Shortest Processing Time First and Hadoop

Laurent Bobelin, Patrick Martineau, Haiwu He

Shortest Processing Time First and Hadoop

Laurent Bobelin*, Patrick Martineau*, Haiwu He†
* LI, Université François Rabelais, Tours, France
Email:firstname.lastname@univ-tours.fr
†CSTNET/CAS, China
Email: haiwuhe@cstnet.cn

***Abstract*—Big data has revealed itself as a powerful tool for many sectors ranging from science to business. Distributed data-parallel computing is then common nowadays: using a large number of computing and storage resources makes possible data processing of a yet unknown scale. But to develop large-scale distributed big data processing, one have to tackle many challenges. One of the most complex is scheduling.**

As it is known to be an optimal online scheduling policy when it comes to minimize the average flowtime, Shortest Processing Time First (SPT) is a classic scheduling policy used in many systems. We then decided to integrate this policy into Hadoop, a framework for big data processing, and realize an implementation prototype. This paper describes this integration, as well as tests results obtained on our testbed.

I. Introduction

Interest into Big Data processing is constantly growing since a few years. Its widespread use and its constant need of growth in terms of scale has lead many researchers to pay attention to it. Lots of tools have been produced to handle big data and tackle the various challenges implied by manipulating it: Distributed File Systems, deployment and management, data import/export, SQL or NoSQL processing, Machine Learning, etc.

Apache Hadoop is one of the major tools used nowadays for Big Data distributed processing. Early releases of Hadoop were limited to an efficient implementation of MapReduce (MR), a programming framework dedicated to bid data. At that time, the scheduler architecture was a weakness of Hadoop: it did not scale well enough due to an over stressed single point of failure, in charge of any scheduling decisions [1]. Since the identification of this problem, researchers community have massively proposed algorithms and architectures to alleviate this component. Lesson learned from initial architecture has led Apache to choose a new architecture named YARN [2] that clearly separates allocation and scheduling decisions depending of if this decision is cluster-wide (i.e. fair share of resources between users) or application-wide (which task should be scheduled and where). Applications have then now to be written using the master/slave paradigm and the master is responsible for application-wide scheduling decisions. This architecture change also let Hadoop become a generic middleware able to run any kind of jobs instead of being able to cope only with MapReduce job. As a side-effect of this architecture clean-up, tools associated with Hadoop comes now as a complete ecosystem, each tool having a clear separated role within the whole cluster operation.

Nowaday Hadoop also comes with a mature set of scheduling algorithms that scale more efficiently than before. On the allocation side, well-known algorithms as FIFO or fair-share based algorithms are included in Hadoop, and the number of possibilities offered by the whole architecture in terms of scheduling grows fast, each release adding new features.

Shortest Processing Time First (SPT) is a well-known rule used in the field of job-shop scheduling [3] known to be optimal if the objective is to minimize the average flowtime. Its principle is to order jobs according to their duration and schedule them by beginning by the shorters. As the new architecture induces an extra cost strongly related to flowtime - the application master that spend resources as long as the application is not over - minimizing average flowtime seems to be a interesting way to optimize resource consumption. We then decided to investigate possible integration of SPT mechanism into Hadoop YARN scheduling.

This paper presents our integration choices, our prototype implementation of SPT into Hadoop, as well as experimental results obtained on our cluster. The rest of this paper is organized as follows. In SectionII we present YARN architecture and its main features. Then in Section III we motivate our work by showing gains expected from integration of SPT into Hadoop, as well as giving a broad description of our implementation. Then in section IV we give results, as well as analysis of why results are so mixed, and compare it to state-of-the-art algorithms in section V. We finally conclude in section VI.

II. Hadoop Scheduling Overview

Hadoop see computing and storage resources as nodes belonging to a cluster, as pictured in figure 1. Each node belongs to a group named *rack* because it merely reflects what a rack is in a data center: two nodes belonging to the same rack are neighbors, and so communications should be more efficient between those two nodes than between two nodes not belonging to the same rack. Each node has an amount of resources (CPU or RAM) that can be used to process data. Computing resource is modeled by vCore, an entity that most of the time is similar to core in a multicore architecture (Hadoop v1 allowed to do some time-share on a core by the possibility of defining computing slot - this is now strongly deprecated).

Hadoop usually relies on HDFS, a distributed file systems dedicated to big data storage. Each time data is stored, Hadoop

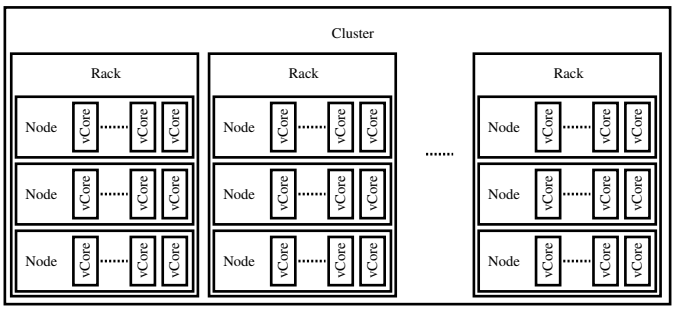

Fig. 1. Hadoop cluster

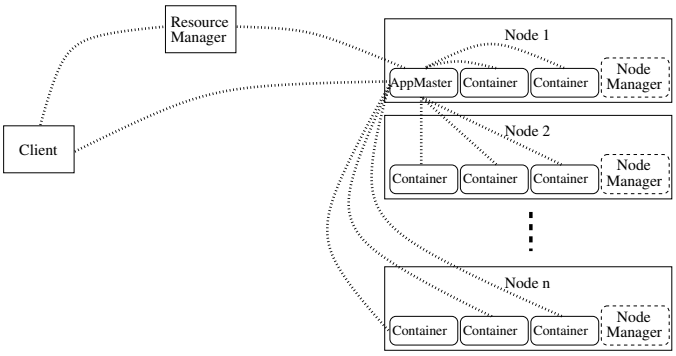

Fig. 2. Hadoop submission and run of an application

splits files into large blocks and distributes them across nodes in a cluster of resources. It uses replication of blocks across the racks to statistically give high probabilities that at least one replica may be accessed easily, and thus ensure that if any task needs this block it will be executed on a node near to the replica. Since Hadoop v2 it is possible to use other distributed file systems, but however most of them uses the same kind of mechanisms to ensure good performances.

Hadoop v2 manages *applications* submitted by *users* to *queues*. Applications are processing workload defined by users: they have to follow the master/slave paradigm. Any time a client submits application to a queue he asks for resource that will be granted for his Master execution. Once the submission is accepted, an Application Master submit resource requests necessary to execute its tasks. It then schedules its tasks on resource acquired before, or asks for resource to the cluster central allocation point. Usually, it uses a kind of generic tool (an execution engine) to deal with scheduling issues, as well as fault tolerance and tasks monitoring. Queues are entities defined by administrators: for each of this, a share of resource is defined, as well as a set of user that may submit applications using this queue.

The cluster-wide allocation of resource is done by a unique component named Resource Manager (RM). Resource Manager handles client requests for application submission as well as resource requests coming from the different AM currently running. RM is responsible for fair resources allocation in regard to queues at which users submits their applications, as defined by the cluster administrator. It allocates resource within Containers. A Container guarantees exclusive access of an amount of resources (nowaday, RAM and vCore) to the application it has been allocated to. For each node, a NodeManager interacts with RM for monitoring and handling Containers allocated by RM to application on the node. An overview of the application submission and execution is given at fig 2. The cluster has n nodes, each of them having 4 cores. Each node then propose 3 vCore, and save the forth core for NodeManager execution. Client request to submit its application to RM; RM then accepts its application, and gives client a Container on node 1 to host its AM. Once the application master (AM) is deployed, it asks for Containers to RM, and execute its tasks within those Containers. Monitoring information from each Container is sent to the AM. Each NM also sends monitoring information to RM (not pictured here).

Overall Hadoop architecture and design relies since its beginning on a few key concepts:

- Homogeneity of resources (resources within the platform are supposed to be distributed evenly on each platform node so as any resource may be replaced seamlessly by another in case of failure)
- Master/slave paradigm is used anywhere possible, since its first version until now. Master/Slave is used for example between RM and NM as well as between AM and Containers.
- heartbeat-based communications that leverages scaling effects. It allows administrator to control monitoring bandwidth consumption by choosing the heartbeat rate.

Since version 2, another design pattern has been widely adopted: event-based communication. The different operation are handled separately by different threads or processes, allowing to decorrelate loads. All those design patterns and architectural choices are driven by the will to scale nicely. It is then normal that scheduling itself relies on simple, fast and efficient online algorithms. The overall behavior of Hadoop's algorithms is quite simple and relies on a *pull* model: any time resources are freed, the scheduler search for resource request to satisfy with these resources (an administrator, depending on the cluster load, may also choose to run continuously this search and ignore this event, which is useful when there is many short running tasks).

3 different schedulers are available in Hadoop v2.6:

- FIFO (First In First Out) is the oldest scheduler implementation, and does not rely on queues.
- Capacity Scheduler handles hierarchy of queues. Administrator defines min and maximum amount of resources that may be used by any application submitted to a given queue. Queues may contain queues that then have to share resource limits defined for the upper level queue. Applications may obtain from the cluster the resource ratio defined for its queue. If multiple applications either from the same queue or from different queues requires resources, they obtain it on first-come-first-serve pattern.
- Fair Scheduler also relies on hierarchy of queues. If cluster resource are underused, an application may obtain

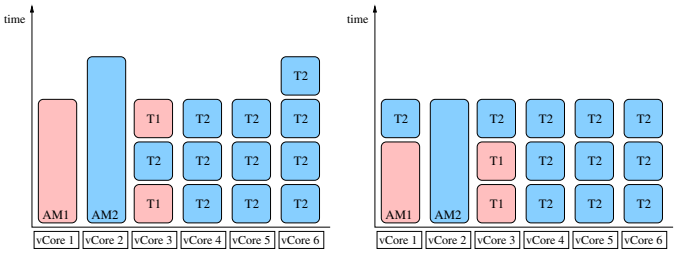

Fig. 3. 2 different schedules for tasks. Left: arbitrary, right: SPT

more than its share of resources. If cluster is heavily loaded, then a newly coming application is ensured to obtain at least its fair share of resource as the RM will use preemption to obtain resource back. In addition to this, a *policy* can be configured for application submitted via the same queue. Possible policies includes FIFO, where no preemption will be used, Fair, where preemption will be used to share resources between applications, based on vCore share, and finally DRF, that calculates share based on the most stressed resource.

III. Integrating SPT into Hadoop

A. Motivation

SPT is optimal for the average flowtime metric. As any application that is executed is driven by an AM that may have resources reserved without actually fully using it (mainly CPU), minimizing flowtime should minimize the number of concurrent AM running, and, by doing so, it should minimize total completion time of a set of applications concurrently submitted. Nowadays schedulers does not take into account the extra load induced by Application Master, and so SPT may be of benefit. This is illustrated by figure 3. In this example, Hadoop cluster has 6 vCores. There are 2 applications running belonging to the same queue: application 1 has to run two sequential tasks of duration 1, while application 2 comes with 10 independant tasks of same unit duration 1. Each application starts at time 0: their Application Masters named AM1 and AM2 starts and consume each one vCore. At first, any application receive vCore according to its need, respectively one and 3 vCore. Then on the left of the figure, application waits till time 3 before gaining a new container. AM1 runs till time 3, while AM2 runs till time 4. On the right side, the scheduler uses SPT; as application 1 is shorter (2 tasks of size 1 instead of 10 for application 2), the scheduler provides application 1 a container for its second task at time 1 instead of 2. Application Master 1 then finishes at time 2, freeing a resource for the last task of application 2 which will end at time 3. Both applications will then finish earlier when using SPT.

B. Integration

As a policy for accepting jobs, SPT may be included in both Capacity and Fair scheduler, or as a Scheduler *per se* for a cluster-wide policy. If Hadoop official documentation states that it is fairly simple to implement new scheduler, the shift to event-based communication done between v1 and v2 of Hadoop architecture does not simplify the task: interfaces to implement in order to plug a new scheduler are in fact empty shells, and so one have to implement event handling to have his scheduler to work. A simpler choice to implement a prototype is to implement SPT as policy for queues: if the queue contains all cluster resources, then its behavior is similar to a cluster-wide scheduler. Capacity Scheduler by default uses FIFO and is not designed to plug new policies but FairScheduler is designed to plug new policies. We then choose to implement SPT as part as policies available for Fair scheduler (with the same default of event-based communication stated before for cluster-wide scheduler).

However integration is not straightforward. Indeed Hadoop is designed based on the principle that any resource may be replaced by another. So, when RM receives a set of resource requests from an application, it fulfills those requests by sending a set of Containers. It is up to application to decide how it will use those Containers, based on its own needs, and taking into account resources already acquired. Indeed in between AM resource requests and RM response, there may be changes in AM application states: some tasks may have finished, some tasks may be ready to be scheduled, etc. There is then in Hadooop implementation no direct relationships between a Container granted to an AM and the resource request that lead RM to grant it. However in our case it is mandatory to do such link in order to schedule tasks of a known duration into a Container of that duration. This led to modify most of scheduler and AM internal as well as the communication protocol between them to support this feature.

IV. Experimental Results

Experiments where run on a machine with 2 processors having each 12 cores, so 24 cores total, 64Gb of RAM, running Linux Fedora 3.15. Our prototype is a modified version of Hadoop 2.5.2. We compared SPT implementation to Fair and FIFO queuing policy within Fair Scheduler. For FIFO and Fair expermiments, we run a genuine 2.5.2 Hadoop version. We configured the system to have 22 cores available as vCore (2 cores where reserved for Client, RM, NodeManager). We configured the maximum number of Application Master running concurrently to 5, other submitted application are staying in SUBMITTED states until a running AM finishes.

For each test, a client submit between 5 and 50 applications to the queue at random start times. Each application sets randomly a duration for any of its tasks, and sets randomly the number of independent tasks to run. Each task is composed of a system sleep (as Container gives exclusive access to resource, there is no need of actual computation). We then choose to set a negligible value for memory requests so as to be able to neglect this parameter. We repeated each test with different random seeds 5 times, and reused those random seeds to have the same experiments running with any of the 3 policies, for a total of 150 tests. Makespan for the tests are given in table I.

TABLE I
TESTS RESULTS (IN MINUTES)

number of app.	Fair	FIFO	SPT
10	5.359543	5.033950	5.195693
15	9.462897	8.994237	9.917407
20	12.507600	11.643107	12.320407
25	15.057217	14.355550	15.571203
30	17.726013	17.233723	16.610773
35	21.098980	20.070550	20.042053
40	22.450530	21.245387	24.466207
45	26.226520	25.071687	25.636037
50	28.938380	27.633343	27.469473
Avg	1059000	1009000	1048000

As we may expect, FIFO gives the best results as it schedules applications in their submission order, while Fair gives worsts results. SPT shows better results in average than Fair policy, and from time to time performs better than FIFO, but on average SPT performances stays closer to Fair ones than FIFO ones.

A closer looks at traces indicates us that SPT policy effect is close to the mechanism existing to handle load induced by AM: queue set up allows to limit the number of Application Master currently running per queue (and by doing so preserve the system for starvation that may occur if all resources are used by Application Master ; in this case there would no more resources to satisfy AM resources requests, and all AM will wait for resources forever). This allows to statically limit the number of currently running AM and so will minimize average flowtime. The effects of SPT may been hidden by this mechanism, mitigating results obtained.

V. RELATED WORK

Many work have been done on improving Hadoop scheduling. One can find many survey, for example [4], [5], [6], [7] or [8]. Algorithms developed range from RM scheduling policy and schedulers to AM schedulers -dedicated or not to MR. It should be noted that many of previous research has been driven by the requirements of former (v1) Hadoop architecture. As Hadoop v1 architecture was integrating all those 3 different kind of algorithms into one single component, and as it was a bottleneck when dealing with scaling, it is often hard to compare those algorithms to algorithms dedicated to YARN that separate cluster-wide and application-wide scheduling and allocation. There are also many attempts to redefine architecture and algorithms at the same time that are irrelevant with our work.

Some works addressed the problem using job duration. Authors of [9] for example use time-based information to do preemption based on jobs duration: as long as a job uses the platform it is *aging* and its priority decline other time ; authors main aim is to provide a more efficient alternative to Hadoop's fair share while preserving a kind of fairness for user, while our work target makespan.

Authors of [10] builds job profiles to predict job duration and then schedule and allocate resources to jobs according to a deadline, using an algorithm similar to the well-known earliest deadline first; it targets only MapReduce tasks as it is based on Hadoop v1.

Haste [11] is a YARN scheduler based on tasks duration and dependencies, but it is dedicated to MapReduce tasks and not any kind of load.

VI. CONCLUSION

In this paper we described integration of Shortest Processing Time in Hadoop and our real-life experiments with our prototype. Despite of what the official documentation says, integrating new feature into Hadoop code is more complicated than implementing an interface, as the event-based implementation pattern used into code eventually led interface to be more or less an empty shell.

Mixed results have been obtained. SPT efficiency is leveraged by the regulation of number of AM concurrently running. This setting has to be done by administrators and wise choice of the ratio between resource used by AM and resources actually used to processing tasks may be complex, and shall depend of the kind of load the cluster have to handle. SPT then may be an help to leverage bad configuration choices on some situations.

ACKNOWLEDGMENT

This work was partially supported by the European FEDER project Big Trend.

REFERENCES

[1] Apache-Foundation, "Apache yarn," 2015, accessed 16-05-2015. [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[2] A. C. Murthy, V. K. Vavilapalli, D. Eadline, J. Niemiec, and J. Markham, *Apache Hadoop YARN: Moving Beyond MapReduce and Batch Processing with Apache Hadoop 2*, 1st ed. Addison-Wesley Professional, 2014.

[3] R. W. Conway, W. L. Maxwell, and L. Miller, *Theory of Scheduling*, 1st ed. Addison-Wesley Publishing Company, 1967, ch. 11.

[4] B. T. Rao and L. S. S. Reddy, "Survey on improved scheduling in hadoop mapreduce in cloud environments," *CoRR*, vol. abs/1207.0780, 2012. [Online]. Available: http://arxiv.org/abs/1207.0780

[5] S. Patil and S. Deshmukh, "Article: Survey on task assignment techniques in hadoop," *International Journal of Computer Applications*, vol. 59, no. 14, pp. 15–18, December 2012, full text available.

[6] A. P. Kulkarni and M. Khandewal, "Survey on hadoop and introduction to yarn," *International Journal of Emerging Technology and Advanced Engineering*, vol. 4, no. 5, pp. 82–87, May 2014.

[7] S. Bardhan and D. A. Menascé, "The anatomy of mapreduce jobs, scheduling, and performance challenges," in *39. International Computer Measurement Group Conference, La Jolla, CA, USA, November 4-8, 2013*, 2013. [Online]. Available: http://www.cmg.org/?s2member_file_download=/proceedings/2013/254-Menasce.pdf

[8] D. H. S. G. Harshitha R, Rekha G S, "A survey on scheduling techniques in hadoop," *International Journal of Engineering Development and Research*, vol. 3, no. ISSN:2321-9939, pp. 248–254, Jan. 2015.

[9] M. Pastorelli, A. Barbuzzi, D. Carra, M. Dell'Amico, and P. Michiardi, "Hfsp: Size-based scheduling for hadoop," in *Big Data, 2013 IEEE International Conference on*, Oct 2013, pp. 51–59.

[10] A. Verma, L. Cherkasova, and R. H. Campbell, "Aria: Automatic resource inference and allocation for mapreduce environments," in *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ser. ICAC '11. New York, NY, USA: ACM, 2011, pp. 235–244. [Online]. Available: http://doi.acm.org/10.1145/1998582.1998637

[11] Y. Yao, J. Wang, B. Sheng, J. Lin, and N. Mi, "Haste: Hadoop yarn scheduling based on task-dependency and resource-demand," in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, June 2014, pp. 184–191.