ISSN 2395-1621

Job Scheduling Using Fair Scheduler With Single-Node and Multiple-Node Clusters

[#1]Soumitra Bhosale, [#2]Pooja Bhosale, [#3]Ketan Dadpe

[1]soumitra13.2@gmail.com
[2]poojabhosale2711@gmail.com
[3]ketandadpe@gmail.com

[#123]Student at Zeal College of Engineering, Pune, INDIA

ABSTRACT

As organizations start to use data-intensive cluster computing systems like Hadoop and Dryad for more applications, there is a growing need to share clusters between users. However, there is a conflict between fairness in scheduling and data locality (placing tasks on nodes that contain their input data). We illustrate this problem through our experience designing a fair scheduler for a 600-node Hadoop cluster at Facebook. To address the conflict between locality and fairness, we propose a simple algorithm called fair scheduling: when the job that should be scheduled next according to fairness cannot launch a local task, it waits for a small amount of time, letting other jobs launch tasks instead. We find that fair scheduling achieves nearly optimal data locality in a variety of workloads and can increase throughput by up to 2x while preserving fairness. In addition, the simplicity of fair scheduling makes it applicable under a wide variety of scheduling policies beyond fair sharing.

Categories and Subject Descriptors D.4.1 [Operating Systems]: Process Management—Scheduling.

General Terms Algorithms, Performance, Design.

ARTICLE INFO

Article History

Received :10th May 2016

Received in revised form :

13th May 2016

Accepted : 15th May 2016

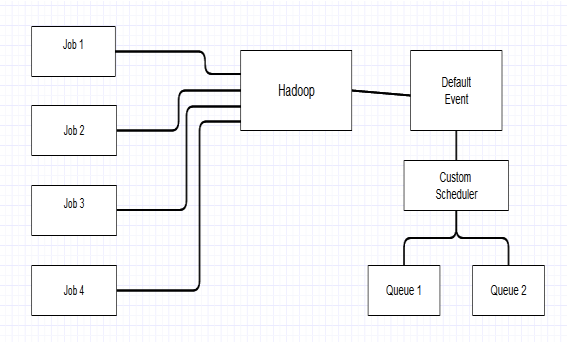
Published online :

18th May 2016

I. INTRODUCTION

Data-intensive cluster computing is increasingly important for a large number of applications including web-scale data mining, machine learning, and network traffic analysis. There has been renewed interest in the subject since the publication of the MapReduce paper describing a large-scale computing platform used at Google. One might imagine data-intensive clusters to be used mainly for long running jobs processing hundreds of terabytes of data, but in practice they are frequently used for short jobs as well. For example, the average completion time of a MapReduce job at Google was 395 seconds during September 2007. While there are large jobs that take more than a day to complete, more than 50% of the jobs take less than 30 minutes. Our users generally strongly desire some notion of fair sharing of the cluster 1 resources. The most common request is that one user's large job should not monopolize the whole cluster, delaying the completion of everyone else's (small) jobs. Of course, it is also important that ensuring low latency for short jobs does not come at the expense of the overall throughput of the system.Many of the problems associated with data-intensive computing have been studied for years in the grid and parallel database communities. However, a distinguishing feature of the data-intensive clusters we are interested in is that the computers in the cluster have large disks directly attached to them, allowing application data to be stored on the same computers on which it will be processed.Maintaining high bandwidth between arbitrary pairs of computers becomes increasingly expensive as the size of a cluster grows, particularly since hierarchical networks are the norm for current distributed computing clusters.

Architecture Diagram :



As the Figure shows we are using a custom scheduler which will contain an advance version of fair scheduler. It will help us change the queue as per the requirements of the job to be completed. By changing the queue we mean to say we will the switch the job using FAIR Non-pre-emption Scheduling. The Fair scheduler is explained below.

HADOOP FAIR SCHEDULER

Fair scheduling is a method of assigning resources to jobs such that all jobs get, on average, an equal share of resources over time. When there is a single job running, that job uses the entire cluster. When other jobs are submitted, tasks slots that free up are assigned to the new jobs, so that each job gets roughly the same amount of CPU time. Unlike the default Hadoop scheduler, which forms a queue of jobs, this lets short jobs finish in reasonable time while not starving long jobs. It is also an easy way to share a cluster between multiple of users. Fair sharing can also work with job priorities - the priorities are used as weights to determine the fraction of total compute time that each job gets.

The fair scheduler organizes jobs into *pools*, and divides resources fairly between these pools. By default, there is a separate pool for each user, so that each user gets an equal share of the cluster. It is also possible to set a job's pool based on the user's UNIX group or any jobconf property. Within each pool, jobs can be scheduled using either fair sharing or first-infirst-out (FIFO) scheduling.

We provide one feature beyond standard weighted fair sharing to support production jobs. Each pool can be given a minimum share, representing a minimum number of slots that the pool is guaranteed to be given as long as it contains jobs, even if the pool's fair share is less than this amount (e.g. because many users are running jobs). HFS always prioritizes meeting minimum shares over fair shares, and may kill tasks to meet minimum shares. Administrators are expected to set minimum shares for production jobs based on the number of slots a job needs to meet a certain SLO (e.g. import logs every 15 minutes, or delete spam messages every hour). If the sum of all pools' minimum shares exceeds the number of slots in the cluster, HFS logs a warning and scales down the minimum shares equally until their sum is less than the total number of slots. Finally, although HFS uses waiting to reassign resources most of the time, it also supports task killing. We added this support to prevent a buggy job with long tasks, or a greedy user, from holding onto a large share of the cluster. HFS uses two task killing timeouts. First, each pool has a minimum share timeout, Tmin. If the pool does not receive its minimum share within Tmin seconds of a job being submitted to it, we kill tasks to meet the pool's share. Second, there is a global fair share timeout, Tf air, used to kill tasks if a pool is being starved of its fair share.

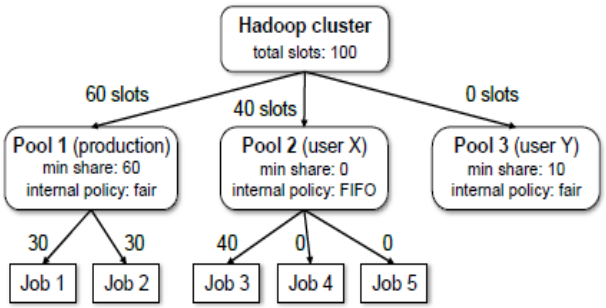


Figure : Example of allocations in HFS.

Pools 1 and 3 have minimum shares of 60 and 10 slots. Because Pool 3 is not using its share, its slots are given to Pool 2. Each pool's internal scheduling policy (FIFO or fair sharing) splits up its slots among its jobs.

We expect administrators to set Tmin for each production pool based on its SLO, and to set a larger value for Tf air based on the level of delay users can tolerate. When selecting tasks to kill, we pick the most recently launched tasks in pools that are above their fair share to minimize wasted work.

CONCLUSION

We present a new approach to scheduling with fairness on shared distributed computing clusters.We constructed a simple mapping from the fair scheduling problem to min-cost flow, and can thus efficiently compute global matching's that optimize our instantaneous scheduling decisions.

ACKNOWLEDGEMENT

We take this opportunity to thank our Head of the Department Prof. Gosul and Prof. Kanawade for providing all the necessary facilities, which were indispensable in the completion of this project report. We thank our project guide Ms.Chaitali Shewale for giving valuable guidance. We are also thankful to all the staff members of Information Technology department of Zeal College of Engineering for their valuable time, support, comments, suggestions and persuasion. We would also like to thank the institute for providing the required facilities, Internet access and important books.

REFERENCES

[1] Amazon EC2. http://aws.amazon.com/ec2.

[2] Apache Hadoop. http://hadoop.apache.org.

[3] Matei Zaharia, "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling", University of California, Berkeley, matei@berkeley.edu.

[4] Shanjiang Tang," DynamicMR: A Dynamic Slot Allocation Optimization Framework for MapReduce Clusters".

[5] Yang Wang," Budget-Driven Scheduling Algorithms for Batches of MapReduce Jobs in Heterogeneous Clouds".

[6] Michael Isard, Vijayan Prabhakaran, "Quincy: Fair Scheduling for Distributed Computing Clusters", Microsoft Research, Silicon Valley — Mountain View, CA, USA. {misard, vijayanp}@microsoft.com.

