International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426

An Improved Scheduling Technique based on Containers

Naina Solanki

Assistant Professor in Computer Science and Engineering Department, Sri Satya Sai University of Technology & Medical Science, Sehore, Madhya Pradesh, India

Abstract: ***In hadoop, job scheduling is an independent module so that users can design or configure their own job scheduler based on their actual application requirements; thereby meet their specific business needs. Currently, hadoop has three schedulers: first is FIFO, Fair scheduler and Capacity Scheduler, all the schedulers can helps in scheduling the resources among different computers. All the three schedulers cannot fully support data locality due to which the performance is affected. In this paper, we took the concept of resource-prefetching into consideration, and proposed a job scheduling algorithm based on data locality. By which the PRISM scheduler will perform much better because along with data locality we also focus on containers by which we can schedule the containers among different map-reduce task.***

Keywords: Hadoop; scheduling; data locality; resources-prefetch, containers

1. Introduction

With the popularizing of Internet technology, whether it is business or personal generated data are in the rapid growth. Researches aim at how to effective and efficient mining useful knowledge from big-data to satisfied different business requirements had made a lot of achievements. The advent of the era of big-data, are making hadoop and Map-Reduce processing framework becoming increasingly popular, many companies and researchers are keen to study hadoop to meet their specific business needs. As one of the core technologies of hadoop, Map-Reduce job processing framework and job scheduling algorithm play a vital role in the overall performance of hadoop. In the dynamic task scheduling and resources allocation policies of hadoop, the input data will be cut into several pieces to storage on each node, and each node keep three copies in default. How to ensure that the data blocks needed is just located in the compute node within different tasks of a job during operation, and improve the utilization of system resources and efficiency of job operation, namely how to ensure good data locality, has become a hot issue in recent years.

Mapreduce in s open source framework which is develop and use by goggle for processing large amount of data and Apache hadoop is also a open source framework for storing and processing large amount of data, for storage purpose it uses HDFS and for processing it uses Map reduce. Hadoop works on cluster which is made by commodity hardware for storing and processing purpose, many companies uses Hadoop cluster like Facebook, twitter amazon. In mapreduce the data are storing as a block in hdfs and mapreduce is work on two phases mapper and reducer, the mapper works independently and parallel to achieve parallelisms, at the map side all the mapper work parallel and send their intermediate result to reducer which combines all the intermediate result and generate actual output. In this process scheduler plays a important role which help us to avoid sending unnecessary data to reducer. In our paper we work on data locality by which we can schedule the task according to the data machine on which machine the data is stored and it will schedule the task on that machine through which we can avoid unnecessary data transmission and reduce network traffic. If the data locality is low then the network traffic cost is very high because every time we need to move the data from one node to another node. In existing Mapreduce technique comes with by default scheduler which is FIFO(first in first out) ,All the task are schedule on the basis of their arrival in the pool there is no concept of priority because if higher priority task in coming we. Zaharia et al. [5] have developed a delay technique by which the data locality rate is improved, in which the scheduler can delay the resource allocation through which it is easier to seen that what what data resides in the map node and then its assign a task or launch a task to the map node by achieving data locality. But its take early time by delaying allocating resources to achieve a data locality.

1.1 Containers

At the fundamental level, a container is a collection of physical resources such as RAM, CPU cores, and disks on a single node. There can be multiple containers on a single node (or a single large one). Every node in the system is considered to be composed of multiple containers of minimum size of memory (e.g., 512 MB or 1 GB) and CPU. The Application Master can request any container so as to occupy a multiple of the minimum size. A container thus represents a resource (memory, CPU) on a single node in a given cluster. A container is supervised by the Node Manager and scheduled by the Resource Manager. Each application starts out as an Application Master, which is itself a container (often referred to as container 0). Once started, the Application Master must negotiate with the Resource Manager for more containers. Container requests (and releases) can take place in a dynamic fashion at run time. For instance, a MapReduce job may request a certain amount of mapper containers; as they finish their tasks, it may release them and request more reducer containers to be started.

1.2 Node Manager

The node manager is an agent which takes care of single node with in a hadoop cluster. The main duties of node manager is to keep update resource manager and monitoring resource usage like memory, CPU usage and its manages by different YARN applications. On start-up, the Node Manager registers with the Resource Manager; it then sends heartbeats with its status and waits for instructions. Its primary goal is to manage application containers assigned to it by the Resource Manager. YARN containers are described by a container launch context (CLC). This record contains environment variables, dependencies that are stored in remotely accessible storage, payloads for Node Manager Services, and the command necessary to create the process. The authencity of the container is validated after that, the Node Manager configures the environment variable for the container, including initializing its monitoring subsystem with the resource constraints' specified application. The Node Manager also kills containers as directed by the Resource Manager.

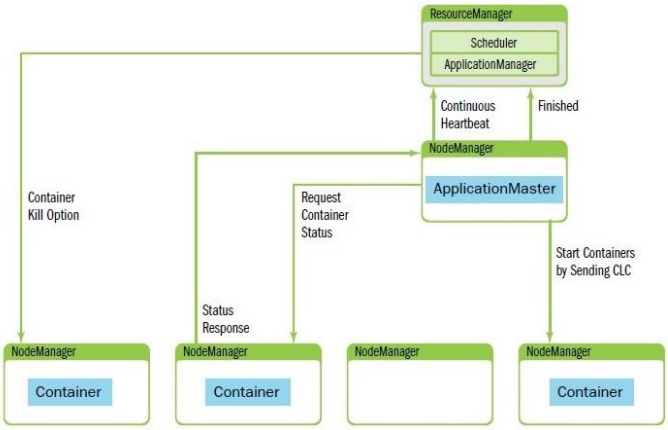
1.3 ApplicationMaster

The ApplicationMaster is the process that coordinates an application's execution in the cluster. Each application has its own unique ApplicationMaster, which is tasked with negotiating resources (containers) from the ResourceManager and working with the NodeManager(s) to monitoring and execution of the tasks. The YARN framework can uses a map-reduce as a generic support ot him, this design permits building and deploying multiple distributed applications with the help of other frameworks. Once the ApplicationMaster is started (as a container), it will periodically send heartbeats to the ResourceManager to affirm its health and to update the record of its resource demands. After building a model of its requirements, the Application Master encodes its preferences and constraints in a heartbeat message to the Resource- Manager. In response, the Application Master will receive a lease on containers bound to a allocated resources at a particular node in the cluster. Depending on the containers it receives from the Resource Manager, the Application Master may update its execution plan to accommodate the excess or lack of resources. Container allocation/deallocation can take place in a dynamic fashion as the application progresses.

1.4 YARN Model

In the earlier versions of HADOOP, each node in the cluster was statically assigned the running capability of a predefined number of map slots and a predefined number of reduce slots. The slots could not be shared between maps and reduces. The static allocation of map task and reduce task slot cannot optimised the performance because in the starting of every task we cannot predict how many map and reduce task will be sufficient to handle the test or to allocate the slots. The resource allocation model in YARN addresses the inefficiencies of static allocations by providing for greater flexibility. As described previously, resources are requested in the form of containers, where each container has a number of no static attributes. YARN currently has attribute support for memory and CPU. The generalized attribute model can also support things like bandwidth or GPUs. In the future resource management model, only a maximum or a minimum for each attribute are defined, and the Application Managers can request for containers with attribute values as multiples of the minimum.

Application Master–Container Manager Communication
At this point, the Resource Manager has handed off control of assigned Node Managers to the Application Master. The Application Master will independently contact its assigned node managers and provide them with a Container Launch Context that includes environment variables, dependencies located in remote storage, security tokens, and commands needed to start the actual process (refer to Figure 4.3). When the container starts, all data files, executables, and necessary dependencies are copied to local storage on the node. Dependencies can potentially be shared between containers running the application.



Once all containers have started, their status can be checked by the Application- Master. The ResourceManager is absent from the application progress and is free to schedule and monitor other resources. The ResourceManager can direct the Node Managers to kill containers. Expected kill events can happen when the ApplicationMaster informs the Resource Manager of its completion, or the Resource-Manager Needs nodes for another application, or the container has exceeded its limits. When a container is killed, the NodeManager cleans up the local working directory. When a job is finished, the ApplicationMaster informs the ResourceManager that the job completed successfully. The ResourceManager then informs the NodeManager to aggregate logs and clean up container-specific files. The NodeManagers are also instructed to kill any remaining containers (including the ApplicationMaster) if they have not already exited.

Application Dependencies
Containers have dependencies on files for execution, and these files are either required at start-up or may be needed one or more times during application execution. For example, to launch a simple Java program as a container, we need a collection of classes and/or a file and potentially more jar files as dependencies. Rather than forcing every application for either access the files remotely and manages all these files by themselves. YARN helps the applications to localize these files by giving ability of data localization.
When starting a container, an Application Master can specify all the files that a container will require and,

therefore, that should be localized. After the files are specified the YARN helps in data localization by hiding all the complication involves in copying, managing and deleting these files.

2. Background

YARN has a pluggable scheduling component. Depending on the use case and user needs, administrators may select a simple FIFO (first in, first out), capacity, or fair share scheduler. The scheduler class is set in yarn-default.xml. Information about the currently running scheduler can be found by opening the Resource Manager web UI.

FIFO Scheduler
The original scheduling algorithm that was integrated within the Hadoop version 1 JobTracker was called the *FIFO* scheduler, meaning "first in, first out." The FIFO scheduler is basically a simple "first come, first served" scheduler in which the Job-Tracker pulls jobs from a work queue, oldest job first. Typically, FIFO schedules have no sense of job priority or scope. The FIFO schedule is practical for small workloads, but is feature-poor and can cause issues when large shared clusters are used.

Capacity Scheduler
The Capacity scheduler is another pluggable scheduler for YARN that allows for multiple groups to securely share a large Hadoop cluster. Developed by the original Hadoop team at Yahoo!, the Capacity scheduler has successfully been running many of the largest Hadoop clusters. To use the Capacity scheduler, an administrator configures one or more queues with a predetermined fraction of the total slot (or processor) capacity. This assignment guarantees a minimum amount of resources for each queue.

The Capacity scheduler permits sharing a cluster while giving each user or group certain minimum capacity guarantees. These minimums are not given away in the absence of demand. Excess capacity is given to the most starved queues, as assessed by a measure of running or used capacity divided by the queue capacity. Thus, the fullest queues as defined by their initial minimum capacity guarantee get the most needed resources. In the idle capacity scheduler we can assign a queue definition property by which we can distribute the resource among the queues. There are multi running queue in the capacity scheduler we just assign a percentage of usage of map slots and reduce slots respectively. All the queue can work under the capacity limit if the other task is completed or the other queue is empty then only its uses the whole capacity of resources.

The Capacity schedule supports memory-intensive applications, so the application can optionally specify higher memory resource requirements than the default. Using information from the node Managers, the Capacity scheduler can then place containers on the best-suited nodes. The Capacity scheduler works best when the workloads are well known, which helps in assigning the minimum capacity. For this scheduler to work most effectively, each queue should be assigned a minimal capacity that is less than the maximal expected workload. Within each queue, multiple applications are scheduled using hierarchical FIFO queues similar to the approach used with the stand-alone FIFO scheduler.

Fair Scheduler
Fair scheduler is another pluggable scheduler in hadoop that provides another sharing of resources between multiple nodes in a cluster. In fair scheduler all the application can have an equal distributed resource that means all the application can have equal number of resources. In the Fair scheduler model, every application belongs to a queue. YARN containers are given to the queue with the least amount of allocated resources. Within the queue, the application that has the fewest resources is assigned the container. By default, all users share a single queue, called "default." If an application specifically lists a queue in a container resource request, the request is submitted to that queue. When the fair scheduler in configure for hadoop then we first assign the name for the queue in the the task is arrived and we can distribute equal amount of resources between the queue. The the task with in the queue can work as a FIFO policy or in a pre-emptive manner and the task can equally share the resources. When the more priority task is assigned in the queue then the fair scheduler can prompt the task first fairly and gives required resources to it. The Fair scheduler also applies the notion of pre-emption, whereby containers can be requested back from the Application Master. Depending on the configuration and application design, pre-emption and subsequent assignment can be either friendly or forceful.

By providing fair sharing, the Fair scheduler allows minimum shares to be assigned to the queues, which is very useful for ensuring that certain users, groups, or production applications always gets sufficient resources. When a queue contains waiting applications, it gets at least its minimum share; in contrast, when the queue does not need its full guaranteed share, the excess is split between other runnings Applications To avoid a single user flooding the clusters with hundreds of jobs, the Fair scheduler can limit the number of running applications per user and per queue through the configurations file. Using this limit, user applications will wait in the queue until previously submitted jobs finish. The YARN Fair scheduler allows containers to request variable amounts of memory and schedules based on those requirements. Support for other resource specifications, such as type of CPU, is under development. To prevent multiple smaller memory applications from starving a single large memory application, a "reserved container" has been introduced. If an application is given a container that it cannot use immediately due to a shortage of memory, it can reserve that container, and no other application can use it until the container is released. The reserved container will wait until other local containers are released and then use this additional capacity (i.e., extra RAM) to complete the job. One reserved container is allowed per node, and each node may have only one reserved container. The total reserved memory can be shown in the Resource Manager UI. A new feature in the YARN Fair scheduler is support for hierarchical queues. Queues may now be nested inside other queues, with each queue splitting the resources allotted to it among its sub queues in a fair scheduling fashion. One use of hierarchical queues is to represent organizational

boundaries and hierarchies. For example, Marketing and Engineering departments may now arrange a queue structure to ref lect their own organization. A queue can also be divided into sub queues by job characteristics, such as short, medium, and long run times. The Fair scheduler works best when there is a lot of variability between queues. Unlike with the Capacity scheduler, all jobs make progress rather than proceeding in a FIFO fashion in their respective queues.

3. Related Work

Recently the importance of the MapReduce clusters has been increase rapidly, many number of organization and institution will uses a MapReduce cluster so the studies of MapReduce schedulers also increase due to which we can schedule the task between the cluster. MapReduce clusters can deal with node failures automatically. If a node fails to give a heartbeat within a timeout period, a MapReduce cluster will re-schedule the node's tasks to different nodes. By default hadoop follows speculative execution means when the task execution at any node is slow then we can launch the task or copy the task to other node also , so the multiple node will execute the task simultaneously and the node will give result first it will take and other node execution is forcefully stopped. Google has announced that this mechanism can improve a job's response time by 44% [1]. The Hadoop scheduler implicitly assumes that the cluster nodes are homogeneous in nature and tasks can make progress linearly, but on the basis of this assumptions it is difficult to speculatively re-execute the task within the cluster and here the problem appear like a stragglers [9]. To overcome this limitation of scheduler and make the speculative execution mechanism effective in heterogeneous environments, researchers then developed another technique LATE (Longest Approximate Time to End) scheduler [9] and SAMR (Self Adaptive Map Reduce Scheduling) algorithm [10]. Yahoo! developed a multi-queue scheduler called Capacity Scheduler [11] for Hadoop clusters, where every queue is guaranteed a fraction of the capacity. In the capacity scheduler we can assign a queue definition property by which we can distribute the resource among the queues. There are multi running queue in the capacity scheduler we just assign a percentage of usage of map slots and reduce slots respectively. All the queue can work under the capacity limit if the other task is completed or the other queue is empty then only its uses the whole capacity of resources.

The fair scheduler [14] also supports multiple queues (also called pools) Jobs are organized into pools and resources are fairly divided between these pools. There are multiple pools or queue in the cluster which get equal shares of the total resources. The jobs can be schedule either in FIFO manner or by fair sharing. In FIFO scheduling the jobs which arrives first will take all the available resources and after completion of job it will release the resources. But in the fair scheduler all the jobs can take equal number on resources among the queue so all the jobs can executed parally by taking some resources and waiting for other job to finish and release the resources. And in Hadoop by default scheduler is FIFO scheduler in which there is no concept of preemption and the job which is first in the queue will take all the resources and after completion release the resources. It is also an effective way of scheduling resources between multiple modes in the cluster. [14].

To improve MapReduce based clusters' data locality, researchers can studies and developed some other technologies like prefetching [15] or node status prediction [8]. The one that is most closely related to our work is the delay scheduling algorithm [5], which was first developed to improve the data locality of Hadoop fair scheduler [14]. Some MapReduce applications will comes with deadlines. J. Polo et al. [12] is developed a scheduler which is focuses on MapReduce jobs that have soft deadlines. It estimates jobs' execution times and tries to let jobs satisfy their deadlines by scheduling resources according to the estimated finishing times. Kamal Kc et al. [13] created a scheduler that works for MapRedeuce applications with hard deadlines. It also estimates the job finishing time according to current resources in a MapReduce cluster. The difference is if a job cannot finish before the hard deadline, the scheduler will not execute the job and will instead inform the user to adjust the job deadline.

4. Proposed Work

In this paper, we proposed new features through which the performance of PRISM scheduler will be improved. We improve the fine grain resource allocation in hadoop 2.0 version along with containers management by which we can achieve data locality also through which the performance of the scheduler is better.

References

[1] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". Commun. ACM, 51(1):107–113, 2008.
[2] Apache Hadoop. http://hadoop.apache.org.
[3] Amazon EC2. http://aws.amazon.com/ec2/
[4] M.C. Schatz, "BlastReduce: high performance short read mapping with MapReduce". http://www.cbcb.umd.edu/software/blastreduce/.
[5] M. Zaharia et al. "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling". In EuroSys, 2010.
[6] HDFS. http://hadoop.apache.org/hdfs/
[7] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Job scheduling for multi-user mapreduce clusters," EECS Department, University of California, Berkeley, Tech. Rep., Apr 2009.
[8] X. Zhang, Z. Zhong, S. Feng, B. Tu, J. Fan, "Improving Data Locality of MapReduce by Scheduling in Homogeneous Computing Environments", in 9th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA), pp. 120-126, 2011.
[9] M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz, I. Stoica, "Improving MapReduce performance in heterogeneous environments", in: Proc. 8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, San Diego,USA, Dec. 2008.
[10] Q. Chen, D. Zhang, M. Guo, Q. Deng, and S. Guo, "SAMR: A selfadaptive MapReduce scheduling

algorithm in heterogeneous environment", in 10th IEEE International Conference on Computer and Information Technology (CIT'10), pp. 2736–2743, 2010.
[11] Capacity Scheduler http://hadoop.apache.org/common/docs/r0.19.2/capacity_scheduler.html
[12] J. Polo, D. Carrera, Y. Becerra, J. Torres, E. Ayguade and, M. Steinder, and I. Whalley, "Performance-driven task co-scheduling for mapreduce environments," in Network Operations and Management Symposium (NOMS), 2010 IEEE, 2010, pp. 373 –380.
[13] K. Kc and K. Anyanwu, "Scheduling hadoop jobs to meet deadlines," in 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 388 –392, 2010.
[14] Fair Scheduler, http://hadoop.apache.org/mapreduce/docs/r0.21.0/fair_scheduler.html
[15] S. Seo, I. Jang, K. Woo, I. Kim, J.-S. Kim, and S. Maeng. "HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment". In Proc. CLUSTER'10, pp. 1–8, 2009.