Extending Hadoop's Yarn Scheduler Load Simulator with a Highly Realistic Network & Traffic Model

Philip Wette, Arne Schwabe, Malte Splietker, Holger Karl
University of Paderborn
33098 Paderborn, Germany
Email: {firstname.lastname}@uni-paderborn.de

***Abstract*—Research on accelerating big-data applications can be divided into *job scheduling* and *flow scheduling*. Job scheduling focuses on the timely and spacial placement of jobs on execution units. Flow scheduling, on the other hand, concentrates on routing of flows originating from actively running jobs. Although both job scheduling and flow scheduling work on accelerating big-data applications, their view on the problem and the available information is very different.**

We propose a new simulation tool to evaluate ideas that jointly solve the job and flow scheduling problem for big-data applications. Our tool combines the Yarn Scheduler Load Simulator with the distributed network emulator MaxiNet. With our work, the interdependency between the network and the jobs running on top of it can be included into the evaluation of new ideas, leveraging research on big-data applications with *joint* job and flow scheduling.

I. Introduction

Recently, research on scheduling flows from big-data frameworks has gained a lot of attraction. Flow schedulers like Varys [1] and Barrat [2] are built to increase the performance of big-data applications by optimizing the network for the running parts of the application. To this end, they use information about the mapping of flows (on the network level) to jobs (on the application level). This information is then used to route flows such that job-related metrics (like the job completion time or the makespan) are optimized.

On the other hand, some work focuses on optimizing big-data *job schedulers*. Each job has certain requirements on computing power, disc space, and communication. Based on these requirements, the job schedulers decide which part of the application is to be executed at which machine at what time. Tetris [3] is an example of a modern job scheduler that optimizes average job completion time while keeping a certain level of fairness between jobs.

So far, only little work has been done on combining job scheduling and flow scheduling by creating a job scheduler that is network aware or by creating a feedback loop between flow and job scheduler. One reason for the absence of such an integration is that it is very complicated to evaluate. Modeling the exact behavior of the interaction between distributed applications and an interconnecting network is a complex task. To create trustable results, a testbed of reasonable size is required, leading to very high costs. To overcome the hurdle, the Yarn Scheduler Load Simulator[1] (SLS) has been built. SLS is a tool for simulating Hadoop clusters to predict the behavior of new job scheduling algorithms. SLS, however, does not model the interconnecting network, which makes it the wrong tool for evaluating network-related scheduling features.

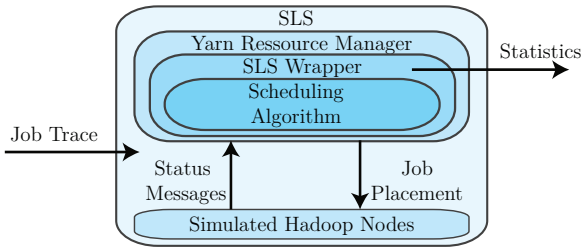

Fig. 1. Schematic view of SLS.

In this work, we extend SLS by a highly realistic network model making it possible to conduct research in combined job and flow scheduling. SLS is used to simulate new scheduling algorithms on real world traces of big-data jobs. Whenever the scheduler decides to start a new job on a set of machines, the network emulator emulates the data transmissions corresponding to the job.

II. Yarn Scheduler Load Simulator

SLS is built to mimic a running Hadoop cluster to the Resource Manager (the entity running the scheduling algorithm). This allows the execution of unmodified Hadoop-scheduling algorithms in the simulated environment.

To benchmark a scheduling algorithm using the Resource Manager, a job trace is given as input to SLS. A job trace is a record of various parameters and statistics of jobs from a real Hadoop cluster. Among other data, a trace contains statistics of every run job and how much data was generated (and transferred) by the job. Job traces can be replayed inside SLS.

Fig. 1 shows a high-level view of SLS. SLS simulates Hadoop by running an unmodified instance of the Resource Manager. The Resource Manager controls a set of simulated Hadoop nodes by assigning jobs and monitoring the cluster. SLS inserts

[1] http://hadoop.apache.org/docs/r2.3.0/hadoop-sls/SchedulerLoadSimulator.html

978-1-4799-7899-1/15/$31.00 ©2015 IEEE

a shim layer between the Resource Manager and its scheduling algorithm. This layer is solely used to collect statistics about the jobs and the scheduler itself. In contrast to a real Hadoop cluster, in SLS there are no nodes actually computing map or reduce tasks. On arrival of a task assignment, nodes wait a certain time (as specified in the job trace) and report back the successful execution of the task.

III. Adding a Realistic Data-Center Model

We extend SLS to *NetSLS* by adding actual data transmissions to the simulated tasks. In Hadoop, processing a task consists of the three sequential phases fetching data, processing data, and storing data. Both fetch and store requires to access data. Depending on the location of the data, this involves the network. To model the network interconnecting the Hadoop nodes, NetSLS uses MaxiNet[4], a highly realistic, distributed network emulator for Software-Defined Networks (SDN). MaxiNet distributes a virtual network onto several physical machines (called workers) making it possible to emulate very large (data-center) networks on only a few physical resources. To distribute a virtual network G_V that is to be emulated on n workers, MaxiNet first partitions G_V into n parts. Then, each partition is emulated at one of the workers. All traffic that is between different partitions is tunneled through the physical network interconnecting the workers into the destination partition.

Fig. 2 shows how NetSLS works. On startup, NetSLS fetches the cluster topology from a file supplemental to the job trace and sets up MaxiNet to emulate the network. For each simulated Hadoop node N_i, NetSLS emulates a corresponding node, $\tilde{N}_i$ using MaxiNet. Whenever the scheduler decides to execute a task on N_i, NetSLS looks up information about data transfers from the job trace and initiates the corresponding transfers at $\tilde{N}_i$. The completion of the transfer is subsequently reported from $\tilde{N}_i$ to N_i which, in turn, reports a successful completion to the job scheduler when the three phases of the job have finished.

As MaxiNet emulates an OpenFlow-capable SDN, an OpenFlow controller manages the network. The controller is external to NetSLS, thus any OpenFlow controller can be used to control the behavior of the network. The flow scheduler resides in the OpenFlow controller. Being a part of the controller, it has access to any information present at the controller. To exchange information with the job scheduler, a custom interface between the job and flow scheduler can be established.

For evaluating novel ideas under different network loads we also integrated DCT^{2}Gen[5], a traffic generator for highly realistic data-center traffic. In NetSLS, DCT^{2}Gen creates background traffic mimicking cross traffic from other applications.

IV. Conclusion

By turning SLS into NetSLS, novel job and flow scheduling algorithms can be tested using little time and effort. Along a job trace and a topology description, the only two inputs to NetSLS are a job scheduling algorithm and an OpenFlow controller implementing the desired flow scheduling algorithm.

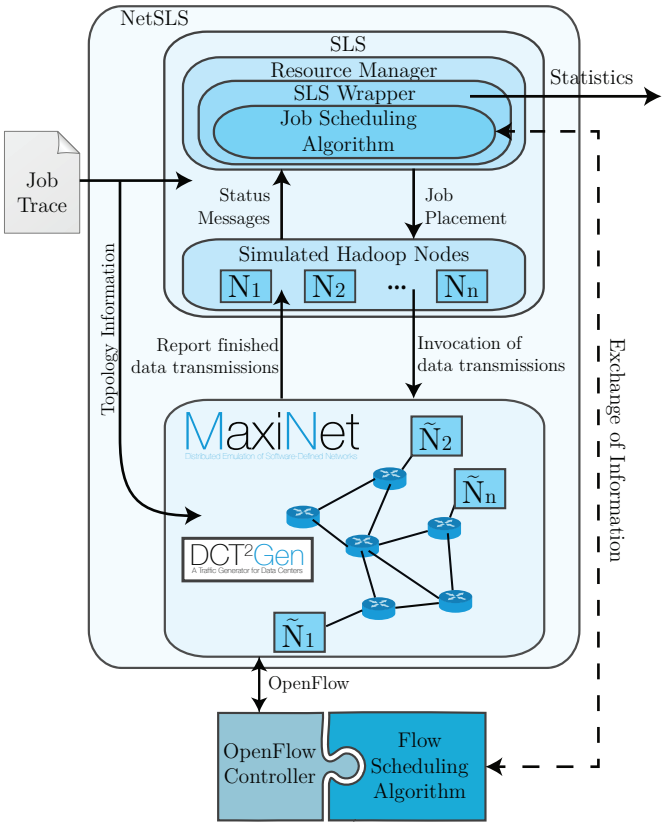

Fig. 2. Schematic view on NetSLS being controlled by an external OpenFlow controller.

NetSLS will then simulate the interplay between the flow and job scheduler on the job trace, leading to fast and realistic evaluation of novel scheduling algorithms without requiring an expensive testbed or a custom simulation.

Acknowledgment

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre “On-The-Fly Computing” (SFB 901). The work in this paper has been partially sponsored by the European Union via the FP7 project NetIDE, grant agreement 619543.

References

[1] M. Chowdhury, Y. Zhong, and I. Stoica, “Efficient coflow scheduling with varys,” in *Proceedings of the Conference on SIGCOMM*. ACM, 2014.

[2] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, “Decentralized task-aware scheduling for data center networks,” in *Proceedings of the Conference on SIGCOMM*. ACM, 2014.

[3] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, “Multi-resource packing for cluster schedulers,” in *Proceedings of the Conference on SIGCOMM*. ACM, 2014.

[4] P. Wette, M. Dräxler, A. Schwabe, F. Wallaschek, M. H. Zahraee, and H. Karl, “MaxiNet: Distributed Emulation of Software-Defined Networks,” in *Proceedings of the IFIP Networking Conference*, 2014.

[5] P. Wette and H. Karl, “DCT2Gen: A Versatile TCP Traffic Generator for Data Centers,” arXiv:1409.2246.