Achieving Data-Aware Load Balancing through Distributed Queues and Key/Value Stores

Ke Wang[*], Ioan Raicu[*†]
[*]Department of Computer Science, Illinois Institute of Technology, Chicago IL, USA
[†]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne IL, USA
kwang22@hawk.iit.edu, iraicu@cs.iit.edu

***Abstract*— Load balancing techniques (e.g. work stealing) are important to obtain the best performance for distributed task scheduling system. In work stealing, tasks are randomly migrated from heavy-loaded schedulers to idle ones. However, for data-intensive applications where tasks are dependent and task execution involves processing large amount of data, migrating tasks blindly would compromise the data-locality incurring significant data-transferring overhead. In this work, we propose a data-aware work stealing technique that combines key-value stores and distributed queues enabling it to achieve good load balancing, all while maximizing data-locality. We leverage a distributed key-value store, ZHT, as a meta-data service that stores task dependency and data-locality information. We implement the proposed technique in MATRIX, a distributed task execution fabric. We evaluate the work with all-pairs application structured as direct acyclic graph from biometrics, and compare with Falkon data-diffusion technique.**

I. INTRODUCTION

As systems are growing exponentially in parallelism [1], more data-intensive applications [2] are becoming loosely-coupled containing many small jobs/tasks (e.g. per-core [16]) with shorter durations (e.g. sub-second). Future programming models will likely employ over-decomposition [3] generating even many more fined-grained tasks than available parallelism. This poses significant challenges on task scheduling system to make extremely fast scheduling decisions (e.g. millions/sec). The Many-task computing (MTC) [4] [7][11] paradigm aims to define and address the challenges of scheduling fine grained data-intensive workloads [5]. MTC applications are structured as direct acyclic graphs (DAG) of discrete tasks, with data dependencies forming the graph edges.

The task scheduling system for MTC will need to be fully-distributed [12][15][17]. Each compute node runs one scheduler and one or more executors. The schedulers are fully-connected, and receive workloads to schedule tasks locally. Load balancing [6] is challenging for fully-distributed architecture. This work adopts the work stealing technique [20][10], in which, the idle schedulers communicate with neighbors to balance their loads. However, as more applications are experiencing data explosion [8] such that tasks are dependent and task execution involves processing large amount of data, data-aware scheduling and load balancing are two indispensable yet orthogonal needs. In this work, We propose a data-aware work stealing technique that combines distributed key-value stores (DKVS) [12][19][15] and distributed queues enabling it to satisfy both needs. We leverage ZHT [9] as a meta-data service that stores task dependency and data-locality information. We apply four distributed task queues to keep tasks in different states. We implement our technique in MATRIX [10][18].

II. DATA-AWARE WORK STEALING

A. DKVS Used as a Meta-Data Service

We apply a DKVS, i.e. ZHT, to store the data dependency and locality information of all the tasks. The "key" is task id, and the "value" is the important meta-data that is defined (see Figure 1) as the following data structure conceptually:

```
typedef    TaskMetaData
{
    int    num_wait_parent;    // number of waiting parents
    vector<string> parent_list;    // schedulers that run each parent task
    vector<string> data_object;    // data object name produced by each parent
    vector<long> data_size;    // data object size (byte) produced by each parent
    long all_data_size;    // all data object size (byte) produced by all parents
    vector<string> children;    // children of this tasks
} TMD;
```

Figure 1: Data structure of task metadata

Upon task submission, the client takes an application workload (represented as a DAG), sends the task meta-data to ZHT, and submits the tasks to MATRIX.

B. Distributed Queues in MATRIX

Each scheduler would maintain four local task queues: task wait queue (*WaitQ*), dedicated local task ready queue (*LReadyQ*), shared work stealing task ready queue (*SReadyQ*), and task complete queue (*CompleteQ*). These queues hold tasks in different states stored as meta-data in ZHT.

a) WaitQ

Initially, the scheduler would put all the incoming tasks to the *WaitQ*. A thread keeps checking every task in the *WaitQ* to see whether the dependency conditions for that task are satisfied. Only if the value of the field of "num_wait_parent" in the meta-data is equal to 0 would the task be ready to run.

b) LReadyQ and SReadyQ

When a task is ready to run, the scheduler makes decision to put it in either the *LReadyQ*, or the *SReadyQ*, according to the size and location of the data required by the task. The *LReadyQ* stores the tasks that require large volume of data, and the majority of the required data is located locally; these tasks could only be executed locally. The *SReadyQ* stores the tasks that could be migrated to any scheduler for load balancing's purpose; these tasks either don't need any input data, or the

demanded data volume is so small that the transferring overhead is negligible. The executor keeps pulling ready tasks to execute. It first pops tasks from *LReadyQ*, and then pops tasks from *SReadyQ* if the *LReadyQ* is empty. When executing a task, the executor first gets the data either from local or remote nodes. If both queues are empty, the scheduler would start doing work stealing.

c) CompleteQ

When a task is done, it is moved to the *CompleteQ*. A thread is responsible for updating the meta-data for all the children of each completed task. The thread first queries the meta-data of the completed task to find out the children, and then updates each child's meta-data.

III. EVALUATION

We evaluate our technique on the Kodiak cluster from the PROBE [21] of Los Alamos National Laboratory with all-pairs application, and compare with Falkon [22] data-diffusion technique [13][14] up to 200 cores. All-Pairs is a common benchmark in Biometrics that describes the covariance of two sequences of gene codes. In this workload, each task execute for 100-ms to compare two 12MB files with one from each set. We run strong-scaling experiments with a 500*500 workload size (250K tasks). This is the same workload referenced in [14].

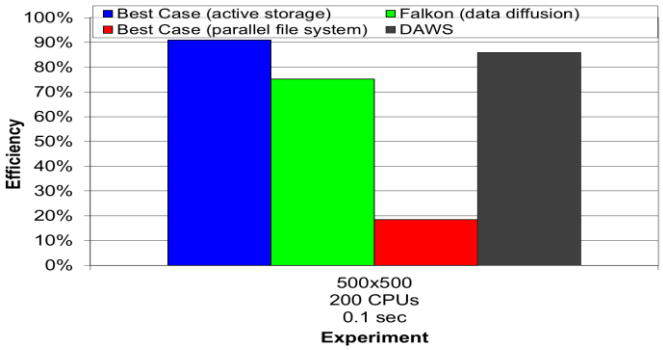

Figure 2: Comparison between Data Diffusion and DAWS

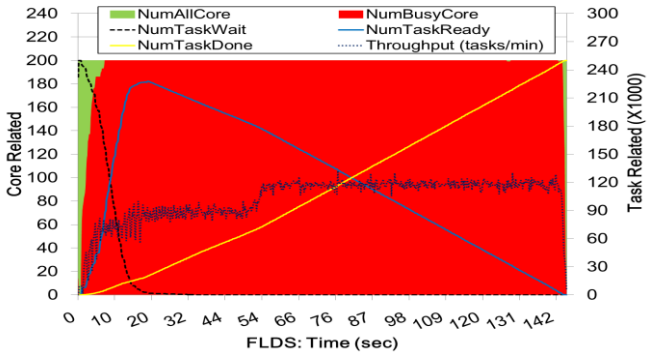

Figure 3: Utilization graph at 200 cores

We compare our data-aware work stealing (DAWS) technique with Data-Diffusion [14] in Figure 2. We see that for 100-ms tasks, our technique improved Data Diffusion by 14.21% (85.9% vs 75%), and it is quite close to the best case using active storage (85.9% vs 91%). Data diffusion applies a centralized index-server for data-aware scheduling, while our technique utilizes DKVS that is much more scalable. In addition, we show the utilization graph running the all-pairs workload in Figure 3. The utilization (read area / green area) is high. At beginning, it takes very little time for balancing the load (the short ramp-up period), and our technique does not exhibit long-tail problem at the final stage.

REFERENCES

[1] V. Sarkar et al. "ExaScale Software Study: Software Challenges in Extreme Scale Systems", ExaScale Computing Study, DARPA IPTO, 2009.

[2] I. Raicu et al. "Many-Task Computing for Grids and Supercomputers", Invited Paper, IEEE MTAGS 2008.

[3] X. Besseron and T. Gautier. "Impact of Over-Decomposition on Coordinated Checkpoint/Rollback Protocol", Euro-Par 2011: Parallel Processing Workshops, Volume 7156, 2012, pp 322-332.

[4] I. Raicu. "Many-Task Computing: Bridging the Gap between High Throughput Computing and High Performance Computing", ISBN: 978-3-639-15614-0, VDM Verlag Dr. Muller Publisher, 2009.

[5] I. Raicu et al. "Toward Loosely Coupled Programming on Petascale Systems", IEEE/ACM Supercomputing 2008.

[6] M. H. Willebeek et al. "Strategies for dynamic load balancing on highly parallel computers," In IEEE Transactions on Parallel and Distributed Systems, volume 4, September 1993.

[7] K. Wang et al. "Paving the Road to Exascale with Many-Task Computing", Doctoral Showcase, IEEE/ACM Supercomputing/SC 2012.

[8] A. S. Szalay et al. "GrayWulf: Scalable Clustered Architecture for Data-Intensive Computing", Proceedings of the 42nd Hawaii International Conference on System Sciences, Hawaii, 5 to 8 January 2009, paper no. 720; available as Microsoft Tech Report MSR-TR-2008-187 at http://research.microsoft.com/apps/pubs/default.aspx?id=79429.

[9] T. Li et al. "ZHT: A Light-weight Reliable Persistent Dynamic Scalable Zero-hop Distributed Hash Table", IPDPS 2013.

[10] K. Wang, et al. "MATRIX: MAny-Task computing execution fabRIc at eXascale," tech report, IIT, 2013.

[11] K. Wang et al. "SimMatrix: Simulator for MAny-Task computing execution fabRIc at eXascales," ACM HPC 2013.

[12] K. Wang et al. "Using Simulation to Explore Distributed Key-Value Stores for Extreme-Scale Systems Services," IEEE/ACM Supercomputing/SC 2013.

[13] I. Raicu et al. "Accelerating Large-scale Data Exploration through Data Diffusion", International Workshop on Data-Aware Distributed Computing 2008, co-locate with ACM/IEEE HPDC 2008.

[14] I. Raicu et al. "The Quest for Scalable Support of Data Intensive Workloads in Distributed Systems", ACM HPDC 2009.

[15] K. Wang et al. "Next Generation Job Management Systems for Extreme-Scale Ensemble Computinig", short paper, ACM HPDC 2014.

[16] K. Wang et al. "Modeling Many-Task Computing Workloads on a Petaflop IBM Blue Gene/P Supercomputer." IEEE 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW) 2013.

[17] K. Wang et al. "Exploring Design Tradeoffs for Exascale System Services through Simulation." Tech Report, LANL 2013.

[18] K. Ramamurthy et al. "Exploring Distributed HPC Scheduling in MATRIX." Tech Report, IIT, 2013.

[19] X. Zhou et al. "Exploring Distributed Resource Allocation Techniques in the SLURM Job Management System." Tech Report, IIT, 2013.

[20] K Wang et al. "Centralized and Distributed Job Scheduling System Simulation at Exascale." Tech Report, IIT, 2011.

[21] G. Grider. "Parallel Reconfigurable Observational Environment (PRObE)," available from http://www.nmc-probe.org, October 2012.

[22] I. Raicu et al. "Falkon: A Fast and Light-weight tasK executiON Framework," IEEE/ACM SC 2007.