Matchmaking: A Solution Path for Workstealing at Scale for Irregular Applications

Hrushit Parikh Vinit Deodhar Ada Gavrilovska Santosh Pande

Georgia Institute of Technology

{parikhhrushit,vinit}@gatech.edu {ada, santosh}@cc.gatech.edu

Abstract

Many classes of high-performance applications and combinatorial problems exhibit large degree of runtime variability. One approach to achieving high machine efficiency and balanced resource use is to over decompose the problem on fine-grained tasks that are then dynamically balanced across the system by approaches such as workstealing. Existing work stealing techniques for such irregular applications, running on large clusters, exhibit high overheads due to potential untimely interruption of busy nodes, excessive communication messages and delays experienced by idle nodes in finding work due to repeated failed steals. In particular, on very large scale clusters, the problem of matching work generation and work consumption is extremely challenging. At the heart of the problem is the dilemma: idle workers do not know where to look for work and the work producers do not know where to send the generated extra work. We contend that the fundamental problem of distributed work-stealing is not one of rapid dissemination of availability of work but one of rapidly bringing together work producers and consumers. In response, we develop a workstealing-based algorithm that performs timely, lightweight and highly efficient *matchmaking* between work producers and consumers. We validate these claims via an implementation of a matchmaking-based scheduler for Charm++, and evaluate using representative benchmarks running on up to 8K cores. Results show that our scheduler is able to outperform other load balancers and distributed work stealing schedulers, and to achieve scale beyond what is possible with current approaches.

1. Introduction

Many emerging classes of large-scale applications, such as natural language processing, bioinformatics, data mining and analytics or data-intensive scientific discovery, exhibit significant complexity and variability that prevent their efficient static parallelization and decomposition [12, 14]. The execution of such workloads is commonly supported by runtimes that provide for problem overdecomposition via fine grained parallelism, and dynamic work migration to balance the load across the HPC machine resources and to maximize application performance and machine efficiency.

Existing state-of-the-art re-balancing techniques rely on (1) hierarchical dynamic load balancers [16], triggered with some periodicity, that exploit locality and topology-awareness to achieve low work migration costs, or on (2) work stealing algorithms [2, 8, 13], in which idle nodes randomly probe other nodes to find a victim with excess work and steal its tasks. Both type of approaches have been shown effective for certain scientific workloads. However, they both suffer from limited scalability and effectiveness for highly dynamic ”divide and conquer” and combinatorial search problems. For such applications, the challenge is that (i) their irregular behavior does not lend itself to the ‘persistence’ property that makes hierarchical load balancers effective [15], while at the same time (ii) the large scale of the machine renders workstealing ineffective due to large overheads incurred. These overheads stem from the fact that ’thieves’ go around the cluster randomly probing nodes to find a ’victim’. Further, due to the scale of the cluster work is sparsely distributed across nodes which further compounds the above problem. This leads to (i) excessive messages in system, (ii) long starvation periods for ‘thieves’ due to multiple failed steals and network latency associated with each steal and (iii) disruptions of busy nodes from repeated steal requests that cannot be satisfied. Thus, idle nodes, i.e., thieves, are starved for a long time, random nodes with not enough (excess) work are interrupted, and overloaded nodes remain overloaded. Variants of the basic work stealing algorithm on large-scale clusters employ schemes in which nodes have some state information about other processing nodes of the cluster [3, 4] or place an upper cap on number of unsuccessful steal requests after which thieves enter a quiescent state and wait for work [1]. These techniques may limit the work stealing overheads, but they fall short on improving its effectiveness for irregular workloads.

In response to these observations, we posit that key limitation of existing approaches for scalability and improved performance (of highly irregular workloads) is their inability to *rapidly* and *with low overheads* “match” work to idle resources.

To address this problem, we argue that workstealing mechanisms should be augmented with nodes performing designated *matchmaking* functionality so as to facilitate matching of work producers (i.e., overloaded nodes with excess work) to work consumers (i.e., idle nodes, or thieves). In one extreme configuration, a single, centralized matchmaker can play the role of an active, continuous re-balancer, orchestrating the work shifting across the parallel machine . The need to randomly probe nodes or *periodically* send and receive load information to make steal decisions [5] is alleviated. We show in this paper (using typical irregular benchmarks such as UTS and Hamiltonian Path Problem executing on up to 8k cores) that even a single matchmaker is sufficient to significantly improve the application scalability and performance and increase the machine efficiency in terms of required resources. We demonstrate the lightweight nature of our scheduler which makes such scaling possible.

More generally, however, matchmaking can be (i) carried out by multiple matchmakers, thus addressing scalability concerns, and (ii) nodes can be mapped to matchmakers based on spatial or temporal metrics. For instance, a matchmaker may consider only nearby nodes when establishing match candidates to limit interconnect use and data movement requirements. Further, this scheme can exploit temporal locality of requests- producer/consumer requests anticipated for a future time window W1 are sent to a particular matchmaker and requests anticipated for window W2 sent to a different matchmaker. Such future information can be derived from a compiler infrastructure as explained in our prior work[11]. This would yield more 'precise' and more 'timely' matches and minimize thief idleness. As a result, matchmakers present a promising path forward to address performance, scalability and efficiency for large-scale deployments of irregular applications.

In summary, this work makes the following contributions:

- Matchmaker-based scheduling – we extend the scheduling taxonomy with a novel construct – matchmaker – that addresses the scalability and efficiency limitations of work stealing solutions for highly irregular applications
- Charm++ prototype – we implement a prototype of a matchmaking scheduler in the Charm++ runtime
- Large-scale evaluation – we compare the benefits of the proposed matchmaking-based approach to existing work stealing and load balancing solutions, using typical benchmarks representative of irregular HPC workloads, like Unbalanced Tree Search (UTS) and Hamiltonian Path Problem, when running at scale on the TACC Stampede machine.
- Scalability and efficiency gains – experimental analysis demonstrate that matchmaker-based workstealing can achieve significantly better scalability and reduced overheads for these classes of applications- improvements of up to 50% in application execution time, and reduction in interconnect messages by up to 2x.

The overall outcome is application speedup, improved machine balance and overall efficiency of how the machine resources are utilized.

2. Background and Motivation

We first motivate the work by discussing the relevance of irregular workloads and the challenges they exhibit when scheduled at scales. We also provide background information on Charm++, which provides the implementation and experimentation context for our work.

2.1 Irregular Applications

High-performance applications have evolved beyond purely traditional scientific simulations, that exhibit high degree of locality and density, and lend themselves well for parallelization on networked supercomputing machines. Emerging classes of applications, including natural language processing, complex network analysis, data mining and analytics, bioinformatics, and novel methods for data-intensive scientific discovery exhibit high degree of irregularity both in their data access patterns, as well as in the distribution of computational requirements over space and time. This presents a challenge for traditional HPC stacks, which are not optimized for workloads with randomized behaviors and sparse work distribution, and have led to the development of a number of novel hardware and software architectures, runtimes and programming models. Combinatorial problems such as N-queens, Hamilton Paths or Circuits or TSP result in highly unpredictable, bursty dynamic load which can result in huge imbalances on large scale clusters. Moreover, with the growing number of cores per node, the memory collisions in caches, network cards etc. can result in huge variations in load latencies and thus, can result in dynamic imbalances that are architecturally induced. Such imbalances are highly unpredictable and irregular and must be fixed only via work stealing/scheduling.

Our work is specifically targeting the problems of scalable execution of such irregular combinatorial problems. At the core of these applications lie computational patterns that are represented as analyses of sparse graphs, unstructured grid or unbalanced tree traversals, combinatorial optimization and recursive parallel codes. These observations motivate the selection of benchmarks used in the experimental evaluations presented in this paper, and have already been established in the past as representative of such irregular workloads.

2.2 Scheduling Frameworks

Irregular problems are particularly well suited for runtimes which support fine-grained parallelism and high-degree of problem decomposition, such as Unified Parallel C (UPC), X10 and Charm++. UPC and X10 being PGAS languages have access and compile time knowledge of tasks and their locations. For instance, UPC provides a load balancing API where the application developer has to manage the private and shared tasks and the workstealing parameters. The runtime performs workstealing based on these parameters. However, workloads like UTS are highly irregular and dynamic and cannot be estimated by the developer. Such applications require the run-time to perform independent task scheduling/balancing. Charm++ is a framework which meets this criteria as its runtime is agnostic to any programmer or compiler provided hints. We believe that this setting solves a different and challenging problem and hence we focus our work on the Charm++ environment.

Charm++ also provides an abstraction over MPI, allowing programmers to focus on computation while the framework performs MPI communication and synchronization. Charm++ is completely machine independent and known to run on Blue Gene/P, Blue Gene/L, Cray XT3, Infiniband clusters such as Stampede, Unix workstations, etc. More importantly charm++ is very stable and extremely scalable (130,000+ cores) as shown from several previous publications[6, 7, 16].

3. Matchmaker-based Workstealing

3.1 Basic Idea

The matchmaker-based workstealing mechanism proposed in this paper, follows the traditional work stealing approach in which scheduling is reactive – nodes begin looking for work when they are idle. However, instead of random probing they contact a matchmaker and their request is mapped to producer requests in a quick timely manner. This minimizes the wait time of idle cores. Further, producer and consumer requests are sent on a per demand basis and not periodically. Thus the total messages are a function of the imbalance in the system i.e better load balancing further reduces messages exchanged. This puts a bound on total messages. Overall, like traditional workstealing, scheduling in our scheme is on demand and driven by idle thieves. We introduce the notion of matchmakers to mitigate issues faced by workstealing for irregular workloads – namely excessive failed steals, network latency penalty and unnecessary interruptions of nodes which are busy but not overloaded. Instead of thieves randomly looking for work across the cluster and failing, we direct a thief to a specific location (i.e., to a matchmaker) where they can be quickly paired with producers who are looking to offload work in the same temporal window.

3.2 Design Considerations

Matchmakers store information from incoming load shift and steal requests, in queues of pending producer and steal requests respectively and use this information to determine appropriate responses. Requests are matched in FIFO manner, i.e, an incoming producer request is matched to first consumer request in consumer queue and vice verse. In this manner, matchmakers maintain a pseudo-global state of the load in the cluster. A matchmaker node is not exclusively a matchmaker. It can become a consumer, if it is idle and subsequently a busy node if it receives work. However, in our algorithm we do not allow them to become a producer (even if they have excess work) to prevent a matchmaker from bouncing work to another matchmaker who may also become overloaded and so on.

In general, producers simply announce to the matchmakers ”work availability” (i.e., requests to ‘shift’ load from a node), and consumers announce willingness to ‘steal’ someone else’s work (i.e., ‘steal’ requests). The matchmaker matches these requests, and actual load migration happens via direct consumer-producer interaction. However, if the actual load migration resource requirements are relatively small (less than 1MB in our experiments), then work migration can be *inlined* – producer nodes can directly send work (tasks) instead of ”work availability” message to matchmaker. Matchmaker then forwards work to any incoming consumer requests. This eliminates the need for consumer-producer interaction and reduces messages.

A workstealing runtime can employ one or more matchmakers. Matchmakers can be statically determined, or can be dynamically selected in an epoch-based manner based on current load distribution throughout the platform. In case of multiple matchmakers, work among matchmakers is reconciled as follows: A matchmaker first tries to fulfill any incoming producer/consumer request from its own consumer/producer queue entries. If that fails, it probes the next matchmaker and so on until either the request is fulfilled or every other matchmaker is probed. In latter case synopsis is complete and original matchmaker enqueues the request in its queue.

An important aspect of the matchmaking scheduling framework is that it is sufficiently flexible to allow different spatial or temporal metrics to be incorporated in the scheduler, in a scalable manner. For instance, a multi-matchmaker instance of this framework, where matchmakers are strategically placed based on cluster topology can lead to hierarchical distribution of matchmakers where load imbalances are more quickly resolved in a localized manner.

More interestingly, for workloads where future load supply or demand can be predicted at run-time [11] or derived from application hints, temporally proximate producer/consumer requests can be sent to a designated matchmaker. In this way, the matchmakers are responsible for anticipated load balancing for specific, smaller temporal windows, thus resulting in ability to scale further in a distributed manner.

4. Experimental Evaluation

In this section we present the select results from the experimental evaluation of the matchmaker-based scheduling. The scheduling algorithm is developed in the Converse run-time layer of Charm++ and primarily written in C. The results presented here compare the matchmaking scheduler to default work-stealing scheduler part of Charm++ (marked `Default` in all graphs), and a conventional centralized load balancer (marked `Cent` in all graphs) in which every node periodically sends its load information to a central node which then makes load shifting decisions [17]. This is done to demonstrate that bottlenecks which arise in such a centralized scheme do not arise in our scheme even when using one matchmaker. The goals of the evaluation are to demonstrate (i) the benefits that use of matchmaking can deliver in improving the scale and performance of irregular workloads, (ii) its inherent light-weight nature and reduced overheads in terms of machine (CPU and interconnects) resource requirements, and (iii) the resulting improvements in overall efficiency with which these types of workloads can be executed at scale.

4.1 Benchmarks and Platform

The evaluation results shown in this paper are based on the following benchmarks which have been widely accepted [7, 10] as representatives of the types of irregular workloads targeted by our work: Unbalanced Tree Search (UTS) and Hamiltonian Path Problem (HPP).

All experiments are performed on the Stampede supercomputing cluster at Texas Advanced Computing Center (TACC). The cluster consists of 6400 Nodes each with 2 Intel Xeon E5 8 core processors to form 16 cores/node. Each node has 32 GB DDR3 memory. The nodes are connected by 56Gb/s FDR InfiniBand interconnect in 2-level fat-tree topology. All code is compiled for CentOS release 6.6 64 bit and MPI 1.5 using Intel ICC compiler version 13.1.0.

4.2 Baseline

The work stealing scheduler in charm++ implements a load stealing algorithm in which thieves randomly select a victim processing node and request task objects from the runtime of that node. If the victim is overloaded, then its runtime sends half the task objects in its task queue, otherwise it sends a negative ack, and the thief probes another node and so on until it gets work. This ”steal-half random workstealing” algorithm has previously been shown to be scalable upto 8K cores [9, 10]. In particular, the charm++ implementation of this algorithm has been shown to scale to 16K cores for irregular applications [7] and hence, is the baseline in our evaluations.

4.3 Performance Measurements

We demonstrate (i)the benefits of the matchmaking scheduler for achieving improvements in application scalability and performance, using execution time as a metric, and (ii)the lightweight nature of this scheduler in terms of imposed system loads and processing requirements.

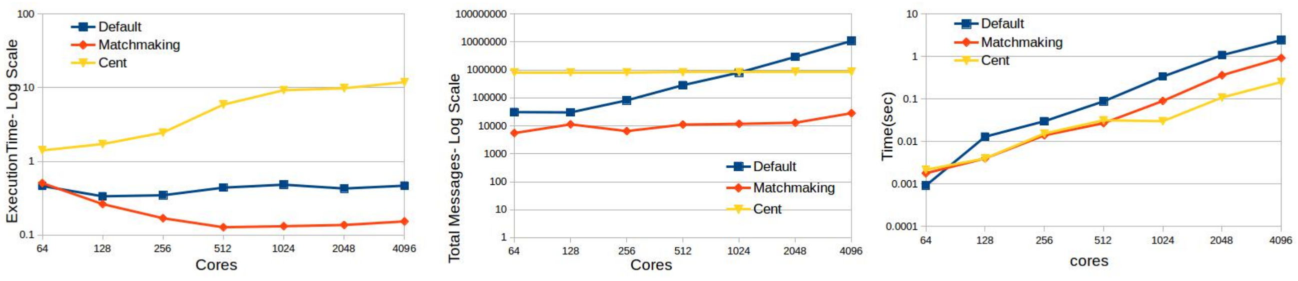
Unbalanced Tree Search (UTS)

We discuss the experimental results for UTS, using the T1L dataset (Figure 1), which represents search of an unbalanced tree containing about 100 million nodes. In Figures 1a and 1b we observe that matchmaking delivers 21%-72% performance benefits compared to default work-stealing when scaling the execution of T1L from 64-4096 cores, with 62%-99% reduction in total messages. We observe that the default workstealing scheduler stops scaling at merely 128 cores, whereas with matchmaking, the execution time continues improving significantly up to 512 cores. Furthermore, beyond the scaling limits, the workstealing scheduler exhibits drastic degradation (notice the log scale of the y axis in Figure 1.a), while the impact on performance with matchmaking is very gradual. This implies that matchmaking provides additional benefits by being less forgiving if resources are not ‘right-sized’ to a particular workload characteristic. This is highly important for dynamic, irregular workloads, where workload requirements are hard to predict. In addition, the matchmaking scheduler also delivers improved performance and supports larger-scaled executions compared to the centralized scheduler, with an added benefit of achieving more balanced machine use. We observed similar trends for the larger UTS problem size (T1XL- Unbalanced tree with 1.6 Billion nodes), when scaling up to 8192 cores. Those figures are not included because of page limits.

Poor scaling of random workstealing stems from the fact that with increase in core counts, work per core becomes sparse and multiple cores are idle and looking for work. Thief cores randomly probe each other which results in increase in negative acknowledgments. This spikes the total messages in the system. Use of designated matchmakers addresses this problem.

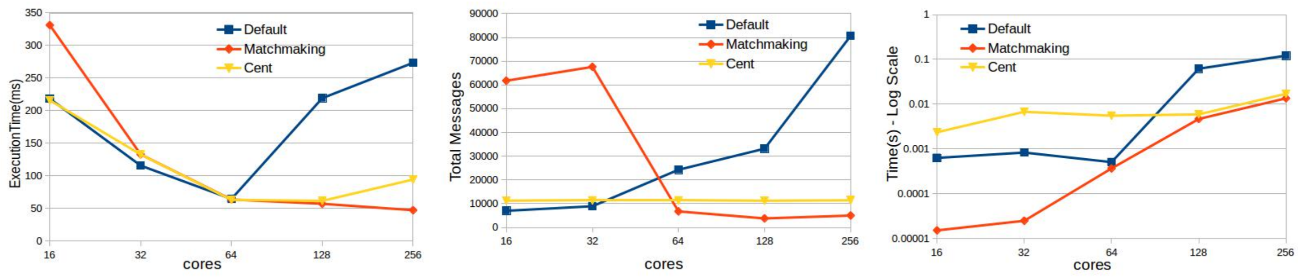
Hamiltonian Path Problem

To show that matchmaking is effective for smaller problem sizes and scales, we also report measurements for the Hamiltonian Path Problem benchmark for a smaller size problem – the ”Knights tour” problem – which scales up to 256 cores. As seen in Figure 2a, the



(a) Execution Time (b) Total Scheduling Messages (c) Idle Core Wait Time

Figure 1: T1L Strong Scaling



(a) ExecutionTime (b) Total Workstealing Messages (c) Idle Core Wait Time

Figure 2: Hamiltonian Strong Scaling

matchmaking scheduler achieves better speedup than the default workstealing scheduler on average by 18.3% and up to 82%. The total messages are reduced by up to 93% (figure 2b). We observe that beyond 64 cores, the default scheduler performance degrades, whereas matchmaking continues to scale to 256 cores. It outperforms the default scheduler because of fewer messages in the system and quick reconciliation of work from producers to consumers as seen in figure 2c.

Additional experimental results (not included here because of page limits) indicate better scalability and improved scheduler efficiency in terms of how well machine resource are utilized for application execution. This efficiency is achieved due to faster work dissemination, better load balance in system, lower wait times for idle cores, and significantly lowered message counts needed for workstealing.

5. Future Work

Future work will consist of evaluation with other benchmarks and applications.Future extensions of framework will include following improvements in matchmaker, consumer and producer modules: *(i) Data Affinity considerations at matchmaker node-* steal requests looking for work are matched with available work based on input data distribution so that the data communication is minimized/localized. *(ii) At consumer node-* Decorating the tasks in terms of load characteristics (memory bound vs compute bound vs I/O bound etc) and determining the load request to the existing conditions at a node, e.g. if too many memory bound tasks currently exist on other cores at a given node, a given core at this node puts out a steal request asking for a compute bound task.
(iii) predictive and distributed scheduling- We have developed a compiler framwork[11] to improve ”accuracy” and ”timeliness” of load estimates. The compiler inserts predictive beacons at ”as-early-as-possible” program points which provide load estimates. Beacons are triggered at run-time and allows a node to anticipate forthcoming load variation and determine whether it will become a producer or consumer or none. Based on this information requests anticipated for ”future time window W1” are sent to a particular matchmaker and requests for window W2 sent to a different matchmaker. This intelligent load prediction + routing enables ”timely” load balancing in a ”distributed” fashion.

6. Conclusion

In this paper, we demonstrate that a key limitation of existing workstealing approaches to scale and improve the performance for irregular applications, is their inability to rapidly and with low overheads ”match” excess work to available, idle, resources. This affects both the application execution time and the overall machine efficiency. To address this problem, we propose a novel workstealing framework based on matchmakers, whose role is to facilitate the matching of work producers to work consumers in a scalable and lightweight manner. A prototype matchmaking scheduler is implemented for Charm++, and compared to the Charm++ default workstealing scheduler and a centralized load balancer. Using benchmarks representative of irregular workloads, we demonstrate that matchmaker-based scheduling leads to significant improvements in the ability to scale these types of workloads and reduce their execution time. To the best of our knowledge this is the first workstealing scheme in which thieves obtain work with only a single steal request. We demonstrate the impact this has on improving application speedup, machine efficiency, reduction in latency in finding work, and dramatic reduction in communication. These properties effectively achieve a scale-up as demonstrated in the paper.

Acknowledgment

The authors gratefully acknowledge that the work of this paper was partially supported by the NSF grant CNS 1320752, CNS 1217577, and DOE’s ExaOS Hobbes project

References

[1] Vijay A. Saraswat, Prabhanjan Kambadur, Sreedhar Kodali, David Grove, and Sriram Krishnamoorthy. Lifeline-based global load balancing. In *Proc. of the 16th ACM Symposium on Principles and Practice of Parallel Programming*, PPoPP '11, 2011.

[2] Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. *J. ACM*, 1999.

[3] Quan Chen, Long Zheng, and Minyi Guo. Dws: Demand-aware work-stealing in multi-programmed multi-core architectures. In *Proceedings of Programming Models and Applications on Multicores and Manycores*, PMAM'14, 2014.

[4] De Boelelaan A, Rob van Nieuwpoort, Rob V. Van Nieuwpoort, Jason Maassen, Jason Maassen, Gosia Wrzesinska, Gosia Wrzesiska, Thilo Kielmann, Thilo Kielmann, Thilo Kielmann, Henri E. Bal, and Henri E. Bal. Adaptive load-balancing for divide-and-conquer grid applications. *J. of Supercomputing*, 2004.

[5] Neary, Michael O., and Peter Cappello. "Advanced eager scheduling for Javabased adaptive parallel computing." Concurrency and Computation: Practice and Experience, 2005.

[6] Menon, Harshitha, and Laxmikant Kale. A distributed dynamic load balancer for iterative applications. *Proc. of the International Conference on High Performance Computing, Networking, Storage and Analysis. 2013.*

[7] Sun, Yanhua, et al. ParSSSE: An adaptive parallel state space search engine. *Parallel Processing Letters*, 2011.

[8] Guo, Yi, et al. "Work-first and help-first scheduling policies for async-finish task parallelism." Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on. IEEE, 2009.

[9] Seung-Jai Min, Costin Iancu, and Katherine Yelick. Hierarchical Work Stealing on Manycore Clusters. In *Proc. of Fifth Conference on Partitioned Global Address Space Programming Models*, 2011.

[10] James Dinan, D. Brian Larkins, P. Sadayappan, Sriram Krishnamoorthy, and Jarek Nieplocha. Scalable work stealing. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, 2009.

[11] Vinit Deodhar, Hrushit Parikh, Ada Gavrilovska, and Santosh Pande. Compiler Assisted Load Balancing on Large Clusters. In *Proc of International Conference on Parallel Architectures and Compilation Technology (PACT"15)*, 2015.

[12] Alessandro Morari, Antonino Tumeo, Daniel Chavarra-Miranda, Oreste Villa, and Mateo Valero. Scaling Irregular Applications through Data Aggregation and Software Multithreading. In *Proc. of Int't Parallel and Distributed Computing Confference (IPDPS'14)*, 2014.

[13] Guo, Yi, et al. "SLAW: A scalable locality-aware adaptive work-stealing scheduler." Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on. IEEE, 2010.

[14] J. Chen, A. Choudhary, S. Feldman, B. Hendrickson, C. Johnson, R. Mount, V. Sarkar, V. White, and D. Williams. Synergistic Challenges in Data-Intensive Science and Exascale Computing. DOE ASCAC Data Subcommittee Report, 2013.

[15] Jonathan Lifflander, Sriram Krishnamoorthy, and Laxmikant V. Kale. Work Stealing and Persistence-based Load Balancers for Iterative Overdecomposed Applications. In *Proc of Symposium on High Performance Distributed Computing (HPDC'12)*, 2012.

[16] Zheng, Gengbin, et al. Periodic hierarchical load balancing for large supercomputers. em International Journal of High Performance Computing Applications (2011): 1094342010394383

[17] Rodrigues, Eduardo R., et al. "A comparative analysis of load balancing algorithms applied to a weather forecast model." Computer Architecture and High Performance Computing (SBAC-PAD), 2010 22nd International Symposium on. IEEE, 2010.