Hybrid Datacenter Scheduling

Abhimanyu Rawat (2015H103081P), Arpit Srivastava (2015H103076P)
M.E., BITS Pilani Campus
Aug-Dec 2016

Abstract

With the emergence of large data-centers, scheduling of workload has become more challenging. Initially, centralized approach for scheduling was widely used, but it had drawbacks of sub-optimal scheduling for the small task as they were queued behind the large tasks.

Then models for fully distributed scheduling were proposed that has other problem of performing badly for large task sets as required amount of the data-center resources was not controlled by a distributed scheduler hence resources were not allocated optimally. In our model, long jobs are scheduled using the centralized scheduler while the short ones are scheduled using the fully distributed scheduler. The distinction of a long job from a short job varies based on the jobs arriving to scheduler. Along with this to avoid queuing of the short jobs behind the long ones a partition of the cluster is always reserved for the small jobs. The reserved size of the partition varies as well depending on the inclination of size of jobs coming for execution. Another feature of our model is that when a task is to be scheduled by the distributed scheduler it probes the nodes for availability, so when sending their response back the nodes could advertise their load and other parameters to facilitate the scheduling decision.

We compare our results with the state-of-the-art fully Distributed Sparrow scheduler, Hybrid scheduler Hawk and Eagle. We have evaluated our model using trace-driven simulation, using traces with mixed types of jobs with heavy load on cluster. On analysis, we were able to achieve a good percentage of improvement over other schedulers.

1 Introduction

Today's data centres are huge in both size and added performance, with the advent of high-end network devices and commodity hardware adding the parallel compute to the service. Large clusters have to deal with an increasing number of job, which can vary significantly in size and have varying latency requirement for long and short jobs. Short jobs, due to their latency sensitive nature are more affected due to delay in scheduling while the Long jobs are not affected much by latency in scheduling. Efficient scheduling of data-center for the heterogeneous loads is of increasing importance to data center operators.

The first-generation cluster schedulers that were used in Hadoop were monolithic in designed and used the centralized approach for scheduling. A centralized scheduler has the information of all the available resources in the data-center. So for taking the scheduling decisions centralized scheduler have long latency time in making a scheduling decision. This delay significantly affects the performance of the short jobs while the long jobs are not that much affected. For many of these reasons the, there is a recent movement towards the distributed scheduling, the pros and cons of distributed scheduling is exactly the opposite of the centralized scheduling, but low latency in scheduling decision they have proved to be superior for the performance of short jobs.

In our model, scheduling proposed is hybrid in nature and have the benefits of both centralized and distributed schedulers. Our model schedules the long jobs using the centralized scheduler and the small jobs using the distributed scheduler. This compensates for the cons of both type of scheduling. In addition to this it task stealing algorithm that is meant to reallocate the queued task to other sparsely loaded server for any sub-optimal scheduling decision taken by the scheduler. Along with that we label the tasks on the basis of their equability, this further improves the performance of the scheduler. While making scheduling decisions for short jobs, distributed scheduler probes the availability of the nodes, in response the nodes instead of sending a binary message can send the status which could further be used to optimize the scheduling decisions.

The rationale of this hybrid approach is that the small number of long jobs does not overwhelm the centralized scheduler. Hence scheduling latencies remain modest while large number of short jobs are scheduled by distributed schedulers hence latency for short jobs doesn't comes into picture. We have simulated our model and compared the performance with Sparrow, Hawk and Eagle scheduler. And the results are demonstrated using the freely available Google Trace and workload derived from Yahoo and Facebook.

2 Related work

First Datacenter schedulers were monolithic in design which leads to scalabilities issues[7]. Second generation schedulers (YARN, Mesos) had two level architecture which decouples resource allocation from application specific logic such as task scheduling. However, the two-level schedulers relied on a centralized scheduling and resource allocation that lead to bottleneck in large clusters. in contrast to them Hawk schedules in distributed manner mostly minimizing the scalability concerns.

Fully distributed highly scalable design such as Sparrow data center scheduler emerged, they perform well for the short jobs under loaded clusters. But its performance degrades in the case of high load and specially if the jobs are heterogeneous in nature. It is due to the design of the Sparrow which is geared to extreme scalability that is not able to draw the benefits from the load information known prior. Moreover, sparrow doesn't have a mechanism to compensate for any suboptimal decision in allocation of task.

Recent studies have moved towards hybrid solutions which uses the benefits of both worlds i.e. the scalability of the fully-distributed schedulers and the efficiency of the centralized schedulers. In Apollo, distributed scheduler utilizes global cluster information via a loosely coordinated mechanism[1]. Apollo does not differentiate in long and short jobs but schedules both of them as same. Apollo have a node level correction mechanism to compensate for the inaccurate decision made. If a task is queued longer than estimated at scheduling time, then Apollo starts duplicate copies of the task on other nodes. Another model is Mercury, which put labels on the jobs as *Guaranteed* or *Queueable*, if a job is guaranteed it will start execution as soon as the job comes to scheduler[5] while if a job is Queueable it can be queued in the ready queue of the node for execution. Mercury is primarily uses set of policies to decide the scheduling of a particular job. Hawk is yet another hybrid scheduler which is a mix of distributed and centralized schedulers for jobs categorized as short and long on the basis of mean execution time. It has employed a randomized task stealing algorithm for finding the short jobs queued behind long jobs. Hawk performs comparatively better over heavy loaded cluster for mixed type of tasks. Other proposed models are Omega.

3 Components

3.1 Data-center, Cluster and Nodes

Data-center is referred to as a place where large number of machines are working together to provide task execution service. In literature data-center is referred as provider of computing as a service. User can submit their jobs to data-center for processing. Data-centers have high end machines that are linked together to work as a combined entity, such that they can be controlled and fed in input from a limited number of channels. Cluster is a set of machines in a datacenter that are configured specifically to work in cohesion with each other by means of some software component. Example, A spark cluster has multiple machines configured to it, so they all are controlled by the same controllers and jobs are scheduled by the same scheduler.

Nodes are essentially being the machines which executes the tasks and are the part of the clusters in a data-center. Nodes are often referred as servers. A node could be a part of more than one clusters in a data-center depending upon its hardware configuration and usage.

3.2 Jobs and Tasks

A job is defined as a batch of tasks. Tasks are the individual in dependent units for execution over the cluster. A collection of units requested by a user or are similar to each other. A job can have restrictions for execution on machines with particular type of attributes like Architecture, OS version, support for specific library, GPU etc. These constrains can be soft or hard, these setting are like

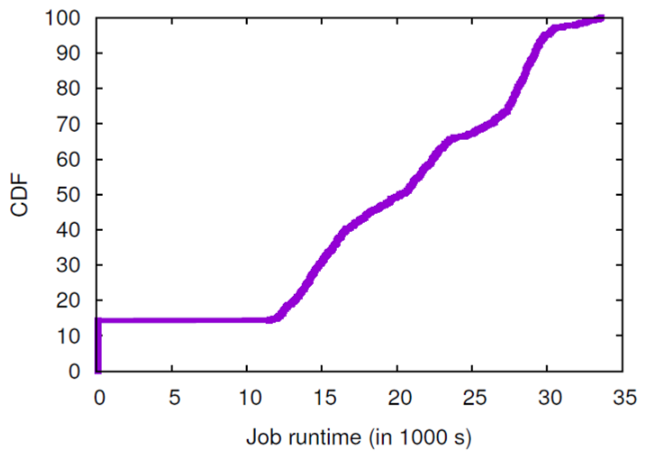

Figure 1: Load variation for Sparrow

preferences for soft type rather than requirement. The start of the job cane be differed by the scheduler at the cluster, depending upon its criticality.

Each individual task maps to a virtual machine or container for execution[6]. The cost of virtualization is not considered for the execution of the tasks. Required/requested configuration (in terms of CPU cores, Memory, TCP ports)for execution of the VM's is provided to tasks.

3.3 Workload

Workload is the amount of jobs that are coming to the data-center for execution. If the number of jobs that are coming for execution are larger then the cluster is heavy loaded. Workload as per the traces provided by Google, Facebook and others show that the nature of the workload is heterogeneous in nature i.e. it's the mixture of short and long jobs if we consider on the basis of execution time. Considering the number of jobs, long jobs comprise of very small number of total number of jobs. But these small number of once in a while coming jobs can consume large amount of data-center resource. Short jobs though large in number finish execution as soon as they arrive. From the analysis of traces of Google where number of long jobs is around 10%, but they consume around 83.65% of the total time in execution. Similar patterns for heterogeneity is observed among other traces from Cloudera, Yahoo and Facebook[4].

3.4 Aim for high utilization

With the advent of cloud based system, more and more computing is moving to the data-centers. Efficiency of the resources is dependent upon provisioning and operational cost, less the resources required for the overhead of execution

of user jobs. The main challenge for the data-center operators is to provide the acceptable level of quality of service during the peak hours of the request rates, which may overwhelm the data center.
For a single server, the main challenge is to maximize resource utilization by collocating workloads without the danger of decreasing the performance due to contention[9]. Thus, several resource scheduling and isolation mechanisms have been proposed, which ensures high performance on single server. While running multiple jobs on multiple servers have orthogonal challenges which are beyond the proposed mechanism for a single server. The problem targeted here is of scheduling multiple jobs on cluster of servers in a scalable fashion such that all resources in the clusters are efficiently used.

3.5 Challenges for cluster schedulers

We next show by means of simulation the performance of fully distributed Sparrow scheduler and centralized YARN scheduler. We observe from the simulation that the performance of the both fully distributed and fully centralized scheduler tends to be optimal for a small number of jobs but as the number of jobs stat growing and the usage of the cluster reaches the highly loaded state the performance of the above-mentioned schedulers start degrading. Centralized schedulers have high cluster utilization overall in term of amount of processing but poor in terms of number of jobs processed. As short jobs get scheduled they may be queued behind the long job hence their total execution tie has waiting time as major part which is not desirable. While on the other end of the spectrum distributed scheduler to perform decently when number of tasks are small, as the number of tasks start to increase the utilization, start to decrease, as distributed scheduler is not aware of the state of the cluster and randomly allocates the jobs to nodes hence it might happen that certain number of nodes are heavily loaded and queued while the rest are relatively free.

So, for the cluster the scheduling should be such that it it able to perform optimally for short as well as long jobs under heavy load conditions. We use the simulation used by the Sparrow paper to investigate the above stated. We consider 12000 jobs, to be simulated over 5000 nodes. 90% of which are considered to be the small jobs. Each short job has 250 tasks and each tasks takes 50 Seconds for execution 10% of the long jobs have 1000 tasks with each task taking 10000 seconds. The job submission time are derived from a Poisson distribution with mean of 50s.

We observe the cluster utilization (*i.e.* the percentage of usage of server at a point of time) per 50 sec. The median utilization came out to be 84.47% and the mean was around 95.65%. This implies that there were essentially 4% (*i.e.* 200 nodes) of cluster free at any point of time that is pretty much to more short jobs. These calculations are made keeping aside the overhead cost of running the essential tasks as we only aim to observe the scheduling cost.

4 Proposed Approach

4.1 System Model

A cluster is considered to be composed of server nodes. A job is a collection of various tasks and job completes when all the tasks of a job complete. The term Long job refers to the job having tasks that are a longer running times and vice-versa for short jobs. Each server has one queue of tasks. When a new task is scheduled on a server if that server has a task already running then the new task is queued at the end of the queue. The server queue management is FIFO.

4.2 Model

Hawk has the following 5 goals:
1. to execute jobs on cluster at high utilization
2. to enhance the performance of the short jobs which are most penalized ones in the highly-loaded cluster
3. to sustain or improve the performance of long jobs.
4. to overcome any suboptimal decision taken during scheduling of jobs
5. provide the user some control on the scheduling of his jobs

To fulfil these requirements, our model relies on the following strategies. To improve the performance, the head-of-line blocking must be avoided for this combination of 5 techniques is used. First, it reserves a small portion of the cluster for the for the short jobs whose size varies with the recent jobs. Second, to maintain a low latency scheduling decisions. Finally, Hawk relies on centralized scheduling for long jobs to maintain good performance for them, even in the face of reserving a part of the cluster for short jobs. The rationale of this choice is to get a better scheduling decision for long jobs.

4.3 Differentiating long and short jobs

The main idea behind this is to process the long and short job differently. We need to draw the line between the long jobs and short jobs. For this purpose, our model uses a $function$ which decides the cutoff time of the tasks in a jobs which is based upon the heuristics of the past few task sizes in the job. Average task execution time serves as a relatively robust for prediction about the job. The cut-off of long and short jobs is varied dynamically for 10 percent variation of the current task size.

4.4 Splitting the Cluster

We reserve a portion of the cluster for the execution of the short jobs. The partitions are termed as short partition and general partition. The long jobs are scheduled on the general partition by the centralized scheduler and the short jobs can be scheduled on any of the partition by the distributed scheduler. The restriction here is that the long jobs cannot be scheduled on the partition reserved for the short job *i.e.* short partition. This is done to avoid short job being queued behind long job and waiting for its completion. We keep the size of the partition variable and is dependent upon the heuristics applied on the basis of the last few tasks which came in.

4.5 Scheduling Short Jobs

Our model maintains a low latency scheduling for the short jobs using the distributed approach. Each short job is scheduled using a different instance of distributed scheduler for scalability reason, these scheduler instances have no knowledge of current cluster state or nor do they interact with the other schedulers or with the current centralized component.
Distributed scheduler schedule the tasks on the entire cluster without consideration of the partition scheme that we have in place. The first scheduling step is achieved as in Sparrow. To schedule n tasks probes are sent to $2n$ nodes. When a probe comes back to the scheduler confirming the availability of the node, once the number of probe responses reach n the task is scheduled on the nodes that responded.

4.6 Task Stealing Algorithm

Our model uses heuristic based task stealing algorithm for compensation for sub-optimal decisions taken during scheduling of the short jobs on a highly-loaded cluster. In a highly loaded cluster the probability of a task getting queued is very high, probe send to a server is most likely to queue the task. So in order to improve the performance by reducing the queuing of the tasks Hawk have an implementation of Randomized task stealing algorithm. This algorithm probes the servers which have tasks queued and if the queues are present it tries to reassign the task to some server which is lightly loaded. So, this avoid long queues at certain nodes in the cluster while some other nodes are not that heavily loaded. As an optimization node in general partition executing the long jobs are considered for queuing check. After a long job have arrived, the next coming several small jobs are allocated by the distributed scheduler are the main targets of the task stealing algorithm task are stolen from any of the queues of the loaded severs and from the head of the queues, it is assumed that head of the queue has waited the longest and should be executed first. By reducing the queuing, the performance of Hawk model increases considerably.

4.7 Scheduling Long Jobs

Long Jobs are scheduled using a centralized scheduler and only on the general partition and the centralized scheduler have no knowledge where the short components are being executed. The centralized approach ensures good performance for three reasons. Firstly, number of long jobs are is small, so centralized component is unlikely to become a bottleneck. Secondly , long jobs have large latency bounds, so they are largely unaffected by the moderate scheduling latency. Thirdly, by scheduling the jobs centrally and by the fact that these long jobs take up the large fraction of the cluster resources, so a fairly accurate view of the per node usage of is available.

4.8 Advertising the Load

To further improve the performance of scheduling, we have mechanism to optimize the scheduling decisions to be taken. While scheduling the short jobs using distributed scheduler. So, when the probes are sent to the nodes for advertising their availability so instead of just sending binary data, nodes can send load statistics to the scheduler so that it can take even better decisions.

5 Experimentation

5.1 Setup

We have used trace driven simulation for the demonstration of our modified/patched Hawk[4] model. We have used the simulation used by the original Hawk[4] paper for experimentation by augmenting our model to it.

5.2 Comparison of Results

We have compared our results with the state-of-the-art fully distributed Sparrow scheduler, Hybrid scheduler Hawk. We have shown that for loaded clusters our model outperforms Hawk and Sparrow for both short and long jobs. The dynamic varying of the parameters allows our model to hold advantages across all the workloads. We have also compared our model to centralized scheduler.

5.3 Workloads

We have used the openly available Google Trace [10, 8], we have filtered the failed jobs out and we are left with 50613 jobs. Task duration vary for a given job, the estimated task duration of the job is the average of the duration of all the jobs.

We have created additional traces using the similar approach taken by [4] *i.e.* using the description of workloads from Yahoo 2011 form [3] and Cloudera and Facebook 2010 workloads from [2]. In [3] [2] workloads are in form of k-means clusters.

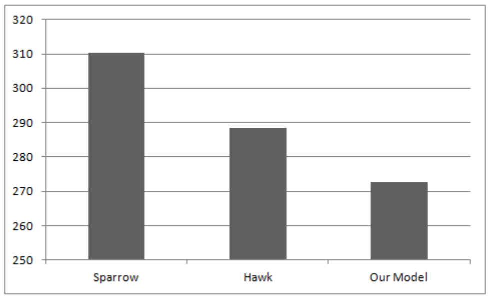

Figure 2: Varying the cutoff

5.4 Varying the task cutoff dynamically

Hawk proposed of using a fixed cutoff for the long and short jobs. This approach works well for high load cluster with heterogeneous mixed tasks. Its performance degrades slightly for jobs whose execution time lies very close to the cutoff value. Additionally it is not able to take the advantage of the state information of the type of jobs coming in for execution. So as to take the advantage of the type of task coming for scheduling, we apply a heuristic algorithm that considers the execution time of the past few tasks and vary the cutoff on the basis of it.

5.4.1 Task cutoff varying algorithm

Our algorithm keeps the execution time of the last few jobs in a list L. From L we take the last 5 jobs that came in for execution and average them out avg. If the average is varying more than 10% of our previously set cutoff. Then we set the avg as the new cutoff.

5.4.2 Analysis

We got 8% to 10% improvement in comparison to Hawk on heavily loaded clusters. The reason for this improvement is that the cutoff value for short and long job varies as per the sizes of the coming jobs. Cutoff increases if the job size tends to increase continuously and vice versa. One important observation is that the number of task stolen by the task stealing algorithm decreases. This happens because now the cutoff moves as per the job sizes, so they are scheduled more optimally in first place so less number of jobs are available for stealing. Similar observations are recorded across multiple runs with small variations across all the traces.

5.5 Varying the partition size dynamically

Hawk uses a fixed size partitioning for reserved and general partitions. This has a drawback, being static the size of partition cannot vary across as necessary as per the density of jobs in the long or short ranges. In our model, we have

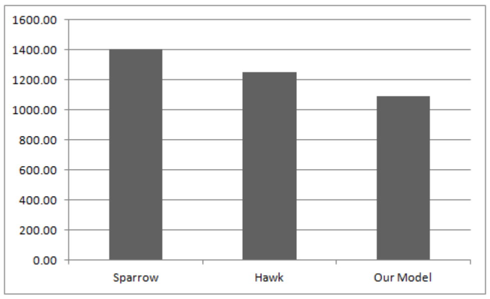

Figure 3: Varying the partition size

added the provision for varying the partition size dynamically dependent upon the type recent number of job types short or long. As the number of short jobs increase the size of the reserved partition increases as well hence more number of short jobs can be scheduled through the reserved partition, further reducing the probability of queueing. On the other side if the job size increases more number of nodes will be made available to the general partition for queueing of big jobs reducing their overall execution time.

5.5.1 Partition varying algorithm

Our algorithm varies the size of the partition as per the movement of the job cutoff. So as per the algorithm, 1 or 2 job cutoff varies on the basis that the size of the reserved partition is varied. So, for every variation in the cutoff we vary the size of the reserved partition. Thus, if the cutoff is reduced form the past cutoff value, then the size of the reserved partition is increased while if the cutoff increases, then the size of reserved partition decreases by 2.

6 Downsides

Though as we have shown that our model performs better than the Hawk and Sparrow. But it has it's own underlying downsides. As we are keeping on differentiating the cutoff and reserved partition size as dynamic, it's performance become prone to absurd variations in the job sizes. Any variations destabilizing the cutoff will hamper the overall utilization of the cluster. Our model can tolerate smooth moving of the cutoff value but sudden changes are not responded. This will only be the problem if the job size varies in such a way that it moves the cutoff adequate to it and then the next job of opposite polarity and it again, tries to move the cutoff to its levels.

7 Conclusion

There significant performance improvement over the fully distributed scheduler Sparrow is improvement due to the long jobs which sparrow couldn't schedule optimally as any of the distributed scheduler is not aware of all the available resources. If the jobs were to be of either type Long or Short Hawk would have not outperformed the other approaches. So, if we know the type of jobs that will be scheduled on the cluster then using Centralized or Distributed approach. But if the type of jobs is not known priory then Hawk will provide better scheduling than either if the two approaches assuming the jobs are heterogeneous.

References

[1] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. Apollo: scalable and coordinated scheduling for cloud-scale computing. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 285–300, 2014.

[2] Yanpei Chen, Sara Alspaugh, and Randy Katz. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *Proc. VLDB Endow.*, 5(12):1802–1813, August 2012.

[3] Yanpei Chen, Archana Ganapathi, Rean Griffith, and Randy Katz. The case for evaluating mapreduce performance using workload suites. In *2011 IEEE 19th annual international symposium on modelling, analysis, and simulation of computer and telecommunication systems*, pages 390–399. IEEE, 2011.

[4] Pamela Delgado, Florin Dinu, Anne-Marie Kermarrec, and Willy Zwaenepoel. Hawk: Hybrid datacenter scheduling. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 499–510, 2015.

[5] Konstantinos Karanasos, Sriram Rao, Carlo Curino, Chris Douglas, Kishore Chaliparambil, Giovanni Matteo Fumarola, Solom Heddaya, Raghu Ramakrishnan, and Sarvesh Sakalanaga. Mercury: Hybrid centralized and distributed scheduling in large shared clusters. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 485–497, 2015.

[6] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. A study of skew in mapreduce applications. *Open Cirrus Summit*, 2011.

[7] Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. Sparrow: distributed, low latency scheduling. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 69–84. ACM, 2013.

[8] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA, November 2011. Revised 2012.03.20. Posted at http://code.google.com/p/googleclusterdata/wiki/TraceVersion2.

[9] Eno Thereska, Hitesh Ballani, Greg O'Shea, Thomas Karagiannis, Antony Rowstron, Tom Talpey, Richard Black, and Timothy Zhu. Ioflow: A software-defined storage architecture. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 182–196, New York, NY, USA, 2013. ACM.

[10] John Wilkes. More Google cluster data. Google research blog, November 2011. Posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.