Heterogeneous Cores For MapReduce Processing: Opportunity or Challenge?

Feng Yan[1,2], Ludmila Cherkasova[1], Zhuoyao Zhang[3], and Evgenia Smirni[2]

[1] Hewlett-Packard Labs, lucy.cherkasova@hp.com
[2] College of William and Mary, fyan,esmirni@cs.wm.edu
[3] University of Pennsylvania, zhuoyao@seas.upenn.edu

***Abstract*—To offer diverse computing capabilities, the emergent modern system on a chip (SoC) might include heterogeneous multi-core processors. The current SoC design is often constrained by a given power budget that forces designers to consider different decision trade-offs, e.g., to choose between many slow cores, fewer faster cores, or to select a combination of them. In this work, we design a new Hadoop scheduler, called DyScale, that exploits capabilities offered by heterogeneous cores for achieving a variety of performance objectives. Our preliminary performance evaluation results confirm potential benefits of heterogeneous multi-core processors for "faster" processing of the small, interactive MapReduce jobs, while at the same time offering an improved throughput and performance for large, batch job processing.**

I. Introduction

One of the new trends in the emergent modern system on a chip (SoC) is to include heterogeneous cores for offering a variety of computing capabilities. The current SoC design is often constrained by a given power budget that forces designers to consider different decision trade-offs, e.g., to choose between many slow, power-efficient cores, fewer faster cores (which consume more power per core), or to select a combination of them. Intuitively, an application that needs to support a higher throughput and that can distribute its load across many cores will favor a choice of many slow cores, while the completion time sensitive applications with performance goals will benefit from fewer, faster cores [13]. A SoC design with heterogeneous multi-core processors might offer the best of both worlds for applications that can take an advantage of these heterogeneous processing capabilities.

MapReduce [6] and its open source implementation Hadoop offer a scalable and fault-tolerant framework for processing large data sets. MapReduce jobs are automatically parallelized, distributed, and executed on a large cluster of commodity machines. Originally, Hadoop was designed for batch-oriented processing of large production jobs. These applications belong to a class of so-called scale-out applications. Traditionally, a simple rule of thumb (applied by the Hadoop users) states [19] that the completion time of a MapReduce job can be reduced twice by processing this job on the double size Hadoop cluster. This rule applies to the large MapReduce jobs that need to process large datasets and that consist of a large number of map (reduce) tasks. Thus, efficient processing of such jobs is "throughput-oriented" and can be significantly improved with additional (scale-out) resources.

However, when multiple users are sharing the same Hadoop cluster, additionally, there are many interactive ad-hoc queries and small MapReduce jobs that are completion time sensitive. For improving the execution time of small MapReduce jobs, one cannot use the scale-out approach, but rather can benefit from the "scale-up" approach, where the tasks comprising such jobs can be executed in "faster" resources.

978-1-4799-0913-1/14/$31.00

Using heterogeneous multi-core processor to improve performance has been well studied [5], [3], [8], [14], [9], [15]. However, the earlier papers focus on a single machine environment while Hadoop is a distributed framework and needs to manage a cluster environment. Thus, it is difficult to apply the traditional techniques for a Hadoop framework. In addition, we aim to support different performance objectives for different classes of Hadoop jobs, and it requires an exact control of running different types of slots in different cores, so the general dynamical thread to core mapping is not suitable here.

Job scheduling in Hadoop is performed by a master node called the JobTracker, which manages a number of worker nodes in the cluster. Each worker node in the cluster is configured with a fixed number of map and reduce slots, and these slots are managed by the local TaskTracker. The worker TaskTracker periodically connects to the master JobTracker to report current status and the available slots. The JobTracker decides the next job to execute based on the reported information and according to a scheduling policy. The popular job schedulers include FIFO, Hadoop Fair scheduler (HFS) [21], and Capacity scheduler [2]. In the case of Hadoop deployment on heterogeneous servers [22], [10], [11], [20], one has to deal with data locality issues and balancing the data placement according to the server capabilities as presented in the earlier work cited above. One of the biggest advantages of Hadoop deployed with heterogeneous processors is that both *fast* and *slow* slots have a similar access to the underlying HDFS data that eliminates the data locality issues.

In this work, we outline design of a new Hadoop scheduler, called DyScale, that exploits capabilities offered by heterogeneous multi-core processors for achieving a variety of performance objectives. It enables creating virtual resource pools based on the core types for multi-class priority scheduling. We describe new mechanisms for enabling the "slow" and "fast" slots in Hadoop and creating the corresponding virtual clusters.

There is a list of interesting *opportunities* for MapReduce processing offered by the heterogeneous processor design. First of all, both *fast* and *slow* Hadoop slots have a similar access to the underlying HDFS data. This eliminates the data locality issues that could make the heterogeneous Hadoop clusters comprised of the *fast* and *slow* servers being inefficient [1]. Therefore, any dataset can be processed by either *fast* or *slow* virtual resource pools, or their combination. Second, the task (job) migration between slow and fast cores enables enhanced performance guarantees and more efficient resource use. Among the *challenges* is the increased complexity of management , e.g., how to dynamically partition the resources for different workloads and track such changes. Since not all

applications benefit the same from using fast cores, there is obvious trade-offs between performance and efficiency.

Our preliminary performance evaluation results demonstrate the efficiency and robustness of the proposed framework. Within the same power budget, the DyScale scheduler that operates over the heterogeneous multi-core processors provides a much better performance for small, interactive jobs compared to using homogeneous processors with (many) slow cores. At the same time, DyScale running with heterogeneous multi-core processors preserves a good performance of large batch jobs compared to using a homogeneous fast core design (with a fewer cores) that results in a significant performance degradation. Thus, the heterogeneous multi-core processors offer a sweet spot for efficient processing of MapReduce job with different performance objectives. The remainder of the paper presents our results in more detail.

II. Framework Design

In this section, we outline a new Hadoop scheduling framework DyScale, which can efficiently use the heterogeneous multi-core processors for MapReduce processing. We first introduce the basic features of DyScale scheduler and then we talk about the enhanced feature of handling spare resources.

DyScale scheduler offers the capability of scheduling jobs according to their performance objectives and the resource preference. Such feature is offered by partitioning different resources into different dedicated virtual resource pools each with its own Job Queue. For example, as shown in Figure 1, fast slots (running on fast cores) can be grouped into the Virtual Fast (vFast) resource pool for serving the Interactive Job Queue and the slow slots (running on slow cores) can be grouped as the Virtual Slow (vSlow) resource pool for serving the Batch Job Queue. A user can submit the small, time-sensitive jobs to the Interactive Job Queue to be executed by fast cores, and the large throughput-oriented jobs to the Batch Job Queue for processing by (many) slow cores.

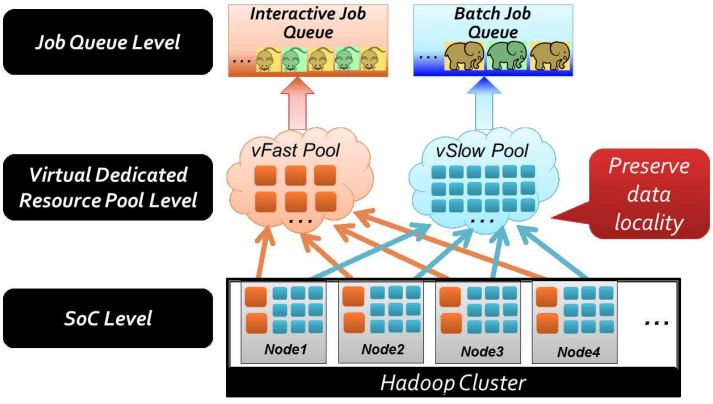

Fig. 1. Virtual Resource Pools.

The attractive part of such virtual resource pool arrangement is that it *preserves data locality* because both the fast and slow slots have the same data access to the datasets stored in the underlying HDFS. Therefore, any dataset can be processed by either *fast* or *slow* virtual resource pools, or their combination.

To support a virtual resource pool design, the TaskTracker needs additional mechanisms such as

- the ability to start a task on a specific core, i.e., to run a slot on a specific core and assign a task to it , and
- maintain the mapping information between a task and the assigned slot type.

TaskTracker always starts a new JVM for each task instance (if the JVM reuse feature in Hadoop is disabled). It is done for a reason that the JVM failure does not impact other tasks or take down the TaskTracker. Running a task on a specific core can be achieved by binding the JVM to that core. We use the *CPU affinity* to implement this feature. By setting the CPU affinity, a process can be bounded to one or a set of cores. TaskTracker calls *spawnNewJvm* class to spawn JVM in a new thread. The CPU affinity can be specified during the spawn to force the JVM to run on the desired core (e.g., fast or slow).

Additional advantage of using the CPU affinity is that it *can be changed during the runtime*. Therefore, if the JVM reuse feature is enabled (note that the JVM reuse can be enabled only for the tasks of the same job), the task can be placed on a desired core by changing the CPU affinity of the JVM. The mapping information between tasks and cores can be maintained by recording *task_ID, JVM_pid, core_id)* in the TaskTracker table. So when a task finishes, the TaskTracker knows whether the released slot is a fast slot or a slow slot.

The JobTracker also needs to know whether the available slot is a slow or fast slot to make resource allocation decision. DyScale communicates this information through the *heartbeat*, which is essentially RPC (Remote Procedure Call) between TaskTracker at a worker and JobTracker at the master node.

TaskTracker asks the JobTracker for a new task when the currently running tasks are below the configured maximum allowed number of map/reduce tasks through a boolean parameter *askForNewTask*. If the TaskTracker can accept new task, JobTracker calls the Hadoop Scheduler for a decision to assign a task to this TaskTracker (to this worker node).

The Scheduler checks *TaskTrackerStatus* to know whether the available slots are Map or Reduce slots. In the DyScale case, the Scheduler additionally needs to distinguish the slot type (i.e., slow or fast slot). There are four types of slots: *i)* fast map slot, *ii)* slow map slot, *iii)* fast reduce slot, and *iv)* slow reduce slot.

In the DyScale framework, the Scheduler checks the *JobQueue* based on the slot type, e.g., if the available slot is a fast slot, then this slot belongs to vFast pool, and the *InteractiveJobQueue* is selected for a job/task allocation. After selecting the *JobQueue*, it takes the first job in the queue and allocates the available slot.

There could be different policies for ordering the jobs inside the *JobQueue* as well as different slot allocation policies. The default policy is FIFO. However, in a general case, the job ordering/resource allocation depends on the performance objectives and can be defined by either Hadoop Fair Scheduler (HFS) [21], or the ARIA SLO-driven scheduler [17], etc..

The JobTracker then puts a list of current actions, such as LAUNCH_TASK, (or KILL_TASK), etc., in the TaskTrackerAction list to tell the TaskTracker what to do next through the *heartbeatResponse*.

Hadoop jobs may arrive with a different intensity over time and not all tasks finish at the same time, so when a resource pool has spare resources (slots) but the corresponding *JobQueue* is empty while the other *JobQueues* have jobs that are waiting for resources, the static resource partition and allocation becomes inefficient. To improve the efficiency of resource usage, we use Virtual Shared (vShared) Resource pool

to utilize the spare resources. The spare slots can be put into the vShare pool and used by any job queue.

III. Experimental Evaluation

In this section, we conduct simulation experiments to emulate the Facebook workload's execution on the Hadoop cluster with servers using homogeneous versus heterogeneous processors. We select configurations with the same power budget and compare the completion time of Hadoop jobs executed on homogeneous versus heterogeneous processors configurations.

As the heterogeneous multi-core processors are not yet available, we perform a simulation study using the MapReduce simulator SimMR [16] and a synthetic Facebook workload [21]. Our goal is to compare the job completion times and to perform a sensitivity study when a workload is executed by different Hadoop clusters deployed with either homogeneous or heterogeneous multi-core processors.

SimMR consists of the following three components:

- *Trace Generator* that creates a replayable MapReduce workload. In addition, the *Trace Generator* can create traces defined by a synthetic workload description that compactly characterizes the duration of map and reduce tasks as well as the shuffle stage characteristics via corresponding distribution functions. This feature is useful for a sensitivity analysis of new schedulers and resource allocation policies applied to different workload types.
- *Simulator Engine* - a discrete event simulator that accurately emulates the job master functionality in the Hadoop cluster.
- *A pluggable scheduling policy* that dictates the scheduler decisions on job ordering and the amount of resources allocated to different jobs over time.

We have extended SimMR to emulate the DyScale environment, i.e., the heterogeneous slow/fast cores that correspond to slow and fast Hadoop slots.

We approximate the performance and power consumption of different cores from the available measurements of the existing Intel processors [13], [7] executing PARSEC benchmark [4]. The Intel processors i7-2600 and E31240 (used in the HP Proliant DL 120 G7 server) are from the same Sandy Bridge microarchitecture family and have almost identical performance [12]. We summarizes all this data in Table I. With a power budget of 84W, we choose three multi-core processor configurations as shown in in Table II.

In our experiments, we simulate the execution of the Facebook workload on three different Hadoop clusters with multi-core processors shown in Table II. For sensitivity analysis, we evaluate results with different cluster sizes. Here, we present the results for cluster sizes with 25, 40 and 70 nodes as they reflect the interesting possible performance situations.

We configure each Hadoop cluster with 1 map and 1 reduce slot per core, e.g., for a Hadoop cluster size with 40 nodes, the three considered configuration have the following number of map and reduce slots:

- the *Homogeneous-fast* cluster has 160 fast map/reduce slots in total,
- *Homogeneous-slow* configuration has 840 slow map/reduce slots in total, and
- the *Heterogeneous* configuration has 120 fast map/reduce slots and 360 slow map/reduce slots in total.

We generate 1000 hadoop jobs according to the distribution shown in Table III. We consider the jobs from the 1st to 5th group as small interactive jobs (e.g., with less than 100 tasks) and the jobs in remaining five groups as large batch jobs. The interactive jobs are 82% of the total jobs and the batch jobs are 18%. The task duration of the Facebook workload can be best fit with LogNormal distribution [18] and the following parameters: LN(9.9511, 1.6764) for map task duration and LN(12.375, 1.6262) for reduce task duration.

Bin	Map Tasks	Reduce Tasks	# % Jobs
1	1	NA	38%
2	2	NA	16%
3	10	3	14%
4	50	NA	8%
5	100	NA	6%
6	200	50	6%
7	400	NA	4%
8	800	180	4%
9	2400	360	2%
10	4800	NA	2%

TABLE III. Job description for each bin in Facebook workload (from Table 3 in [21]).

For a fair comparison of these three configurations, each job is submitted in the FIFO order, so that there is no bias due to the specific ordering policy nor queuing waiting time for each job, e.g., each job can use the entire cluster resources. For the *heterogeneous* configuration, the SimMR simulation also supports the vShared resource pool so that a job can use both fast and slow resources of the entire cluster.

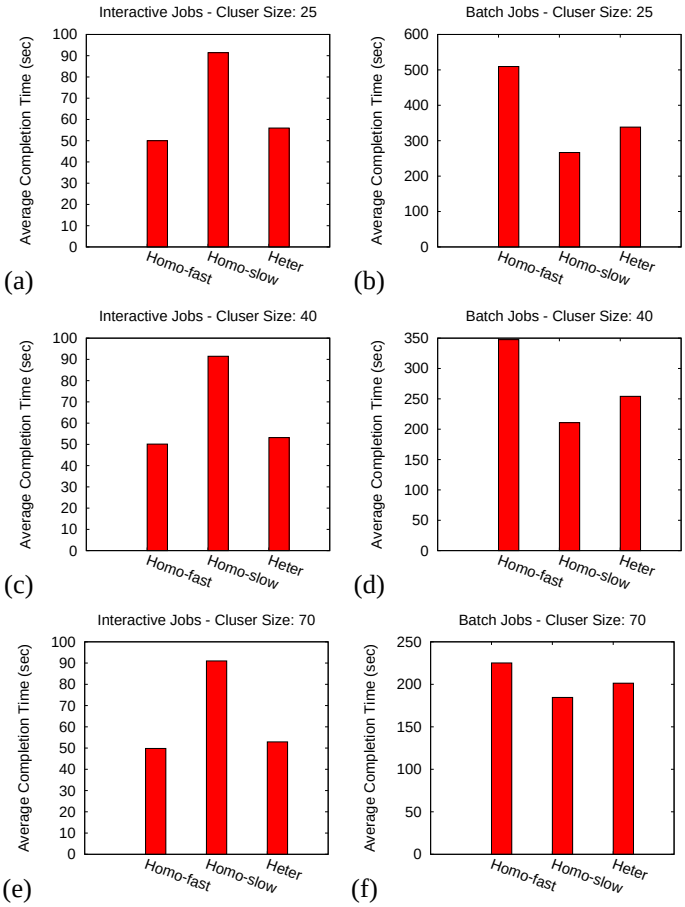

Fig. 2. Completion time of interactive jobs and batch jobs under different configurations: (a)-(b) the Hadoop cluster with 25 nodes, (c)-(d) the Hadoop cluster with 40 nodes, (e)-(f) the Hadoop cluster with 70 nodes.

Type	Processor Name	Technology	Frequency	Power per Core	Normalized Power	Normalized (PARSEC) Performance	Normalized Map Task Performance	Normalized Reduce Task Performance
Type 1	i7-2600 Sandy Bridge	32 nm	3.4 Ghz	21 W	1.0	1.0	1.0	1.0
Type 2	i5-670 Nehalem	32 nm	3.4 Ghz	16 W	0.81	0.92	0.92	0.98
Type 3	AtomD Bonnell	45 nm	1.7 Ghz	4 W	0.19	0.45	0.45	0.83

TABLE I. PROCESSOR SPECIFICATIONS.

Configuration	Type 1	Type 2	Type 3	Power
Homogeneous-fast	4	0	0	84 W
Homogeneous-slow	0	0	21	84 W
Heterogeneous	0	3	9	84 W

TABLE II. PROCESSOR CONFIGURATIONS UNDER THE SAME POWER BUDGET OF 84 W.

We plot the results in Figure 2 and each row corresponds to the cluster size of 25, 40, 70 respectively. The left column shows the average completion time for interactive jobs and the right column is the average completion time for batch jobs.

From these graphs, we can see that for interactive jobs, *Homogeneous-fast* and *Heterogeneous* configurations achieve very close completion time and consistently outperform the *Homogeneous-slow* configuration by being almost twice faster. Indeed, the small interactive job performance does benefit from using faster cores compared to a larger number of slow cores available in *Homogeneous-slow* configuration. These jobs have a limited parallelism and once their tasks are allocated necessary resources, these jobs cannot take advantage of the extra slots available in the system. For small jobs, the fast slots (scale-up approach) are the effective way to achieve a better performance.

For batch jobs, as expected, the scale-out approach shows its advantage because batch jobs have a large number of map tasks. For example, *Homogeneous-slow* configuration consistently outperforms *Homogeneous-fast*, and can be almost twice faster when cluster size is small (e.g., 25 nodes). The interesting result is that the *Heterogeneous* configuration is almost neck-to-neck with the *Homogeneous-slow* configuration for batch jobs. Moreover, as the Hadoop cluster size increases, the performance of *Heterogeneous* configuration becomes more close to the *Homogeneous-slow* configuration (note, that the batch jobs can effectively utilize the additionally available fast slots in the vShared resource pool).

By comparing these results, it is apparent that the heterogeneous multi-core processors with both fast and slow cores become an interesting design point for supporting different performance objectives of MapReduce jobs. For time-sensitive interactive jobs, it may significantly improve the job completion time (at the same power budget). The large batch jobs are benefiting from the larger number of the slower cores that improve throughput of these jobs. Moreover, the batch jobs are capable of taking advantage and effectively utilizing the additionally available fast slots in the vShared resource pool supported by the DyScale framework.

IV. CONCLUSIONS

We proposed a new Hadoop scheduling framework, called DyScale, which aims at taking the advantage of heterogeneous multi-core processors' capabilities for achieving a variety of performance objectives. Our experimental evaluation shows with proper scheduling strategy, heterogeneous multi-core processors configuration outperforms traditional homogeneous processors under the same power budget. In the future work, we plan to exploit the possible migration features, compare our scheduler to more state-of-the-art schedulers, and under more challenge scenarios.

ACKNOWLEGEMENT

This work was completed during F. Yan's internship at HP Labs. Prof. E.Smirni and F. Yan are supported by NSF grants CCF-0937925 and CCF-1218758.

REFERENCES

[1] F. Ahmad et al. Tarazu: Optimizing MapReduce on Heterogeneous Clusters. In *Proc. of ASPLOS*, 2012.
[2] Apache. Capacity Scheduler Guide, 2010.
[3] M. Becchi and P. Crowley. Dynamic thread assignment on heterogeneous multiprocessor architectures. In *Proceedings of the 3rd conference on Computing frontiers*, pages 29–40. ACM, 2006.
[4] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Technical Report TR-811-08, Princeton University*, 2008.
[5] V. Craeynest et al. Scheduling heterogeneous multi-cores through performance impact estimation (pie). In *Proc. of the 39th International Symposium on Computer Architecture*, 2012.
[6] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
[7] H. Esmaeilzadeh, T. Cao, X. Yang, S. M. Blackburn, and K. S. McKinley. Looking back and looking forward: power, performance, and upheaval. *Commun. ACM*, 55(7):105–114, 2012.
[8] A. Fedorova et al. Operating system scheduling on heterogeneous core systems. In *Proc. of OSHMA Workshop*, 2007.
[9] E. Grochowski, R. Ronen, J. Shen, and H. Wang. Best of both latency and throughput. In *IEEE Intl. Conf. on Computer Design (ICCD'2004)*.
[10] G. Gupta, C. Fritz, B. Price, R. Hoover, J. DeKleer, and C. Witteveen. ThroughputScheduler: Learning to Schedule on Heterogeneous Hadoop Clusters. In *Proc. of ICAC*, 2013.
[11] G. Lee, B.-G. Chun, and R. H. Katz. Heterogeneity-aware resource allocation and scheduling in the cloud. In *Proc. of the 3rd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud*, 2011.
[12] PassMarkSoftware. CPU Benchmarks, 2013.
[13] S. Ren, Y. He, S. Elnikety, and S. McKinley. Exploiting Processor Heterogeneity in Interactive Services. In *Proc. of ICAC*, 2013.
[14] D. Shelepov and A. Fedorova. Scheduling on heterogeneous multicore processors using architectural signatures. In *Workshop on the Interaction between Operating Systems and Computer Architecture*, 2008.
[15] K. Van Craeynest and L. Eeckhout. Understanding fundamental design choices in single-isa heterogeneous multicore architectures. *ACM Transactions on Architecture and Code Optimization*, 9(4):32, 2013.
[16] A. Verma, L. Cherkasova, and R. H. Campbell. Play It Again, SimMR! In *Proc. of Intl. IEEE Cluster'2011*.
[17] A. Verma, L. Cherkasova, and R. H. Campbell. ARIA: Automatic Resource Inference and Allocation for MapReduce Environments. In *Proc. of ICAC*, 2011.
[18] A. Verma, L. Cherkasova, V. S. Kumar, and R. H. Campbell. Deadline-based workload management for mapreduce environments: Pieces of the performance puzzle. In *NOMS*, 2012.
[19] T. White. *Hadoop:The Definitive Guide*. Page 6,Yahoo Press.
[20] J. Xie et al. Improving mapreduce performance through data placement in heterogeneous hadoop clusters. In *Proc. of the IPDPS Workshops: Heterogeneity in Computing*, 2010.
[21] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *Proc. of EuroSys*. ACM, 2010.
[22] Z. Zhang, L. Cherkasova, and B. T. Loo. Performance Modeling of MapReduce Jobs in Heterogeneous Cloud Environments. In *Proc. of IEEE CLOUD*, 2013.