Synthetic Generator for SLS

Summary

This document briefly summarizes the intuition behind the synthetic load generator we ae adding to the SLS simulator in YARN-6363.

SLS is a useful tool to test changes in the ResourceManager, and explore functional and performance implications of different parametrizations. The current SLS supports rumen traces, as well as sls input traces, which extensively describe a set of jobs. What is missing is a compact way to define a workload, by only characterizing its statistical properties (e.g., average job size, number of jobs, etc..). The synthetic generator we introduce in YARN-6363 fill this gap.

The load generator is organized as a JobStoryProducer (compatible with rumen, and thus gridmix for later integration). We seed the Random number generator so that results are reproducible.

We organize the jobs being generated around */workloads/job_class* hierarchy, which allow to easily group jobs with similar behaviors and categorize them (e.g., jobs with long running containers, or map-only computations, etc..). The user can control average and standard deviations for many of the important parameters, such as number of mappers/reducers, duration of mapper/reducers, size (mem/cpu) of containers, chance of reservation, etc. We use weighted-random sampling (whenever we pick among a small number of options) or LogNormal distributions (to avoid negative values) when we pick from wide ranges of values---see appendix on LogNormal distributions.

Command line invocation

The SLS can be configured to run using the synthetic generator with the following arguments (tested):
`slsrun.sh --tracetype=SYNTH --tracelocation=<syn.json> --output-dir=<outdir>`

Configuring the synthetic generator

Below we provide an example of syn.json file and add on a side (non-json valid) comments for what each configurable means.

```
{
  "description" : "tiny jobs workload",    //description of the meaning of this collection of
workloads, not used programmatically, just here for documentation.
  "num_nodes" : 10,  //total nodes in the simulated cluster
  "nodes_per_rack" : 4, //number of nodes in each simulated rack
  "num_jobs" : 10, // total number of jobs being simulated
  "rand_seed" : 2, //the random seed used for deterministic randomized runs

// a list of "workloads", each of which has job classes, and temporal properties
"workloads" : [
    {
      "workload_name" : "tiny-test", // name of the workload
      "workload_weight": 0.5,  // used for weighted random selection of which workload to sample
from
      "queue_name" : "sls_queue_1", //queue the job will be submitted to

//different classes of jobs for this workload
       "job_classes" : [
        {
          "class_name" : "class_1", //name of the class
          "class_weight" : 1.0, //used for weighted random selection of class within workload

        //nextr group controls average and standard deviation of a LogNormal distribution that
        //determines the number of mappers and reducers for thejob.
          "mtasks_avg" : 5,
          "mtasks_stddev" : 1,
          "rtasks_avg" : 5,
          "rtasks_stddev" : 1,

             //averge and stdev input param of LogNormal distribution controlling job duration
          "dur_avg" : 60,
          "dur_stddev" : 5,
```

```
            //averge and stdev input param of LogNormal distribution controlling mappers and
reducers durations

          "mtime_avg" : 10,

          "mtime_stddev" : 2,

          "rtime_avg" : 20,

          "rtime_stddev" : 4,


            //averge and stdev input param of LogNormal distribution controlling memory and
cores for mappers and reducers


          "map_max_memory_avg" : 1024,

          "map_max_memory_stddev" : 0.001,

          "reduce_max_memory_avg" : 2048,

          "reduce_max_memory_stddev" : 0.001,

          "map_max_vcores_avg" : 1,

          "map_max_vcores_stddev" : 0.001,

          "reduce_max_vcores_avg" : 2,

          "reduce_max_vcores_stddev" : 0.001,


       //probability of running this job with a reservation

          "chance_of_reservation" : 0.5,

       //input parameters of LogNormal distribution that determines the deadline slack (as a
multiplier of job duration)

          "deadline_factor_avg" : 10.0,

          "deadline_factor_stddev" : 0.001,

        }

       ],

// for each workload determines with what probability each time bucket is picked to choose the
job starttime. In the example below the jobs have twice as much chance to start in the first
minute than in the second minute of simulation, and then zero chance thereafter.

      "time_distribution" : [

        { "time" : 1, "weight" : 66 },

        { "time" : 60, "weight" : 33 },

        { "time" : 120, "jobs" : 0 }

     ]

        }

 ]

}
```

Notes on LogNormal distribution:

LogNormal distributions represent well many of the parameters we see in practice (e.g., most jobs have a small number of mappers, but few might be very large, and few very small, but greater than zero. It is however worth noticing that it might be tricky to use, as the average is typically on the right side of the peak (most common value) of the distribution, because the distribution has a one-side tail (see drawing in figure).