Power, Reliability, Performance: One System to Rule Them All

Bilge Acun, Akhil Langer, Esteban Meneses, Harshitha Menon, Osman Sarood, Ehsan Totoni, Laxmikant V. Kalé

Traditionally, the emphasis of High Performance Computing (HPC) data centers and applications has been on performance. However, it is anticipated that future generation supercomputing systems will face major challenges in reliability, power management, and thermal variations. Disruptive solutions are required to optimize performance in the presence of these challenges. We believe that a smart parallel runtime system that is part of each job, and that interacts with an adaptive resource manager for the whole machine, is key to overcome the challenges of next generation supercomputing data centers. We have demonstrated that a smart and adaptive runtime system can:

- improve efficiency in a power-constrained environment [1],
- increase performance with load balancing algorithms [2],
- control the reliability of supercomputers with substantial thermal variations [3],
- configure hardware components to operate within power constraints and/or to save energy [4], [5].

Although these research directions were developed in isolation, they indicate that smart runtime systems have a great potential to overcome the barriers towards exascale computing. What the HPC community lacks is an integrated solution that combines past research into a single system that optimizes across multiple dimensions. We propose a comprehensive design in which the data center resource manager:

- dynamically interacts with the individual runtime systems of jobs,
- optimizes for both performance and power consumption, and
- operates in an environment with system failures under constraints supplied by users or administrators.

At the heart of the proposed solution for power-efficiency, reliability and performance of an HPC data center lies an adaptive and dynamic parallel runtime system. An adaptive runtime system can migrate tasks and data from one processor to any other processor available to the job. This ability can solve many challenges that upcoming supercomputers face - application load imbalance across processors, high fault rates, power and energy constraints, and thermal variations. These challenges are often contradictory in terms of their requirements. For example, applying power and temperature constraints can compromise performance and lead to load imbalance across processors. We strive to achieve a healthy balance where we try to maximize performance in presence of the known as well as contingent constraints and events.

In Figure 1, we show a unified diagram of several important components of a data center, their functions and interactions with each other in order to address the challenges of power, reliability, and performance. Currently, data center users are primarily concerned about the performance of their job. In future, however, power consumption of their jobs may become a major concern. On the other hand, the data center administrators have different and more complex concerns - while they want to guarantee good performance to individual jobs, they need to ensure that the total power consumption of the data center does not exceed its allocated budget and that the job throughput of the data center remains high despite node failures and thermal variations. We achieve the objectives of both users and system administrators by allowing dynamic interaction between the system resource manager/scheduler and the job runtime system. While the job scheduler strives to allocate the system resources optimally to the jobs based on their power and performance characteristics, the job runtime system implements the decision of the job scheduler by being malleable to shrink or expand itself to the nodes assigned by the scheduler and by doing dynamic load balancing whenever beneficial. Furthermore, the runtime system can turn on/off or reconfigure various hardware components without impacting application performance, if adequate hardware control is provided by vendors. Our evaluations demonstrate that these runtime capabilities result in greater power efficiency for common HPC applications.

An Adaptive Runtime System for HPC

An adaptive runtime is an essential component of a system optimized for power efficiency, reliability and performance. Adaptive runtime systems enable dynamic collection of performance data, dynamic task migration (load balancing), temperature restraint and power capping with optimal performance.

CHARM++ is a C++ based parallel programming framework supported by an adaptive runtime system, which enhances user productivity and allows programs to run portably from small multicore computers (laptops, phones) to the

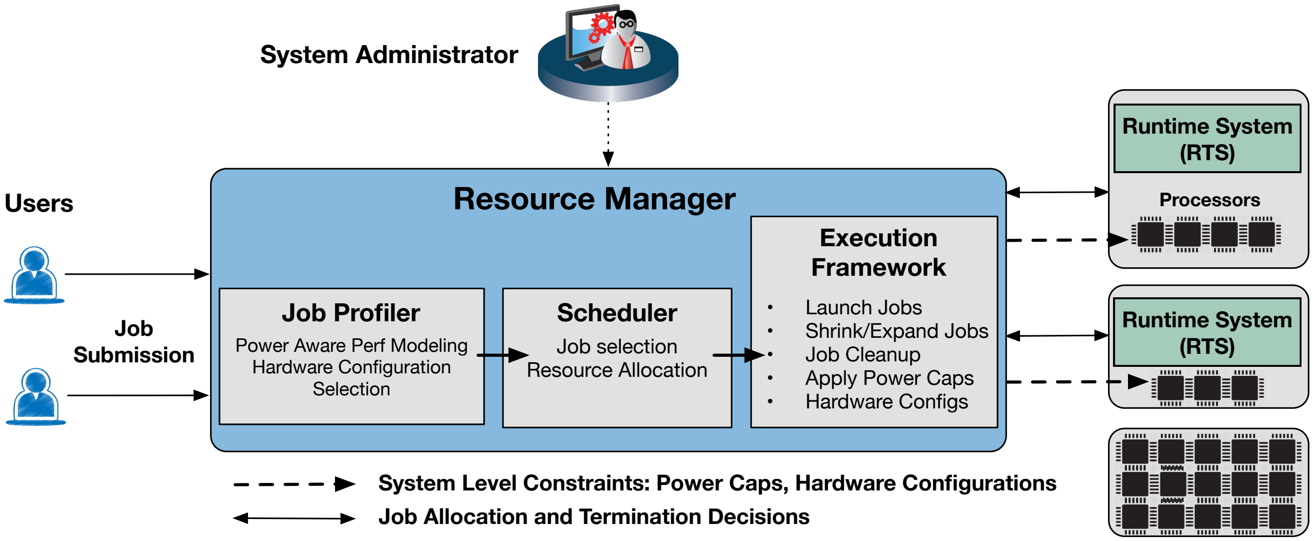

Figure 1: Figure shows overall system design with two major components interacting with each other: Resource Manager and the Runtime System.

largest supercomputers [6]. It enables users to easily expose and express much of the parallelism in their algorithms while automating many of the requirements for high performance and scalability. CHARM++ has been in production use since 2001 and it has thousands of users across a wide variety of computing disciplines with multiple large scale applications including: NAMD for molecular dynamics, ChaNGa for cosmology, OpenAtom for quantum chemistry simulations, and many others [6].

CHARM++ has three main attributes: over-decomposition, asynchronous message-driven execution, and migratability. Over-decomposition entails having the programmer divide the computation in an application into small work and data units so that there are many more such units than the number of processors. Message-driven execution involves scheduling work units based on when a message is received for them. Migratability refers to the ability to move data and work units between processors. These attributes enable the CHARM++ adaptive runtime system to provide many useful features including dynamic load balancing, fault tolerance, and job malleability (shrink-expand the number of processors the application is running on).

CHARM++ collects information about the application and the system in a distributed database including loads of processors, loads of each object, communication patterns, and core temperatures. When the number of processors are large, centralized data collection becomes a performance bottleneck. Therefore, data collection and decision making are done in a hierarchical fashion. This information is used by different modules of the adaptive runtime to make decisions such as improving load balance, handling faults, and enforcing power constraints.

Load balancing: CHARM++ uses a measurement-based mechanism for load balancing. It relies on a heuristic known as *the principle of persistence*, which states that for over-decomposed iterative applications, the computation load and the communication pattern of tasks or objects tend to persist over time. It uses the load statistics of the application code collected by the runtime system. This has the advantage that it provides an automatic, application independent way of obtaining the load statistics without any input from the user. The system also allows the user to specify predicted loads overriding system predictions. Using the load statistics, CHARM++ executes a chosen load balancing strategy to determine a mapping of objects to processors and then carries out migrations based on this mapping. CHARM++ consists of a suite of load balancers including several centralized, distributed and hierarchical strategies. The runtime system can also automate the decision of when to call the load balancer [7]. It can use the instrumented load information to predict the future load and make load balancing decisions. It automatically triggers load balancing when imbalance is detected and when the benefit of load balancing is estimated to be more than its overhead.

Fault tolerance: The CHARM++ runtime system implements both *proactive* and *reactive* strategies for reliability [8]. In the proactive techniques, the runtime system evacuates all objects from a node that a monitoring system predicts is going to crash soon. Since failure prediction is not completely accurate, the reactive techniques recover the information lost after a failure brings down one node of the system. Those latter strategies are mostly based on checkpoint/restart. Therefore, the global application's state is routinely stored and recovery implies retrieving a prior

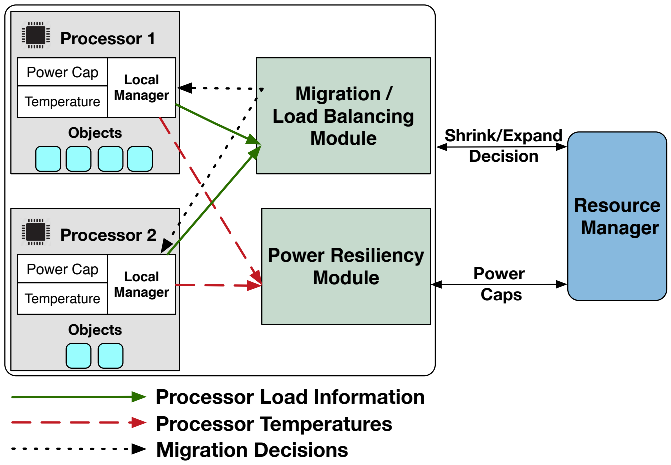

Figure 2: Various components of the adaptive runtime system and their interaction with resource manager.

global state.

Shrink-expand: The migratability of CHARM++ objects enables a unique ability called *job malleability*; during runtime, a job can shrink (decrease) or expand (increase) the number of nodes it is running on. This feature does not require any additional code from the application developer [9]. Shrink or expand operations can be triggered by an external command or it could be an internal decision made by the runtime. During a shrink operation, the runtime system reduces the number of processors that the application is running on. First, it moves the objects away from the processors that are not going to be used anymore. The unused processors can then be returned back to the resource manager. For an expand operation, the runtime launches new processes on the additional processors that the resource manager has allocated and distributes objects from current processors to the newly allocated processors. In addition to moving objects, the runtime system must adjust its distributed data structures, including spanning trees and location managers.

Figure 2 shows the internal components and functioning of the CHARM++ Runtime System (RTS). There are three important components of the RTS - Local Manager (LM), Load Balancing Module (LBM), and Power Resiliency Module (PRM). Each processor has an LM that is responsible for managing the objects residing on that processor and for interacting with other components of the RTS. The LM of each processor periodically sends its total compute load and compute load of each of its objects to the LBM, and the CPU temperature is sent to the PRM. LBM makes the load balancing decisions and it also redistributes load in response to shrink-expand commands from the resource manager. Object migration decisions are communicated to the respective LM by the LBM. PRM, on the other hand, is responsible for ensuring that the CPU temperatures remain below the job specific temperature threshold (determination of this threshold is shown in Figure 5). The PRM controls CPU temperature by adjusting the power cap of the CPU. When a processor's temperature is above the threshold, its power cap is lowered. And when the temperature is well below the desired threshold, the corresponding power cap is increased while ensuring that the total power of the job remains below the power budget allocated to the job (the determination of this budget is described in the next section). Jobs may not have administrator rights to constrain the power consumption of their CPUs. Therefore, the new power caps are communicated to the resource manager which applies them to each CPU.

THROUGHPUT MAXIMIZATION UNDER A POWER BUDGET

The U.S. Department of Energy has set a power limit of 20MW for an exascale supercomputer that it is willing to purchase and deploy. We demonstrate that power aware resource management along with malleable jobs can lead to significant improvements in job throughput of a data center running under a power budget.

Recent advances in processor hardware design allow users to control the amount of power consumed by the processor using software with Running Average Power Limit (RAPL) driver. Processors can be power capped to run below their Thermal Design Power (TDP) value, where TDP is the maximum amount of power a processor can consume. The maximum number of nodes in a data center with a power budget is determined by the TDP of the nodes. Power capping makes it possible to control the power consumption of nodes and thus have additional nodes while remaining within the power budget of the data center. This is called an *overprovisioned* system [10]. Earlier research shows that an increase in the power allocated to a processor does not yield a proportional increase in job's performance [1]. Different jobs react differently to an increase in power allocated to the CPU. The idiosyncrasies in jobs performance based on allocated CPU power, points to the possibility of running different applications at different power levels. Overprovisioned systems can significantly improve performance of applications that are not sensitive to CPU power by capping CPU powers to values well below their TDP and adding more nodes to get benefits from strong scaling. The Power Aware Resource Manager (PARM) [1] leverages this capability by optimally distributing the available resources to the jobs - the total power budget of the data center and the compute nodes.

The response of an application to CPU power can be captured by its power-aware speedup [11]. The *power-aware speedup* is the ratio of the execution time of a job running on a CPU capped at a certain power level compared to the execution time of the same job when running on the lowest allowed power level allowed by the CPU [1]. A higher value for power-aware speedup implies that the application is sensitive to changes in the amount of power allocated to the CPU.

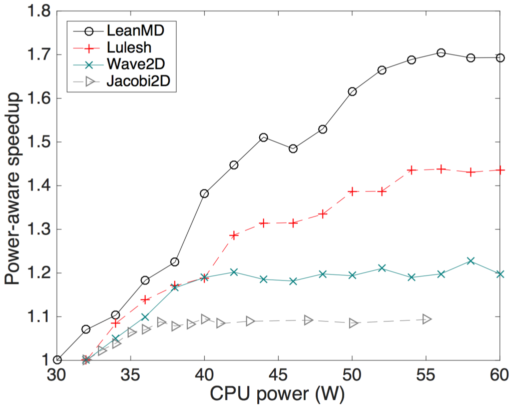

Figure 3: Power-aware speedups of four applications running on 20 nodes. The applications vary from being CPU intensive to memory intensive.

Figure 3 shows *power-aware speedups* of four HPC applications having different characteristics under different CPU power caps [1] (The minor deviations from the monotonic behavior in the Figure are likely due to external factors such as OS jitter, network delays, etc.). LeanMD, which is a molecular dynamics application has the highest power-aware speedup since is the most CPU intensive one. Whereas Jacobi2D, which is a stencil application, has the lowest since it is memory intensive. PARM makes scheduling decisions by selecting jobs and their resource configurations (power budget and compute nodes) such that the total power-aware speedup of running jobs is maximized.

PARM is an essential part of the overall system we propose in this work. It dynamically interacts with the adaptive runtime system of jobs, the system hardware, the user and the system administrator to perform several critical tasks (Figure 1). There are three important components of PARM:

- Job Profiler: Before a job is added to the scheduler queue, it is profiled to develop a power-aware strong scaling model that is used to calculate the power aware speedups. This profiling mechanism has negligible overhead as it is sufficient to run the application for a few iterations to get the necessary data points.

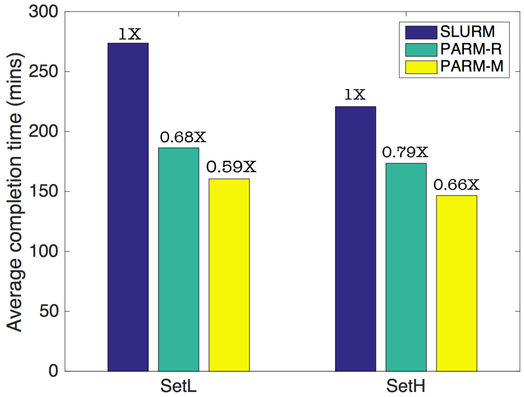

Figure 4: Comparison of average completion time of jobs with SLURM and PARM, in Rigid(R) and Malleable(M) variants. SetL has jobs that have low sensitivity to CPU power and SetH has jobs that have high sensitivity.

- Scheduler: PARM implements its resource allocation optimization strategy as an Integer Linear Program (ILP) with the objective of maximizing power-aware speedup of running jobs under power constraints. Whenever a new job arrives or a running job terminates, PARM's scheduler is triggered, and re-optimizes scheduling and resource allocation decisions. PARM's ILP is fast enough to run frequently with negligible overhead [1].
- Execution Framework: This component implements the scheduler decisions by launching jobs, sending shrink/expand decisions to the runtime system of the jobs, and by applying power caps on compute nodes. Job runtime systems interact with the execution framework to convey job termination, completion of shrink/expand operation and any changes to CPU power caps as determined by PRM module of the runtime system.

Figure 4 shows the benefits of using PARM as compared to power-unaware SLURM which is an open source resource manager used in many supercomputers. Two versions of PARM are compared - PARM-Rigid and PARM-Malleable. In PARM-Rigid, node allocation decision to any job is rigid, that is it cannot be changed once the job starts running. PARM-Malleable, on the other hand, has an additional degree of freedom that allows it to change the nodes allocated to a running job which is made possible by the shrink/expand feature of CHARM++ . The number at the top of each bar in Figure 4 represents average completion time as a percentage of the average completion time using SLURM scheduler. PARM-Malleable was able to reduce average completion time of jobs by up to 41%. The effects of reducing the CPU power on performance are large for jobs with high power sensitivity (SetH). Therefore, the benefits of adding more nodes at the cost of reducing CPU power are small with SetH. Consequently, PARM gives less performance benefit over SLURM with SetH as compared to SetL.

Improving Reliability Through Temperature Restraint and Load Balancing

Checkpoint/restart is the most popular mechanism to provide fault tolerance in HPC. The total execution time T of an application, on an unreliable system, is given by the equation:

$$T = T_{solve} + T_{checkpoint} + T_{recover} + T_{restart}$$

where T_{solve} represents the total effort required to solve the problem; $T_{checkpoint}$ accumulates all the time spent on saving the checkpoints of the system; $T_{recover}$ stands for the total work that is lost and must be recovered as a result of failures in the system; $T_{restart}$ is usually constant and represents the amount of time required to resume execution after a crash. A system using checkpoint/restart has to choose an appropriate checkpoint period (denoted by τ). There is a delicate balance in the value of τ. A long value of τ (low checkpoint frequency) decreases $T_{checkpoint}$, but may increase $T_{recover}$. Conversely, a short value of τ (high checkpoint frequency) means a reduced $T_{recover}$, but may enlarge $T_{checkpoint}$. The optimum value of τ strongly depends on the mean-time-between-failures (MTBF) of the system.

The MTBF of an electronic component is directly affected by its temperature. That relation is usually exponential and there is some experimental evidence that a $10°C$ increase on a processor's temperature decreases its MTBF in half [3]. Therefore, the reliability of a system can be controlled by restraining the temperature of its components. The cooler the system runs, the more reliable it is, but the slower it runs. That is because temperature constraints are realized by restraining the power of the CPU. This interplay between power management and reliability has been studied in other context [12]. The runtime allows each core to work at the maximum possible power as long as it is within the maximum temperature threshold. If any of the cores goes above the maximum temperature threshold, their power is further reduced causing its temperature to fall. However, this can cause a performance degradation for tightly coupled applications due to thermal variations. The LBM will automatically detect any load imbalance and will make the load balancing decisions [2].

The runtime system must strike a balance in the temperature at which each component should be restrained. Moreover, that balance depends on the application. Different codes generate different thermal profiles on the system at different stages of the application. Some codes are more compute intensive and tend to heat up the processors more quickly. Appropriate, application-based temperature thresholds are stored as part of the Job Profiler in Figure 1. In the end, the runtime system aims at reducing the total execution time of an application, considering the MTBF of the system and subject to the power limitations [3].

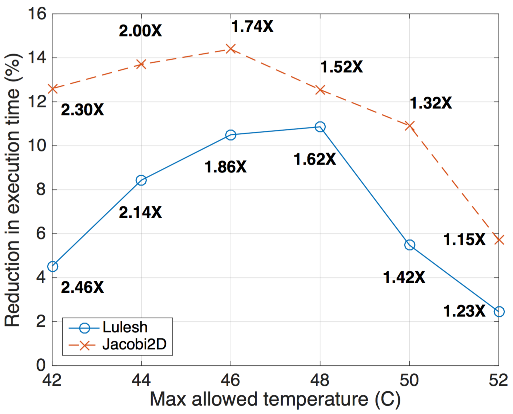

Figure 5: Reduction in execution time and change in MTBF for different temperature thresholds

Figure 5 shows percentage reduction in execution time after constraining core temperatures to different thresholds for two different applications. The reduction in execution time shown in Figure 5 is calculated compared to the baseline case where processor temperature is not constrained. Figure 5 also shows the ratio of MTBF for the machine using our scheme relative to the baseline case where core temperatures are not constrained. For example, by restraining core temperatures to $42°C$ in case of Jacobi2D, the MTBF for the machine increased 2.3 times while the execution time reduced by 12% compared to the baseline case where core temperatures are not constrained. The inverted U shape of both the curves strongly suggests a trade-off between reliability (MTBF) and the slowdown induced by the temperature restraint.

The resource manager sends the PRM of the runtime system (Figure 2) the upper bounds of the temperatures that honor the power envelope of the system. Those temperature values are used as input to an internal *resilience* component in PRM and they will be changed according to the algorithms that optimize performance and consider the MTBF of the system and the characteristics of the application running. The output will be propagated to further components in PRM that will later consolidate the final power limits and they will be communicated back to the resource manager. A dynamic runtime system is fundamental in controlling the reliability of the system and honoring the power envelope at the same time. Since thermal variations are dynamic, a reactive runtime system efficiently responds to those changes and provides a healthy balance between performance and reliability in the system. In addition, a broader range of reliability concerns can be addressed by this scheme. Soft errors (e.g. bit flips caused by high-energy particles) can also be addressed in tandem with hard errors [13]. Circuits using near-threshold voltage will introduce a more complex scenario with higher performance variability and higher transient-failure rate.

Dynamic Configuration of System Components

The runtime system can take advantage of the currently available hardware "knobs" for controlling power such as frequency scaling and power capping. However, greater power savings are possible if there is more runtime control over hardware components. We demonstrate, via cycle accurate simulations, that the runtime system can turn off or reconfigure many components without significant performance penalty based on the properties of the running HPC application.

HPC systems should ideally be *energy proportional*; the hardware components should consume power and energy only when their functionality is being used. However, network links are always "on", independent of their utilization. In addition, processor caches consume large amounts of power, even when they are not improving the performance of the running application. We propose a runtime system approach that can save this wasted energy by dynamically re-configuring the hardware based on the application needs.

Caches consume up to 40% of a processor's power [4]. A large fraction of cache power consumption can be saved by turning off some cache banks in cases where the application performance would not be degraded. Many common HPC applications cannot take advantage of the caches effectively. For example, molecular dynamics applications typically have small working data sets and do not need the large last level caches(LLC). On the other hand, grid-based physical simulation applications typically have very large data sets that do not fit in caches and the data reuse in cache is minimal. However, the hardware is not able to predict the application behavior. Therefore, we propose a runtime system approach where the runtime uses profiling data to reconfigure the cache to save power without significant performance loss. Using a set of representative HPC applications, our previous study demonstrates that on average, 67% of cache energy can be saved with only 2.4% performance penalty [4].

A similar approach applies to HPC networks as well. Networks consume up to 30% of system power even when there is no communication since the links are always on. Our previous study demonstrates that typical HPC applications do not use a large fraction of the links in most of their execution time [5]. The reason is that HPC topologies such as Dragonfly are designed to handle the most challenging communication patterns such as all-to-alls in FFT modules. However, typical applications have sparse communication patterns such as nearest neighbor that cannot exploit the massive number of links in HPC networks like Dragonfly. We propose using the runtime system to turn the links on and off adaptively. This hardware configuration case is harder to handle since the usage of network links can be impacted by features such as adaptive routing. Therefore, the runtime system should handle different hardware designs based on their exact specification. Our results demonstrate that up to 80% of the power consumption of network links can be saved using our adaptive runtime strategy [5].

Final Remarks

Important challenges, such as power, reliability, and thermal variations, loom in the future of supercomputing. Addressing these concerns is imperative to harness the next generation of high performance machines. We propose a unified system design with a smart runtime system which interacts with the system resource manager.

The combined system offers several important features. First, it honors the power constraints by wisely scheduling jobs and re-allocating their resources when utilization changes. Second, it controls the reliability by a temperature-aware module that cools down the system to an application-based optimal level. Third, it can re-configure the hardware via the runtime without sacrificing performance.

About the Authors

Bilge Acun is a Ph.D. student in the Department of Computer Science at University of Illinois at Urbana-Champaign. Her research interests include power-aware software design, malleability and variability in large-scale applications. *acun2@illinois.edu.*

Akhil Langer is a Software Engineer at Intel Corporation. His research interests include scalable distributed algorithms and power and communication optimizations in HPC. He received his Ph.D. in Computer Science from University of Illinois at Urbana-Champaign. *akhil.langer@intel.com.*

Esteban Meneses is an Assistant Professor in the School of Computing at the Costa Rica Institute of Technology. His research interests include reliability in HPC systems, parallel-objects application design, and accelerator programming. He received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign. *esmeneses@tec.ac.cr.*

Harshitha Menon is a Ph.D. student in the Department of Computer Science at University of Illinois Urbana Champaign. Her research interests include scalable load balancing algorithms and adaptive runtime techniques to improve the performance of large-scale applications. *gplkrsh2@illinois.edu.*

Osman Sarood is a Software Engineer at Yelp Inc. His research interests include performance optimization for parallel and distributed computing under power and energy constraints. He received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign. *osarood@yelp.com*.

Ehsan Totoni is a Research Scientist at Intel Labs. His research interests include programming systems for HPC and big data analytics. He received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign. *ehsan.totoni@intel.com*.

Laxmikant V. Kale is a professor of Computer Science Department at the University of Illinois at Urbana-Champaign. His research interests include various aspects of parallel computing, with a focus on enhancing performance and productivity via adaptive runtime systems. *kale@illinois.edu*.

References

[1] O. Sarood, A. Langer, A. Gupta, and L. V. Kale, "Maximizing throughput of overprovisioned hpc data centers under a strict power budget," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. New York, NY, USA: ACM, 2014.

[2] O. Sarood, P. Miller, E. Totoni, and L. V. Kale, "'Cool' load balancing for high performance computing data centers," in *IEEE Transactions on Computer - SI (Energy Efficient Computing)*, September 2012.

[3] O. Sarood, E. Meneses, and L. V. Kale, "A 'cool' way of improving the reliability of HPC machines," in *Proceedings of The International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, CO, USA, November 2013.

[4] E. Totoni, J. Torrellas, and L. V. Kale, "Using an adaptive hpc runtime system to reconfigure the cache hierarchy," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. New York, NY, USA: ACM, 2014.

[5] E. Totoni, N. Jain, and L. Kale, "Power management of extreme-scale networks with on/off links in runtime systems," *ACM Transactions on Parallel Computing*, 2014.

[6] B. Acun, A. Gupta, N. Jain, A. Langer, H. Menon, E. Mikida, X. Ni, M. Robson, Y. Sun, E. Totoni, L. Wesolowski, and L. Kale, "Parallel programming with migratable objects: Charm++ in practice," ser. SC, 2014.

[7] H. Menon, N. Jain, G. Zheng, and L. V. Kalé, "Automated load balancing invocation based on application characteristics," in *IEEE Cluster 12*, Beijing, China, September 2012.

[8] E. Meneses, X. Ni, G. Zheng, C. Mendes, and L. Kale, "Using migratable objects to enhance fault tolerance schemes in supercomputers," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 26, no. 7, pp. 2061–2074, July 2015.

[9] A. Gupta, B. Acun, O. Sarood, and L. V. Kale, "Towards realizing the potential of malleable parallel jobs," in *Proceedings of the IEEE International Conference on High Performance Computing*, ser. HiPC '14, Goa, India, December 2014.

[10] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski, "Exploring hardware overprovisioning in power-constrained, high performance computing," in *Proceedings of the 27th international ACM conference on International conference on supercomputing*. ACM, 2013, pp. 173–182.

[11] R. Ge and K. W. Cameron, "Power-aware speedup," in *2007 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2007, pp. 1–10.

[12] R. Melhem, D. Mossé, and E. M. Elnozahy, "The interplay of power management and fault recovery in real-time systems," *IEEE Trans. Comput.*, vol. 53, no. 2, pp. 217–231, Feb. 2004.

[13] X. Ni, E. Meneses, N. Jain, and L. V. Kale, "Acr: Automatic checkpoint/restart for soft and hard error protection," in *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. IEEE Computer Society, Nov. 2013.