Yarn Scheduler Load Simulator

Project Documentation

Contents

Overview

Overview

The Yarn scheduler is a fertile area of interest with different implementations, e.g., Fifo, Capacity and Fair schedulers. Meanwhile, several optimizations are also made to improve scheduler performance for different scenarios and workload. Each scheduler algorithm has its own set of features, and drives scheduling decisions by many factors, such as fairness, capacity guarantee, resource availability, etc. It is very important to evaluate a scheduler algorithm very well before we deploy in a production cluster. Unfortunately, currently it is non-trivial to evaluate a scheduler algorithm. Evaluating in a real cluster is always time and cost consuming, and it is also very hard to find a large-enough cluster. Hence, a simulator which can predict how well a scheduler algorithm for some specific workload would be quite useful.

The Yarn Scheduler Load Simulator (SLS) is such a tool, which can simulate large-scale Yarn clusters and application loads in a single machine. This simulator would be invaluable in furthering Yarn by providing a tool for researchers and developers to prototype new scheduler features and predict their behavior and performance with reasonable amount of confidence, thereby aiding rapid innovation.

The simulator will exercise the real Yarn ResourceManager removing the network factor by simulating NodeManagers and ApplicationMasters via handling and dispatching NM/AMs heartbeat events from within the same JVM. To keep tracking of scheduler behavior and performance, a scheduler wrapper will wrap the real scheduler.

The size of the cluster and the application load can be loaded from configuration files, which are generated from job history files directly by adopting Apache Rumen.

The simulator will produce real time metrics while executing, including:

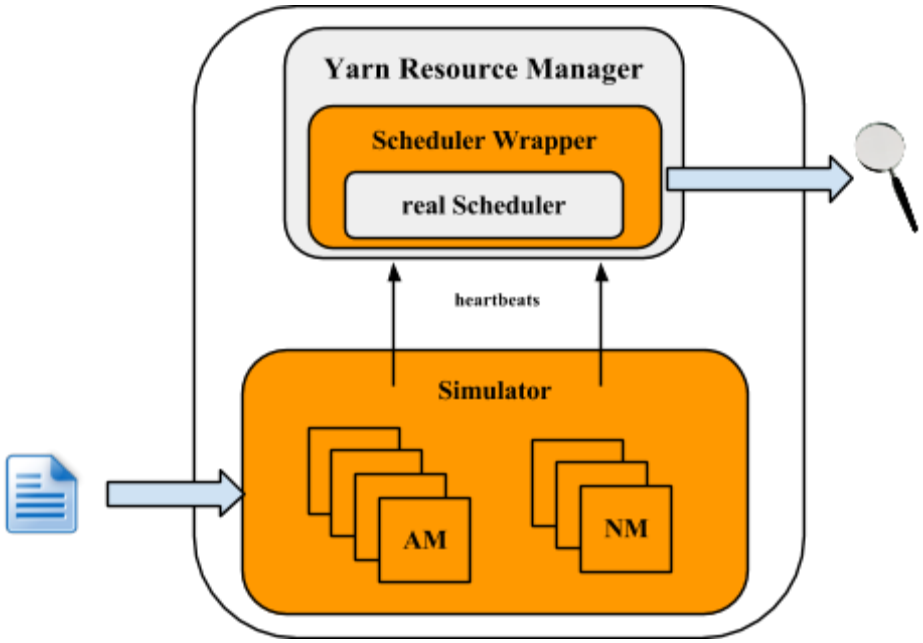
- Resource usages for whole cluster and each queue, which can be utilized to configure cluster and queue's capacity.
- The detailed application execution trace (recorded in relation to simulated time), which can be analyzed to understand and validate the scheduler behavior (individual job's turn around time, throughput, fairness, capacity guarantee, etc.).
- Several key metrics of scheduler algorithm, such as time cost of each scheduler operation (allocate, handle, etc.), which can be utilized by Hadoop developers to find the code spots and scalability limits.

Goal

- Exercise the scheduler at scale without a real cluster using real job traces.
- Being able to simulate real workload.

Architecture

The following figure illustrates the implementation architecture of the simulator.



The simulator takes input of workload traces, and fetches the cluster and applications information. For each NM and AM, the simulator builds a simulator to simulate their running. All NM/AM simulators run in a thread pool. The simulator reuses Yarn Resource Manager, and builds a wrapper out of the scheduler. The scheduler wrapper can track the scheduler behavior and produce several logs, which are the outputs of the simulator and can be further analyzed.

Use cases

- Engineering
  - Verify correctness of scheduler algorithm under load.
  - Cheap/practical way for finding code hotspots/critical-path.
  - Validate the impact of changes and new features.
  - Determine what drives the scheduler scalability limits.
- QA
  - Validate scheduler behavior for "large" clusters and several workload profiles.
- Solution/Sales
  - Sizing model for predefined/typical workloads.
  - Cluster sizing tool using real customer data (job traces).
  - Determine minimum SLAs under a particular workload.

Usage

This section will show how to use the simulator. Here let `$HADOOP_ROOT` represent the Hadoop install directory. If you build Hadoop yourself, `$HADOOP_ROOT` is `hadoop-dist/target/hadoop-$VERSION`. The simulator is located at `$HADOOP_ROOT/share/hadoop/tools/sls`. The folder `sls` contains four directories: `bin`, `html`, `sample-conf`, and `sample-data`.

- `bin`: contains running scripts for simulator.
- `html`: contains several html/css/js files we needed for real-time tracking.

- `sample-conf`: specifies the simulator configurations.
- `sample-data`: provides an example rumen trace, which can be used to generate inputs of the simulator.

The following sections will describe how to use the simulator step by step. Before start, make sure that command `hadoop` is included in your `$PATH` environment parameter.

Step 1: Configure Hadoop and the simulator

Before we start, make sure Hadoop and the simulator are configured well. All configuration files for Hadoop and the simulator should be placed in directory `$HADOOP_ROOT/etc/hadoop`, where the ResourceManager and Yarn scheduler load their configurations. Directory `$HADOOP_ROOT/share/hadoop/tools/sls/sample-conf/` provides several example configurations, that can be used to start a demo.

For configuration of Hadoop and Yarn scheduler, users can refer to Yarn's website (http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/).

For the simulator, it loads configuration information from file `$HADOOP_ROOT/etc/hadoop/sls-runner.xml`. Here we illustrate each configuration parameter in `sls-runner.xml`. Note that `$HADOOP_ROOT/share/hadoop/tools/sls/sample-conf/sls-runner.xml` contains all the default values for these configuration parameters.

- `yarn.sls.runner.pool.size`
  The simulator uses a thread pool to simulate the NM and AM running, and this parameter specifies the number of threads in the pool.
- `yarn.sls.nm.memory.mb`
  The total memory for each NMSimulator.
- `yarn.sls.nm.vcores`
  The total vCores for each NMSimulator.
- `yarn.sls.nm.heartbeat.interval.ms`
  The heartbeat interval for each NMSimulator.
- `yarn.sls.am.heartbeat.interval.ms`
  The heartbeat interval for each AMSimulator.
- `yarn.sls.am.type.mapreduce`
  The AMSimulator implementation for MapReduce-like applications. Users can specify implementations for other type of applications.
- `yarn.sls.container.memory.mb`
  The memory required for each container simulator.
- `yarn.sls.container.vcores`
  The vCores required for each container simulator.
- `yarn.sls.runner.metrics.switch`

The simulator introduces Metrics (http://metrics.codahale.com/) to measure the behaviors of critical components and operations. This field specifies whether we open (`ON`) or close (`OFF`) the Metrics running.

- `yarn.sls.metrics.web.address.port`
  The port used by simulator to provide real-time tracking. The default value is 10001.
- `org.apache.hadoop.yarn.server.resourcemanager.scheduler.fifo.FifoScheduler`
  The implementation of scheduler metrics of Fifo Scheduler.
- `org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler`
  The implementation of scheduler metrics of Fair Scheduler.
- `org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler`
  The implementation of scheduler metrics of Capacity Scheduler.

Step 2: Run the simulator

The simulator supports two types of input files: the rumen traces and its own input traces. The script to start the simulator is `slsrun.sh`.

```
$ $HADOOP_ROOT/share/hadoop/tools/sls/bin/slsrun.sh
  --input-rumen|--input-sls=<TRACE_FILE1,TRACE_FILE2,...>
  --output-dir=<SLS_SIMULATION_OUTPUT_DIRECTORY>
  [--nodes=<SLS_NODES_FILE>]
  [--track-jobs=<JOBID1,JOBID2,...>]
  [--print-simulation]
```

- `--input-rumen`: The input rumen trace files. Users can input multiple files, separated by comma. One example trace is provided in `$HADOOP_ROOT/share/hadoop/tools/sls/sample-data/2jobs2min-rumen-jh.json`.
- `--input-sls`: Simulator its own file format. The simulator also provides a tool to convert rumen traces to sls traces (`rumen2sls.sh`). Refer to appendix for an example of sls input json file.
- `--output-dir`: The output directory for generated running logs and metrics.
- `--nodes`: The cluster topology. By default, the simulator will use the topology fetched from the input json files. Users can specifies a new topology by setting this parameter. Refer to the appendix for the topology file format.
- `--track-jobs`: The particular jobs that will be tracked during simulator running, separated by comma.
- `--print-simulation`: Whether to print out simulation information before simulator running, including number of nodes, applications, tasks, and information for each application.

In comparison to rumen format, here the sls format is much simpler and users can easily generate various workload. The simulator also provides a tool to convert rumen traces to sls traces.

```
$ $HADOOP_ROOT/share/hadoop/tools/sls/bin/rumen2sls.sh
  --rumen-file=<RUMEN_FILE>
  --output-dir=<SLS_OUTPUT_DIR>
  [--output-prefix=<SLS_FILE_PREFIX>]
```

- `--rumen-file`: The rumen format file. One example trace is provided in directory `sample-data`.
- `--output-dir`: The output directory of generated simulation traces. Two files will be generated in this output directory, including one trace file including all job and task information, and another file showing the topology information.
- `--output-prefix`: The prefix of the generated files. The default value is "*sls*", and the two generated files are `sls-jobs.json` and `sls-nodes.json`.

Metrics

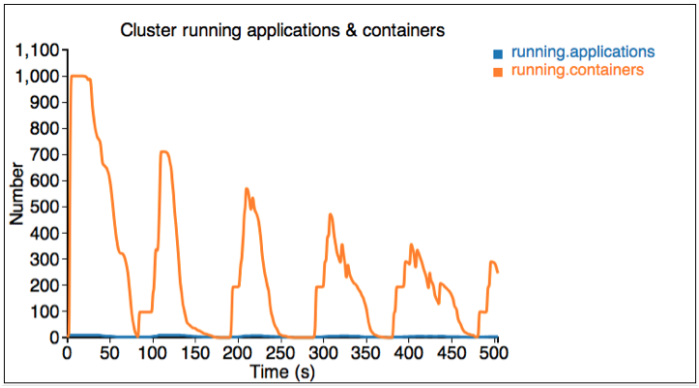
The Yarn Scheduler Load Simulator has integrated Metrics to measure the behaviors of critical components and operations, including running applications and containers, cluster available resources, scheduler operation time cost, et al. If the switch `yarn.sls.runner.metrics.switch` is set `ON`, Metrics will run and output it logs in `--output-dir` directory specified by users. Users can track this information during simulator running, and can also analyze these logs after running to evaluate the scheduler performance.

Real-time Tracking

The simulator provides an interface for tracking its running in real-time. Users can go to `http://host:port/simulate` to track whole running, and `http://host:port/track` to track a particular job or queue. Here the `host` is the place when we run the simulator, and `port` is the value configured by `yarn.sls.metrics.web.address.port` (default value is 10001). Here we'll illustrate each chart shown in the web page.

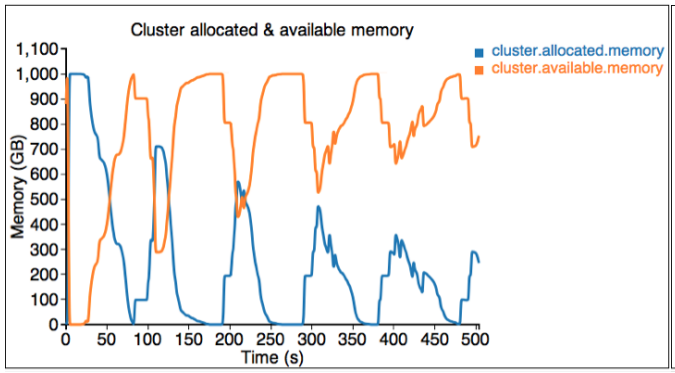
Number of running applications and containers

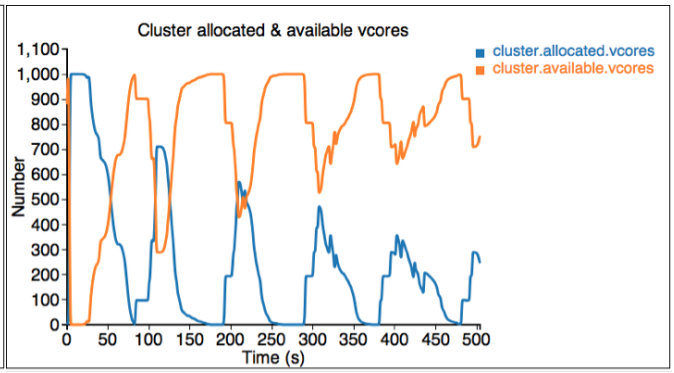
The first figure describes the number of running applications and containers.



Cluster allocated and available resources

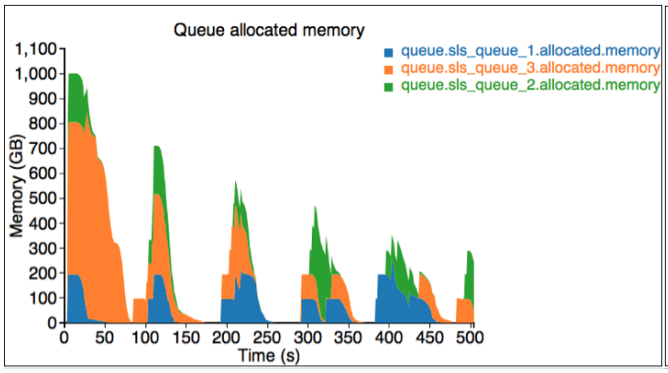
The following figures describe the allocated and available resources (memory, vCores) in the cluster.

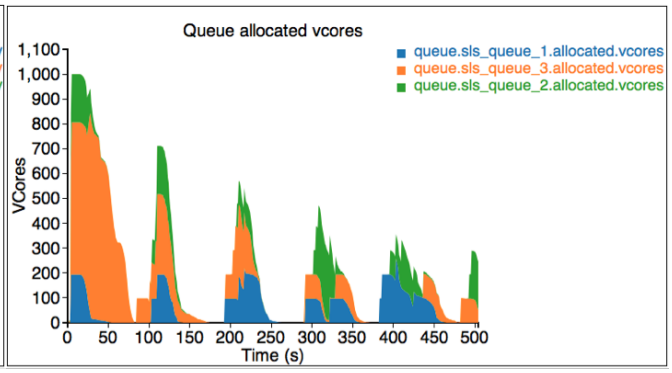




Queue allocated resources

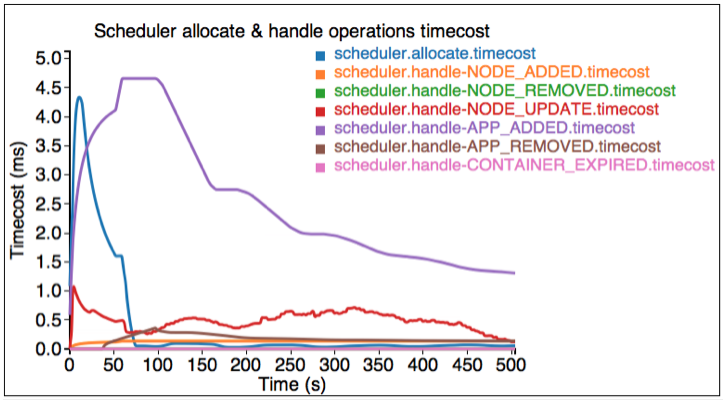
The following figures illustrate the allocated resource for each queue. Here we use Fair scheduler with three queues: `sls_queue_1`, `sls_queue_2`, and `sls_queue_3`.The first two queues are configured with 25% weight, while the last one has 50% weight.





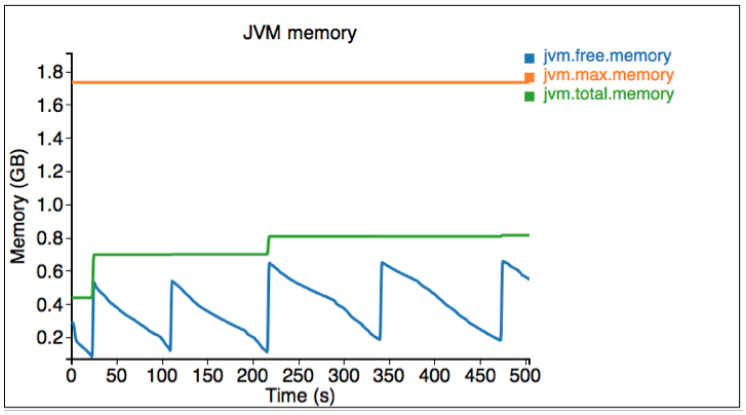
Scheduler operation timecost

The following figure describes the timecost for each scheduler operation.



Simulator memory usage

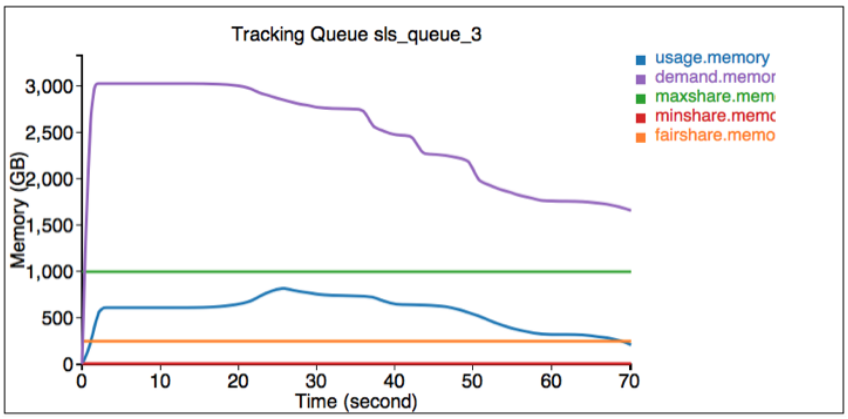
Finally, we measure the memory used by the simulator.



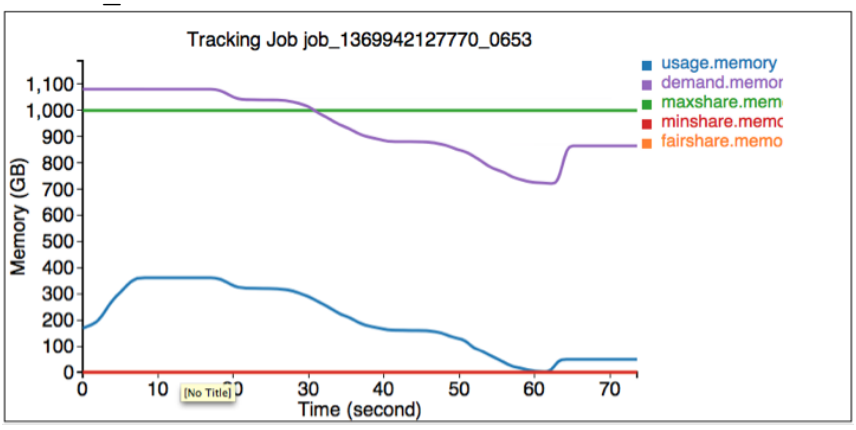
Tracking particular queues and jobs

The simulator also provides an interface for tracking some particular jobs and queues. Go to `http://Host:Port/track` to get these information.

Here the first figure illustrates the resource usage information for queue `sls_queue_3`.



The second figure illustrates the resource usage information for job `job_1369942127770_0653`.



Offline Analysis

After the simulator finishes, all logs are saved in the output directory specified by `--output-dir` in `$HADOOP_ROOT/share/hadoop/tools/sls/bin/slsrun.sh`.

- File `realtimetrack.json`: records all real-time tracking logs every 1 second.
- File `jobruntime.csv`: records all jobs' start and end time in the simulator.
- Folder `metrics`: logs generated by the Metrics.
-

Users can also reproduce those real-time tracking charts in offline mode. Just upload the `realtimetrack.json` to `$HADOOP_ROOT/share/hadoop/tools/sls/html/showSimulationTrace.html`. For browser security problem, need to put files `realtimetrack.json` and `showSimulationTrace.html` in the same directory.

Appendix

Resources

YARN-1021 is the main JIRA that introduces Yarn Scheduler Load Simulator to Hadoop Yarn project.

SLS JSON input file format

Here we provide an example format of the sls json file, which contains 2 jobs. The first job has 3 map tasks and the second one has 2 map tasks.

```
{
  "am.type" : "mapreduce",
  "job.start.ms" : 0,
  "job.end.ms" : 95375,
  "job.queue.name" : "sls_queue_1",
  "job.id" : "job_1",
  "job.user" : "default",
  "job.tasks" : [ {
    "container.host" : "/default-rack/node1",
    "container.start.ms" : 6664,
    "container.end.ms" : 23707,
    "container.priority" : 20,
    "container.type" : "map"
  }, {
    "container.host" : "/default-rack/node3",
    "container.start.ms" : 6665,
    "container.end.ms" : 21593,
    "container.priority" : 20,
    "container.type" : "map"
  }, {
    "container.host" : "/default-rack/node2",
```

```
    "container.start.ms" : 68770,
    "container.end.ms" : 86613,
    "container.priority" : 20,
    "container.type" : "map"
  } ]
}
{
  "am.type" : "mapreduce",
  "job.start.ms" : 105204,
  "job.end.ms" : 197256,
  "job.queue.name" : "sls_queue_2",
  "job.id" : "job_2",
  "job.user" : "default",
  "job.tasks" : [ {
    "container.host" : "/default-rack/node1",
    "container.start.ms" : 111822,
    "container.end.ms" : 133985,
    "container.priority" : 20,
    "container.type" : "map"
  }, {
    "container.host" : "/default-rack/node2",
    "container.start.ms" : 111788,
    "container.end.ms" : 131377,
    "container.priority" : 20,
    "container.type" : "map"
  } ]
}
```

Simulator input topology file format

Here is an example input topology file which has 3 nodes organized in 1 rack.

```
{
  "rack" : "default-rack",
  "nodes" : [ {
    "node" : "node1"
  }, {
    "node" : "node2"
  }, {
    "node" : "node3"
  } ]
}
```