FEI YANG -Reading Summary of Apache Hadoop YARN: Yet Another Resource Negotiator

Tianyang Liu, Su Pu, Yicheng Wang

Link: https://docs.google.com/document/d/1bJB6HLzDCZyQIvTRpM1oOWjGhEcS36Np9s6T7oiC6bQ/edit

In the first generation of Hadoop, MapReduce had only FIFO schedulers at the very beginning, and thus could not support application scenarios like multi users and multi arrays. Hadoop On Demand (HOD) was introduced to solve this problem by using resource managers Torque, to divide several virtual Hadoop clusters apart from individual physical Hadoop clusters, and to cope with the problem of interference of various types of applications. This novel framework however, has two serious problems: First and foremost, it seperates only the computational resource but not storage resource, which results in really bad data locality. In condensed data applications, data locality must be considered and optimized, indicating the great shortback of HOD. Second, the virtual groups seperated by HOD cannot share each others resource, which means when a group is in idle, we cannot assign the vacant resources to those busy groups through certain mechanisms, making the overall resource utilization rate fairly low. To compensate the shortcomings of HOD, researchers introduced multi-users multi-arrays coordinators, among which the most typical ones are capacity schedulers and fairing schedulers from the Yahoo company. In fact these technologies are still widely being used today by most companies, but intrinsically cannot solve the problems of MapReduce architecture through the view of schedulers. This is because of two reasons: First this cannot support more computational models. Specifically MapReduce solidates the two period models i.e. map and reduce into Hadoop system. As a result it cannot easily support more computational frameworks like DAG and some iterative frameworks. Some people may argue that, many computing frameworks like Giraph has already initialized the BSP model successfully in Map-Only works. We would refute however, that all resources used in Map are exactly the same and could not transform according to specific applications. The second problem here is, that all application related logic as well as resource management logic are put in a single JobTracker, which makes the main servers undertaking too much pressure, and meantime limits the system scalability. Yarn, in this case, was born.

Yarn is the short term from Yet Another Resource Negotiator as we can see from the name of this article. It is believed to adopt the masters slaves structure and overall use double layers coordination framework. Yarn is primarily composed of 4 parts. First the Resource Managers. It is in charge of the main service of resource management and there is only one in the entire system doing resource management, scheduling and overwatching. It supports plugable processing and has FIFO, Fairing Schedulers and Capacity Schedulers. Second the Node Managers. It does single node management, periodically sending resource request to Resource Managers and receive commands from Application Masters to boot and recollect Containers. Third the Application Masters which controls a single application. It sends requests to Resource Managers to obtain resources using which to boot internal tasks. Meantime it monitors running tasks and handling exceptions. Fourth the Containers. It is an abstract of resource through packaging the CPU resource and memory resources of a single node. Application Masters could not work before getting Containers. Besides, Application Masters are running inside Containers. The first demo of Hadoop was concentrated on running massive MapReduce tasks to process a crawl on web. And to various companies, Hadoop has become the data and computationally shared and accessed. This wide adoption and unique utilization has widen the initial design deviated from its initial target, but exposed two main issues. First the tight coupling of a programming model with resource management base which forces users to waste MapReduce models. And second, the contralized using of control flow which ends up to be endless scalability concerns. This work summarized the design and development of YARN which decouples the programming model and delegates many scheduling functions. It work also discussed the current situation of YARN using by Yahoo. Yahoo has switched all its clusters approximately 7000 into YARN, and runs almost 500k works everyday. The article compared throughput and resource utilization rates between 1.0 and 2.0 versioon upon 2500 nodes. Compared to Hadoop 1.0 YARN increases system throughput from 77k to 100k and double the utilization rates in experiments.

Three strong points of this article are as follows:
1. Before YARN, the Hadoop need a strict dependency on MapReduce, since all the MapReduce jobs have to run as a batch process through a single daemon, which would greatly influence the scalability and speed. This problems seems to be a bottleneck when dealing with extremely large data. but the distrubuted application in YARN will only pick a contrainer to execute.
2. YARN uses ResourceManager in place of cluster manager, and the Rescource Manager is a global manager. Different structures have different advantages, for example, MapReduce is good at dealing with large ETL task while Spark is better at Machine Learning. When a cluster in a company has 100 machine. Before YARN, they

need to distribute the machines into two parts, like a half for Spark and the rest for MapReduce. The problems are that, the machines for MapReduce only run for 1 hour per day to finish the task while the machines for Spark need to run for the whole day but there are still lots of tasks waiting. So this kind of resource distribution method is not efficient [2]. However, as for YARN it knows which machine is unoccupied, and when the users submit task to the cluster, YARN only need to distribute required resource to the task.
3. In the paper, the author showed the great improvement compared with Hadoop 1.0: in terms of numbers of jobs runing on 2500 machine cluster, Hadoop 1.0 could only hold 77k, while on YARN it can hold 100k jobs. And the CPU utilization almost doubled when change to YARN.

Three weak points of this article are as follows:
1. The resource manager has the single point faliure pleblem. Wehn one single problem coming in, the RM will recover from the failure by reading the states from persisten storage, it will break whole system consistency, and all the runing containers and application will be killed and restarted.
2. The node manager offers local services to the containers running on that node, the log aggregation service will upload data written by the application to stdout and stderr to HDFS once the application completes. That will create workload to the HDFS and cause scalbility issue.
3. Compare to other systems,YARN only handle memeory resources. YARN only use simple Unix process which have weaker isolation than Linux.

Some new research ideas following this paper would be:
1. In the future study, just as we mentioned about the problem in the previous part, the YARN should work with decreasing the pressure on the HDFS and optimise the log aggregation problem.
Work is going on to allow AM to continue to work even if RM fails and then resync when RM recovers.
2. To optimize the RM recover systm, optimize the strategy restart all the applications in the RM when fail happens. The better way is to keep AM working on when RM fails and then resync the RM after it recovers
3. YARN can also improve its packaging mechanism, it can create some lightweight mechanism on packaging and resource isolation.

References:
[1] Jennings B, Stadler R. Resource management in clouds: Survey and research challenges[J]. *Journal of Network and Systems Management*, 2015, 23(3): 567-619.
[2] Thain, Douglas, Todd Tannenbaum, and Miron Livny. "Distributed computing in practice: the Condor experience." *Concurrency and computation: practice and experience* 17.2-4 (2005): 323-356.
[3] Capit, Nicolas, et al. "A batch scheduler with high level components." CCGrid 2005. *IEEE International Symposium on Cluster Computing and the Grid*, 2005.. Vol. 2. IEEE, 2005.
[4] https://newpush.com/2014/06/yarn-review/
[5] http://dongxicheng.org/mapreduce-nextgen/apache-hadoop-yarn-paper-on-socc2013/