77–88.

[19] A. Bhatele, N. Jain, Y. Livnat, V. Pascucci, and P.-T. Bremer, "Analyzing network health and congestion in dragonfly-based supercomputers," in *Proceedings of the IEEE International Parallel & Distributed Processing Symposium*, ser. IPDPS '16 (to appear). IEEE Computer Society, May 2016, lLNL-CONF-678293.

[20] N. Jain, A. Bhatele, X. Ni, N. J. Wright, and L. V. Kale, "Maximizing throughput on a dragonfly network," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 336–347. [Online]. Available: http://dx.doi.org/10.1109/SC.2014.33

[21] A. Bhatele, W. D. Gropp, N. Jain, and L. V. Kale, "Avoiding hot-spots on two-level direct networks," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, Nov 2011, pp. 1–11.

[22] J. Brandt, K. Devine, A. Gentile, and K. Pedretti, "Demonstrating improved application performance using dynamic monitoring and task mapping," in *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*. IEEE, 2014, pp. 408–415.

[23] X. Yang, J. Jenkins, M. Mubarak, R. B. Ross, and Z. Lan, "Watch out for the bully! job interference study on dragonfly network," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '16, 2016.