Efficient Distributed Workstealing via Matchmaking

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 load variability. One approach to achieving balanced resource use is to over decompose the problem on fine-grained tasks that are then dynamically balanced using 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. We contend that the fundamental problem of distributed work-stealing is of rapidly bringing together work producers and consumers. In response, we develop an algorithm that performs timely, lightweight and highly efficient *matchmaking* between work producers and consumers which results in accurate load balance. Experimental evaluations show that our scheduler is able to outperform other distributed work stealing schedulers, and to achieve scale beyond what is possible with current approaches.

Categories and Subject Descriptors D [Software]: Systems

Keywords work-stealing, irregular applications, scheduling

1. Introduction

Many emerging classes of large-scale applications, such as data-intensive scientific discovery and combinatorial problems such as Graph Search exhibit highly unpredictable and dynamic load variation which results in huge imbalances on large scale clusters. The execution of such workloads is commonly supported by run-times that provide for problem over-decomposition and dynamic work migration to balance the load across the HPC machine resources which maximizes application performance. Work migration for irregular combinatorial applications is handled by workstealing[3] in which idle nodes randomly probe other nodes to find a victim with excess work and steal its tasks. However, traditional workstealing on large scale machines suffers from limited scalability due to high overheads. 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.

In response to these observations, we posit that the key limitation of existing approaches 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 *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). The need to randomly probe nodes or *periodically* send and receive load information to make steal decisions [1] is alleviated.

2. Matchmaking Based Workstealing

Basic Idea

A node in the system is classified as either **busy**, **producer**, **consumer** or **matchmaker**. A *busy* node is a node that is currently executing a task. It may or may not have any pending tasks in its queue. A *producer* node is a node that has pending tasks in its queue such that the number of pending tasks are above a threshold. A producer finds a matchmaker and sends a request announcing the availability of load. A *consumer* is a node which is idle. It determines the current matchmaker node and sends a match request for a producer who it can ‘steal’ work from. A *matchmaker* is a node which is (currently) assigned the responsibility of matching producers and consumers. The advantages of this scheme are (i) instead of thieves randomly looking for work across the cluster and failing, we direct them to a specific location (i.e. matchmaker) where they can be quickly paired with producers who are looking to offload work in same the timeframe. (ii) producer and consumer requests are sent on a per demand basis and not periodically. Thus, total messages are a function of imbalance in the system i.e. better load balancing further reduces the number of messages exchanged. This puts a bound on total messages. The overall impact is drastic reduction of (a) failed steals (hence idle time of thieves), (b) network latency penalty and (c) unnecessary interruptions of nodes which are busy but not overloaded.

Design Considerations

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

An important aspect of the matchmaking framework is its flexibility to allow different spatial or temporal metrics to be incorporated in the scheduler. For instance, in one design- matchmakers can be strategically selected based on cluster topology which leads

Figure 1

Figure 2: T1L Strong Scaling

to hierarchical distribution of matchmakers and load imbalances are quickly resolved in a localized manner.

More interestingly, for workloads where future load supply or demand can be predicted at run-time (as shown in our prior work [2])-temporally proximate producer/consumer requests can be sent to a designated matchmaker- i.e producer/consumer requests anticipated for a future time window W_1 are sent to a particular matchmaker and requests anticipated for window W_2 sent to a different matchmaker. In this way, matchmakers are responsible for anticipated load balancing for specific, smaller temporal windows, thus resulting in ability to scale further in a distributed manner.

3. Experimental Evaluation

The matchmaking algorithm is developed within 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 of Charm++ (marked **Default** in all graphs), and a conventional centralized load balancer (marked **Cent** in all graphs). The charm++ scheduler employs steal-half random workstealing strategy[3]. In centralized scheduler every node periodically sends its load information to a central node which then makes load shifting decisions. The schedulers are evaluated using the commonly used Unbalanced Tree Search (UTS) benchmark [4]. We show results for the T1L data set which is an unbalanced geometric tree of 100 million nodes.

In Figures 2a and 2b we observe that matchmaking delivers 21%-72% performance benefits with 62%-99% reduction in total messages compared to default work-stealing when scaling the execution of T1L from 64-4096 cores. We observe that the default work-stealing scheduler stops scaling at merely 128 cores, whereas with matchmaking, the execution time continues improving up to 512 cores. Furthermore, beyond the scaling limits, the default scheduler exhibits drastic degradation while the impact on performance with matchmaking is very gradual.

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 and also idle wait time of thieves. Work remains concentrated within a small subset of nodes while the remaining nodes stay idle. Use of designated matchmakers addresses this problem.

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.

Acknowledgments

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

References

- [1] Neary, Michael O., and Peter Cappello. "Advanced eager scheduling for Javabased adaptive parallel computing." *Concurrency and Computation: Practice and Experience*, 2005.
- [2] Vinit Deodhar, Hrushi 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.
- [3] 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*, 2009.
- [4] Stephen Olivier, Jun Huan, et al. Uts: An unbalanced tree search benchmark. In *Proceedings of the 19th International Conference on Languages and Compilers for Parallel Computing, LCPC'06*