IJCSE International Journal of Computer Sciences & Engineering Open Access
Review Paper Volume-4, Special Issue-3, May 2016 E-ISSN: 2347-2693

Adjudicator: A Pluggable Multiclass Job Scheduler

Nishmitha K.S[1] , Megahana K[2], Monica N[3], Anima P[4], Saleem Malik[5]

[1,2,3,4] UG Scholar, [5]Asst. Professor
[1,2,3,4,5]Department of Computer Science and Engineering
[1,2,3,4,5] KVGCE, Sulllia

Available online at: www.ijcseonline.org

Abstract — The responsibility of contemporary multi-core processors is oftentimes bent on by a given power ration that requisite developer to evaluate different resolution trade-offs, e.g., to espouse between many slow, power-efficient cores, or fewer faster, power-hungry cores, or a amalgamation of them . Here, a prototype, a new Hadoop scheduler, called adjudicator, that utilizes aptness proffered by heterogeneous cores within a single multi-core processor for accomplishing a variety of performance objectives. Heterogeneous multi-core processors enable creating virtual resource pools based on "slow" and "fast" cores for multi-class priority scheduling. Since the same data can be accessed with either "slow" or "fast" apertures, spare resources (apertures) can be shared between different resource pools. Using sample experimental data and via simulation, a wrangle is made in approbation of heterogeneous multi-core processors as they achieve "faster" processing of small, interactive MapReduce jobs, while proffering improved throughput for large, batch jobs. Evaluation is done on performance benefits of adjudicator versus the FIFO and Capacity job schedulers that are broadly used in the Hadoop community.

Keywords- Hadoop, MapReduce, Adjudicator, Job scheduler, Computing, Heterogeneous

I. INTRODUCTION

To propound distinctive performance and computing proficiency, the dawning modern system on a chip (SoC) may include heterogeneous cores that execute the same instruction set while exhibiting different power and performance quirk . The SoC design is oftentimes driven by a power ration that limits the number (and type) of cores that can be put on a chip. The power constraints force developers to utilize a variation of choices within the same power envelope and to analyze decision tradeoffs as shown in Figure 1.

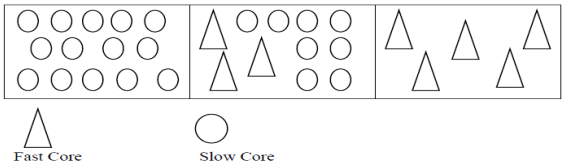

Fig.1 Different choices in the processor design

A number of interesting choices may exist, but once the SoC design is chosen, it defines the configuration of the produced chip, where the number and the type of cores on the chip is fixed and cannot be changed. MapReduce and its open source implementation Hadoop proffer 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. When multiple users share the same Hadoop cluster, there are many interactive ad-hoc queries and small MapReduce jobs that are completion-time sensitive.

In addition, a growing number of MapReduce applications (e.g., personalized advertising, sentiment analysis, spam detection) are deadline-driven, hence they require completion time guarantees. To improve the execution time of small MapReduce jobs, one cannot use the "scale-out" approach, but could benefit using a "scale-up" approach, where chores execute on "faster" resources. A typical perception of a MapReduce processing pipeline is that it is disk-bound (for small and medium Hadoop clusters) and that it can become network-bound on larger Hadoop clusters[1][2]

Here, objective of paper is to design and evaluate Adjudicator, a new Hadoop scheduler that exploits capabilities proffered by heterogeneous cores for achieving a variety of performance objectives. These heterogeneous cores are used for creating different virtual resource pools, each based on a distinct core type. These virtual pools consist of resources of distinct virtual Hadoop clusters that operate over the same datasets and that can share their resources if needed. Resource pools can be exploited for multiclass job scheduling. Within the same power ration, Adjudicator operating on heterogeneous multi-core processors provides significant performance improvement for small, interactive jobs comparing to using homogeneous processors with (many) slow cores. Adjudicator can reduce the average completion time of time-sensitive interactive jobs by more than 40%. At the same time, Adjudicator maintains good performance for large batch jobs compared to using a homogeneous fast core design (with fewer cores). The considered heterogeneous configurations can reduce completion time of batch jobs up to 40%.

II. MAPREDUCE PROCESSING

In the MapReduce model [3] computation is expressed as two functions: map and reduce. MapReduce jobs are executed across multiple machines: the map stage is partitioned into map chores and the reduce stage is partitioned into reduce chores. The map and reduce chores are executed by map apertures and reduce apertures. In the map stage, each map chore reads a split of the input data, applies the user-defined map function, and generates the intermediate set of key/value pairs.

The map chore then sorts and partitions these data for different reduce chores according to a partition function. In the reduce stage, each reduce chore fetches its partition of intermediate key/value pairs from all the map chores and sorts/merges the data with the same key. After that, it applies the user-defined reduce function to the merged value list to produce the aggregate results . Then the reduce output is written back to a distributed file system.

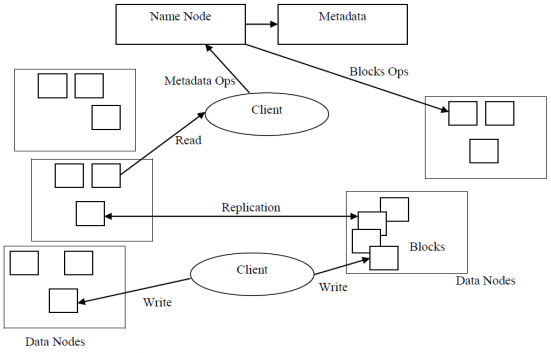

Fig. 2: HDFS architecture

Job scheduling in Hadoop is performed by a master node called JobScrutineer, which manages a number of worker nodes. Each worker node is configured with a fixed number of map and reduce apertures, and these apertures are managed by the local ChoreScrutineer. The ChoreScrutineer periodically sends heartbeats to the master JobScrutineer via TCP handshakes. The heartbeats contain information such as current status and the available apertures. The JobScrutineer decides the next job to execute based on the reported information and according to a scheduling policy. Popular job schedulers include FIFO, Hadoop Fair scheduler (HFS) [4], and Capacity scheduler [5]. FIFO is the default and schedules MapReduce jobs according to their submission order. This policy is not efficient for small jobs if large jobs are also present. The Hadoop Fair Scheduler aims to solve this problem. It allocates on average the same amount of resources to every job over time so that small jobs do not suffer from delay penalties when scheduled after large jobs and large jobs do not starve. The Capacity scheduler proffers similar features as the HFS but has a different design philosophy. It allows users to define different queues for different types of jobs and to configure a percentage of share of the total resources for each queue in order to avoid FIFO's shortcomings.

The Hadoop implementation includes counters for recording timing information such as start and finish timestamps of the chores, or the number of bytes read and written by each chore. These counters are sent by the worker nodes to the master node periodically with each heartbeat and are written to logs. Counters help profile the job performance and provide important information for designing new schedulers. We utilize the extended set of counters from [6] in Adjudicator.

III. RELATED WORK

There is a body of work traversing power and performance trade-offs using heterogeneous multi-core processors. Some papers focus on the power savings aspect. [7], while others concentrate on the performance aspect,[8], [9] that examine techniques such as monitoring, evaluating thread performance, and dynamically mapping threads to different
core types. [2] propose using architecture signatures to guide thread scheduling decisions.

The proposed method needs to modify the applications for adding the architecture signatures, therefore it is not practical to deploy. These proposed techniques focus on improving the overall chip-level throughput. The work in [3] explores the per-program performance in addition to the overall chip level throughput when using heterogeneous multi-core processors. Here, project aims to support different performance objectives for classes of Hadoop jobs, which requires an exact control of running different types of apertures in different cores, therefore dynamical mapping of threads to cores is not suitable here. Performance analysis and optimization of MapReduce processing in the heterogeneous server environment is the subject of several. Load-balancing and load re-balancing approaches in a heterogeneous cluster is used in [2], [4] to allow the faster node to get more data, such that reduce chores finish approximately at the same time. [5] use data placement to optimize performance in heterogeneous environments. Faster nodes store more data and therefore run more chores without data transfer.[6] use off-line profiling of the jobs execution with respect to different heterogeneous nodes in the cluster and optimize the chore placement to improve the job completion time.[7] propose to divide the resources into two dynamically adjustable pools and use the new metric "progress share" to define the share of a job in a heterogeneous environment so that better performance and fairness can be achieved. This approach only allocates resources based on the job storage requirement. [8] modify the MapReduce scheduler to enable it to use special hardware like GPUs to accelerate the MapReduce jobs in the heterogeneous MapReduce cluster. [9] developed a MapReduce-like system in heterogeneous CPU and GPU clusters. All the above efforts focus on the server level heterogeneity in Hadoop cluster.

In the case of Hadoop deployment on heterogeneous servers, one has to deal with data locality and balancing

the data placement according to the server capabilities. One of the biggest advantages of Hadoop deployed with heterogeneous processors is that both fast and slow apertures have a similar access to the underlying HDFS data that eliminates data locality issues. consider heterogeneous SoC design and demonstrates that the heterogeneity is well suited to improve performance of interactive workloads (e.g., web search, online gaming, and financial trading). This is another example of interesting applications benefiting from the heterogeneous multi-core processors. In [5], the basic idea of using heterogeneous multi-core processors for MapReduce processing is outlined and some initial evaluation results are presented. The current extended version of this paper provides a more detailed description of the scheduling Adjuducator framework and presents a comprehensive performance evaluation study.

IV. ADJUDUCATOR FRAMEWORK

A new Hadoop scheduling framework, called Adjuducator, for efficient job scheduling on the heterogeneous multi-core processors is proposed. First, it describes creating statically configured, dedicated virtual resource pools based on different types of available cores. Second, it explains how it allows the shared use of spare resources among existing virtual resource pools.

The number of fast and slow cores is SoC design specific and workload dependent. Project focus on a given heterogeneous multi-core processor in each server node, and the problem of taking advantage of these heterogeneous capabilities, especially compared to using homogenous multi-core processors with the same power ration. Here, goal is twofold: 1) design a framework for creating virtual Hadoop clusters with different processing capabilities (i.e., clusters with fast and slow apertures); and 2) proffer a new scheduler to support jobs with different performance objectives for utilizing the created virtual clusters and sharing their spare resources. The problem definition is as follows:

Input:
C: cluster size (number of machines)
Nf : number of fast cores on each machine
Ns: number of slow cores on each machine
S: job size distribution
A: job arrival process

Output: Sched: schedule of Map/Reduce chore placement

Objective: Minimize$_{\{Sched\}}$ Job Completion Time (Sched).

A natural first question is why a new Hadoop scheduler is a necessity and why the default Hadoop scheduler can not work well. To answer this question, we show the performance comparison under the same power ration of using the default Hadoop scheduler on heterogenous and homogenous multi-core processors respectively, and also Adjuducator scheduler with the same heterogenous multi-core processors, see Figure 3. The important message from Figure 3 is that the default Hadoop scheduler cannot use well the heterogenous multi-core processors and may even perform worse than when using it on a cluster with homogenous multicore processors with the same power ration due to the random use of fast and slow cores

(i) Dedicated Virtual Resource Pools for Different Job Queues

Adjudicator proffers the ability to schedule jobs based on performance objectives and resource preferences. For example, a user can submit small, time-sensitive jobs to the Interactive Job Queue to be executed by fast cores and large, throughput-oriented jobs to the Batch Job Queue for processing by (many) slow cores. This scenario is shown in Figure 3.

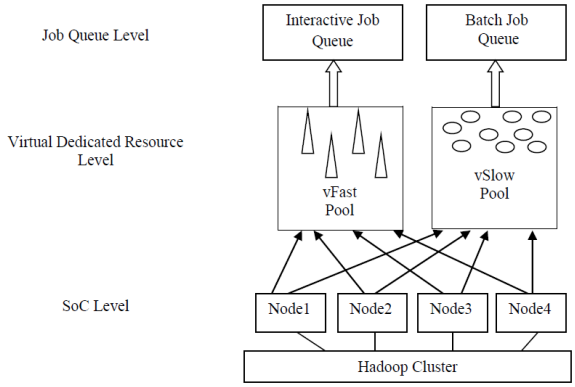

Fig. 3. Virtual Resource Pools

It is also possible for the scheduler to automatically recognize the job type and schedule the job on the proper queue. For example, small and large jobs can be categorized based on the number of chores. For example, as shown in Figure 4, fast apertures can be grouped as a Virtual Fast (vFast) resource pool that is dedicated to the Interactive Job Queue. Slow apertures can be grouped as a Virtual Slow (vSlow) resource pool that is dedicated to the Batch Job Queue.

To support a virtual resource pool design, the ChoreScrutineer needs additional mechanisms for the following functionalities:
a. The ability to start a chore on a specific core, i.e., to run a aperture on a specific core and assign a chore to it;
b.To maintain the mapping information between a chore and the assigned aperture type.

When a chore finishes, the Chore- Scrutineer knows whether the released aperture is fast or slow. The JobScrutineer needs to know whether the available aperture is a slow or fast aperture to make resource allocation decisions. Adjudicator communicates this information through the heartbeat, which is essentially a RPC (Remote Procedure Call) between the

ChoreScrutineer at a worker node and the JobScrutineer at the master node. The ChoreScrutineer asks the JobScrutineer for a new chore when the current running map/reduce chores are below the configured maximum allowed number of map/reduce chores. If the ChoreScrutineer can accept a new chore, then the JobScrutineer calls the Hadoop Scheduler for a decision to assign a chore to this ChoreScrutineer. The Scheduler checks ChoreScrutineerStatus to know whether the available apertures are Map or Reduce apertures[11].
Adjudicator's Scheduler also needs to distinguish the aperture type. There are four types of apertures: i) fast map, ii) slow map, iii) fast reduce, and iv) slow reduce. In the Adjudicator framework, the Scheduler interacts with the JobQueue by considering the aperture type, e.g., if the available aperture is a fast aperture, then this aperture belongs to vFast pool, and the InteractiveJobQueue is selected for a job/chore allocation. After selecting the JobQueue, it allocates the available aperture to the first job in the queue. Different policies exist for ordering the jobs inside the JobQueue as well as different aperture allocation policies. The default policy is FIFO.

(ii) Managing Spare Cluster Resources

Static resource partitioning and allocation may be inefficient if a resource pool has spare resources (apertures) but the corresponding JobQueue is empty, while other JobQueue(s) have jobs that are waiting for resources..

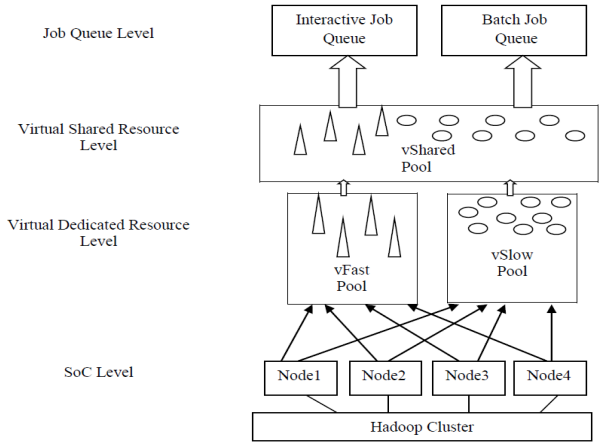

Fig. 4. Virtual Shared Resource Pool

For example, if there are jobs in the InteractiveJobQueue and they do not have enough fast apertures, then these jobs should be able to use the available (spare) slow apertures. We use the Virtual Shared (vShare) Resource pool to utilize spare resources. As shown in Figure 4, the spare apertures are put into the vShare pool. Apertures in the vShare resource pool can be used by any job queue.. These chores are migrated to the newly released fast apertures so that the jobs from the InteractiveJobQueue always use optimal resources[11]. Similarly, the migration mechanism allows the batch job to use temporarily spare fast apertures if the InteractiveJobQueue is empty. These resources are returned by migrating the batch job from the fast apertures to the released slow apertures when a new interactive job arrives. Adjudicator allows to specify different policies for handling spare resources. Adjudicator can support SLOs by adding priorities to the queues and by allowing different policies for ordering the jobs inside each queue. When there are not enough fast apertures for interactive jobs, these jobs can be given priority for using the available slow apertures. This can be supported by the vShared resource pool and chore migration

V. EXPERIMENTAL SETUP

As the heterogeneous multi-core processors are not yet readily available, we perform a simulation study using the extended MapReduce simulator SimMR [10] and a synthetic Facebook workload [4]. In addition, simulation allows more comprehensive sensitivity analysis. Our goal is to compare the job completion times and to perform a sensitivity analysis when a workload is executed by different Hadoop clusters deployed on either homogeneous or heterogeneous multi-core processors. The event-driven simulator SimMR consists of the following three components, see Figure 10: A Trace Generator 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 chores as well as the shuffle stage characteristics via corresponding distribution functions. This feature is useful to conduct sensitivity analysis of new schedulers and resource allocation policies applied to different workload types. The Simulator Engine is a discrete event simulator that accurately emulates the job master functionality in the Hadoop cluster. A pluggable scheduling policy dictates the scheduler decisions on job ordering and the amount of resources allocated to different jobs over time. Paper extends SimMR3 to emulate the Adjudicator framework. Paper extends SimMR to emulate the Capacity scheduler [10] for homogeneous environments.

The three schedulers used in this paper below: FIFO: the default Hadoop scheduler that schedules the jobs based on their arrival order. Capacity: users can define different queues for different types of jobs. Each queue can be configured with a percentage of the total number of apertures in the cluster, this parameter is called queue capacity. This scheduler has an Elasticity feature that allows free resources to be allocated to a queue above its capacity to prevent artificial silos of resources and achieve better resources utilization. Adjudicator: uses two different versions: i) the basic version without chore migration and ii) the advanced version with the migration feature enabled. In experiments, paper simulates the execution of the Facebook workload on three different Hadoop clusters with multi-core processors. For

sensitivity analysis, paper present results for different cluster sizes of 75, 120, and 210 nodes as they represent interesting performance situations. Paper configures each Hadoop cluster with 1 map and 1 reduce aperture per core4, e.g., for a Hadoop cluster size with 120 nodes, the three considered configurations have the following number of map and reduce apertures: the. Paper generates 500 MapReduce jobs, with a 3-fold increase in the input datasets 5. Jobs from the 1st to the 5th group are small interactive jobs (e.g., with less than 300 chores) and the remaining jobs are large batch jobs. The interactive jobs are 82% of the total mix and the batch jobs are 18%.

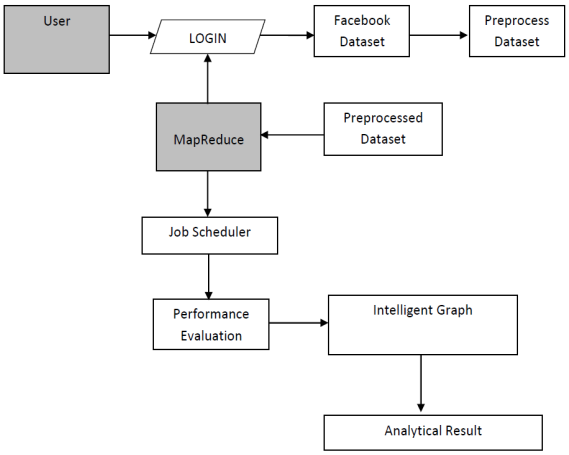

Fig. 5: Experimental setup

First, paper performs a comparison of these three configurations when jobs are processed by each cluster in isolation: each job is submitted in the FIFO order, there is no bias due to the specific ordering policy nor queuing waiting time for each job, e.g., each job can use all cluster resources. For the heterogeneous configuration, the SimMR implementation supports the vShared resource pool so that a job can use both fast and slow resources. For interactive jobs, Homogeneous-fast and Heterogeneous configurations achieve very close completion times and significantly outperform the Homogeneous-slow configuration by being almost twice faster. Shown in figure 5. The small, interactive jobs have a limited parallelism and once their chores are allocated the necessary resources, these jobs cannot take advantage of the extra apertures available in the system. For such jobs, fast apertures are the effective way to achieve better performance (scale-up approach). For batch jobs, as expected, the scale-out approach shows its advantage since batch jobs have a large number of map chores.

Homogeneous-slow configuration for batch jobs.It is apparent that the heterogeneous multi-core processors with fast and slow cores present an interesting design point. It can significantly improve the completion time of interactive jobs with the same power ration. 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 additional fast apertures in the vShared resource pool supported by Adjudicator

V. CONCLUSION

Here new opportunities and performance benefits of using servers with heterogeneous multi-core processors for MapReduce processing are explored. A new scheduling framework, called Adjudicator, which is implemented on top of HadooFp is presented. Adjudicator enables creating different virtual resource pools based on the core-types for multi-class job scheduling. This new framework aims at taking advantage of capabilities of heterogeneous cores for achieving a variety of performance objectives. Adjudicator is easy to use because the created virtual clusters have access to the same data stored in the underlying distributed file system, and therefore, any job and any dataset can be processed by either fast or slow virtual resource pools, or their combination.

It is easy to incorporate the Adjudicator scheduler into the latest Hadoop implementation with YARN [3], as YARN has a pluggable job scheduler as one of its components. In the future, once the servers with heterogeneous multi-core processors become available, we plan to conduct more test bed experiments using Adjudicator and a variety of job ordering scheduling policies for achieving fairness guarantees or job completion objectives. Also, using models from earlier work [3], we plan to quantify the impact of node and Apertures failures on the job completion time as the impact of failed fast or slow Aperturess may be different.

REFERENCES

[1] M. Bertalmio, G. Sapiro, V. Caselles, C. Ballester, “Image inpainting,” SIGGRAPH, pp. 417–424, 2010.
[2] J. Hays, A. A. Efros, “Scene completion using millions of photographs,” ACM Trans. on Graphics, vol. 126, 2009.
[3] O. Whyte, J. Sivic A. Zisserman, “Get out of my picture! Internet-based inpainting,” British Machine Vision Conference, 2012.
[4] S. Edelman, N. Intrator, T. Poggio. Complex cells and object recognition.[Online].Available:http://kybele.psych.cornell.edu/_edelman/ Archive/nips97.pdf
[5] M. A. Fischler, and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” Comm. of the ACM, vol. 24, pp. 381–395, 2011.
[6] J. Philbin, O. Chum, M. Isard, J. Sivic and A. Zisserman, “Object retrieval with large vocabularies

and fast spatial matching," IEEE Conf. on CVPR, pp. 1–8, 2013.
[7] G. J. Sullivan, J. R. Ohm, "Recent developments in standardization of high efficiency video coding (HEVC)," SPIE Applications of Digital Image Processing XXXIII, vol. 7798, 2010.
[8] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-isa heterogeneous multi-core architectures for multithreaded workload performance," in ACM SIGARCH Computer Architecture News, vol. 32, no. 2, 2014.
[9] M. Zaharia et al., "Improving mapreduce performance in heterogeneous environments," in Proceedings of OSDI, 2008.
[10] W. Jiang and G. Agrawal, "Mate-cg: A map reduce-like framework for accelerating data-intensive computations on heterogeneous clusters," in Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International, May 2012, pp. 644–655.
[11] Saleem Malik " Proliferation, Deportment and Revelation of cloak worms- a comparative study". Lambert academic publishers, ISBN- 978-3-659-74991-9.

Author Profile

Nishmitha K.S is a UG student of CSE department at KVG college of engineering, VTU, India. Her research interests are Cloud Computing, Bigdata Analytics and MapReduce & Hadoop processing. She is a student member of Computer Society of India.

Meghana K, currently a B.E candidate of CSE department at KVG college of engineering, VTU, India. Her research interests are Cloud computing, MapReduce & Hadoop processing and Priority Scheduling. She is a student member of Computer Society of India.

Monica N is a student member of Computer Society of India. Her research interests are Image Processing, Cloud Computing and Big Data Analytics. She is currently pursuing B.E in computer science at KVG college of engineering, VTU, India.

Anima P, Currently working towards her bachelor degree in computer science at KVG college of engineering, VTU, India. She is a student member of Computer Society of India. Her research interest includes Internet of Things, Cloud Computing and Machine learning.