ds-sim: an open-source and language-independent distributed systems simulator

User Guide

School of Computing
Macquarie University, Sydney, Australia

Core Developer:	Young Choon Lee
Contributors:	Young Ki Kim and Jayden King
ds-sim Version:	14-Jan, 2024
Last modified:	31 Mar 2024 by Young Choon Lee
Changes:	the detailed description of -v added

1 Overview

`ds-sim` is an open-source, language-independent and configurable distributed systems simulator. It is designed to perform a quick and realistic simulation of job scheduling and execution in distributed systems, such as computer clusters and (cloud) data centres (Figure 1). In addition, simulation results are reproducible. The implementation of ds-sim is based on discrete-event simulation [1]. ds-sim consists of two main components: `ds-client` and `ds-server` (i.e., the client-server model). The client-side simulator (ds-client) acts as a job scheduler (shown in the green box in Figure 1) while the server-side simulator (ds-server) simulates everything else including users (job submissions) and servers (job execution), shown in the red boxes in Figure 1. In particular, ds-client (or any name of your choice) is a pluggable component that can be implemented with different scheduling policies/algorithms as long as it follows ds-sim's communication protocol.

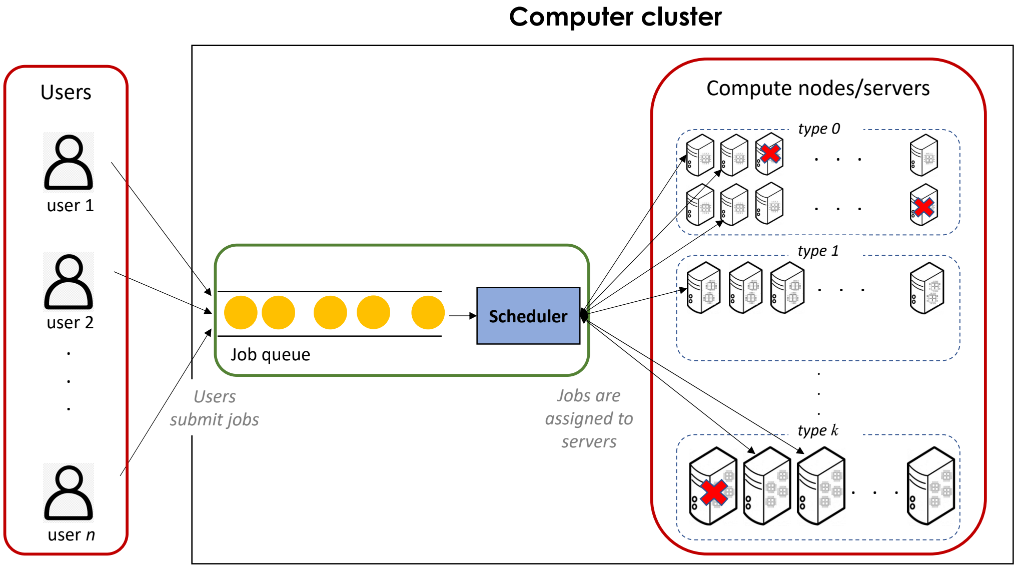

Figure 1: Job scheduling. Note ds-sim simulates a queue for each server in addition to the global job queue shown in the green box. In other words, a job is placed in a queue if the server scheduled to is unable to run it (busy running one or more jobs).

Broadly, ds-sim performs a simulation repeating three key steps:

- ds-server generates a job and submits it to ds-client,
- ds-client makes a scheduling decision and sends it to ds-server, and
- ds-server runs the job based on the scheduling decision.

More specifically,

Initialise simulation:

- ds-server configures a simulation including servers and jobs by reading a configuration file (e.g., `ds-sample-config01.xml`)
- ds-server writes a system information file (`ds-system.xml`) containing only system/server information extracted from the configuration file
- ds-client optionally reads the system information file[1]

Initialise system clock (starts at simulation time zero)

While there is a simulation event to handle **do**

- Get a simulation event (job submission, job completion, server failure, etc)
- Set clock to next event time (based on the latest simulation event)
- Handle the event, such as scheduling a job
- Update system states
- Record statistics

End while

Generate simulation report

2 Background

Distributed (computing) systems come in various sizes and scale (Figure 2). They range from a single workstation computer with several processors, a cluster of compute nodes (aka servers) to a federation of geographically distributed data centres with millions of servers, e.g., Google data centres, Amazon Web Services (AWS) clouds and etc. The main component of distributed systems is servers. These servers are networked together and often form a single system, e.g., a compute cluster or data centre. Roughly, when these servers are virtualised, the system is called a 'cloud' data centre or simply a cloud. Note that servers of a particular distributed system cannot be assumed to be identical/homogeneous; they are often heterogeneous in terms of processor architecture and resource capacity.

(a) a server w/ multiple CPUs (b) a server rack (c) a data centre (d) a federated cloud

Figure 2: Distributed systems.

Whether a distributed system is privately owned or publicly available, it in general deals with a diverse set of jobs of multiple users. In other words, servers in a distributed system are shared among many jobs, such as a user application like our simulator, a MapReduce data-processing job and a long-running web service. These jobs have different resource requirements in terms of the number of CPU cores, memory and disk space.

[1] Alternatively, ds-client can find out system information, such as the number of server types, the number of servers in each type and server resource capacity (e.g., #cores, RAM and disk storage)

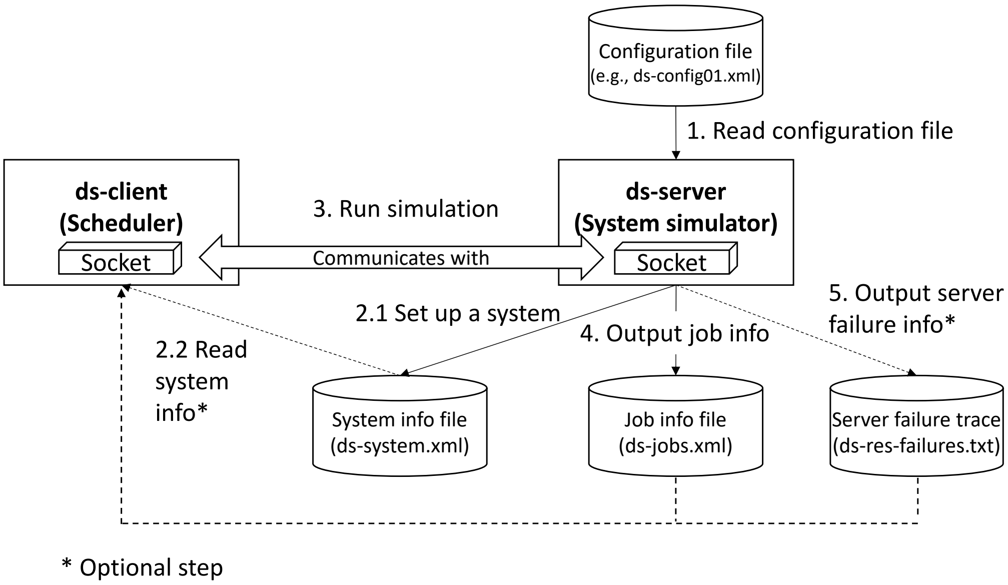

Figure 3: ds-sim workflow.

Scheduling is the key technique for ensuring the efficient use of computer systems including distributed systems. Job scheduling is the process of assigning jobs to system resources both time-wise and space-wise with one or more objectives and constraints, such as performance optimisation with respect to response time, resource utilisation and capital/operational cost, and deadline and budget constraints.

3 Architecture

`ds-sim` consists of two major components, `ds-server` and `ds-client` (Figure 3). They work in a client-server model via a socket as a communication channel. ds-server is a server-side system simulator that oversees the simulation. It simulates a distributed system, including servers and job submission/execution, with a user specified configuration (e.g., ds-sample-config01.xml) and pairs up with ds-client (a client-side simulator or job scheduler) that contains one or more scheduling algorithms. Up on the successful pairing, ds-server generates ds-system.xml for ds-client to read. This XML file is a subset of the configuration file containing only the information on the system (servers) to simulate.

The communication between System simulator and Scheduler in ds-sim is based on a simple protocol similar to Simple Mail Transfer Protocol (SMTP). For instance, Scheduler at the start of simulation sends HELO to System simulator then it gets OK from System simulator in response (see the complete protocol in the Simulation Protocol Section).

ds-sim logs all scheduling events–job submission, job start, job completion, server state information including server failure and recovery–into the output log ending with simulation statistics with respect to different performance metrics are also printed. These metrics include the number of servers used per type and the average utilization, the average turnaround time, total cost and system-wise utilisation. The output log can be used to visualize the simulation using ds-viz [2]. In addition to the main simulation log, there are two additional logs of simulation, ds-jobs.xml and ds-res-failures.txt. These two additional log files are used for two main purposes: (1) debugging/analysing the job scheduler and (2) reproducing simulation results by specifying them as inputs for a simulation. ds-res-failures.txt is only generated if server failures are simulated.

4 Simulation entities

Two main entities in ds-sim are jobs and servers.

4.1 Jobs

ds-sim currently only supports independent rigid jobs, i.e., no dependencies. In particular, a job can be seen as a single Linux process for which the run time remains the same irrespective of resources available (i.e., rigid). For example, if a job (with its resource requirements of 2 CPU cores, 500MB of RAM and 2GB of disk space) completes its execution in 300 seconds on a 2.4GHz quad-core server, its run time on a 3.2GHz octa-core server is still 300 seconds. For this reason, ds-sim does not distinguish clock speed of processors between servers or server types.

Jobs in ds-sim are generated based on the four key attributes of one or more job types as shown in Table 1. The actual job attributes are shown in Table 2 (see sample configuration files in Appendix B and Appendix D for more attributes generated after the simulation). While the former attributes are specified in the simulation configuration file (e.g., sample-config01.xml) before the simulation, the latter attributes are generated at the start of simulation and outputted to a jobs file, e.g., ds-jobs.xml.

XML attribute name	Data type	Data range
type	char array (string)	64
minRunTime	integer	[1, 604800 (or 7 days)]
maxRunTime	integer	[1, 604800]
populationRate[2]	integer	[1, 100]

Table 1: Job type attributes. The number of job types can be between 1 and 128 indexed by 0 and 127, respectively.

Description		XML attribute name	Data type	Data range
Identification	Number	id	integer	[0, 999999]
	Type	type	char array (string)	[0, 127]; 64[3]
Time (in sec)	Submission time	submitTime	integer	[0, 604799[4]]
	Estimated runtime	estRunTime	integer	[1, 604800]
Resource[5] requirements	CPU cores	cores	integer	[1, 1024]
	Memory (MB)	memory	integer	[1, 2147483647]
	Disk (MB)	disk	integer	[1, 2147483647]

Table 2: Actual job attributes.

The actual maximum resource requirements are bounded by the resource capacity of largest server type. In other words, resource requirements of any one job do not exceed resource capacities of a server of the largest type. For example, if a server of the largest type is with 64 cores, 8GB (8192MB) of RAM and 500GB (512000MB) of disk, the actual limits are bound by these values, not the absolute maximums, i.e., 1024, 2147483647 and 2147483647. The minimum memory/disk limit per core is 200 (MB).

The total number of jobs is dynamically determined by several factors: simulation duration, population rate, workload (i.e., minLoad and maxLoad in the workload element in the simulation configuration), and min and max runtimes.

The actual job attributes used in a simulation are as follows; they can be found in a jobs file.

- id: a sequence number based on the initial submission time
- type: an identifier of job category based on, for example, run time
- submitTime: it can be either the time of initial submission or the time of re-submission after the job is pre-empted/failed/killed
- estRunTime: the estimated runtime of a job is some arbitrary number within a certain predefined range based on the actual runtime; hence, it is not completely random.
- cores: the number of required CPU cores
- memory: the amount of required RAM (in MB)
- disk: the amount of required disk space (in MB)

[2] The sum of population rates must be 100%. In particular, the population rate of a job type is the percentage of jobs of that type among all jobs.

[3] While the number of job types can be between 1 and 128 (indexed by 0 and 127, respectively), the length of job type name is limited to 64 characters.

[4] The latest job submission time is 604799th second (or 7 days - 1 second) elapsed from the start of simulation.

[5] Resource requirements of jobs are not specified by the user. Rather, they are automatically generated based on primarily server resource capacities. See the note below for more detail.

A state of job can be one of the following.

- submitted (0): ds-server sends a job to ds-client (the job can be one resubmitted after it was pre-empted/failed/killed)
- waiting (1): job is waiting in a local queue of a server
- running (2): being executed on a server (no execution across multiple servers is possible in the current implementation)
- suspended (3): (not implemented yet)
- completed (4): job is successfully completed its execution
- pre-empted (5):
- failed (6): job failed due to server failure
- killed (7): job killed, e.g., by KILJ

4.2 Servers

A server in distributed systems is a networked computer or compute node equipped with its own resources, such as processors, memory and disk. Servers can be of either a physical machine or a virtual machine. In either case, it is assumed they do not interfere each other's performance, i.e., no/little resource contention or (near) perfect performance isolation.

Servers in ds-sim are identified by their type and id with the five key attributes as shown in Table 3.

XML attribute name	Data type	Data range
type	char array (string)	64
limit	integer	[1, 1000]
bootupTime (in sec)	integer	[0, 600]
hourlyRate (in $)	float	[0, 1000000]
cores	integer	[1, 1024]
memory	integer	[1, 2147483647]
disk	integer	[1, 2147483647]

Table 3: Server attributes. The maximum number of servers is 100,000 (1000/type * 100 types).

The actual server attributes used in a simulation are as follows. Unlike job attributes, these server attributes are not stored in any file; rather, they can be found by querying (using the GETS command) during the actual simulation.

- type: an identifier of server category; it can be any name within 64 characters
- id: a sequence number; a server is identified by type and id, e.g., small 2 and large 17
- limit: the number of servers of a particular type; server ids start from 0 to 'limit' - 1
- hourlyRate: the hourly rental rate in dollars; the actual charge is calculated for the number of seconds a server is used
- cores: the number of CPU cores
- memory: the amount of RAM (in MB)
- disk: the amount of disk space (in MB)

There are three other attributes that can be specified in the servers element, for enabling the simulation of resource/server failures: failureModel, failureTimeGrain and failureFile. Either the first two attributes or the last one should be used at any given time. In the former case, ds-sim generates server failures based on one of six failure models (failureModel) with failure frequency (failureTimeGrain), on-the-fly. In the latter case, ds-sim uses a failures file (ds-res-failures.txt) that has been generated in a previous simulation. Six failure modes are as follows: teragrid, nd07cpu, websites02, ldns04, pl05 and g5k06 (see [2] for more detail). Lower and upper bounds of server failure time granularity (failureTimeGrain) are 60 and 3600 (in seconds), respectively.

Each server maintains a queue. If a job is assigned/scheduled to a server that has sufficient resources readily available, it will run immediately. Otherwise, the job will be places in the local queue. In the meantime, a job can be placed in the global queue ('GQ') instead of being scheduled to a server when there is no server to run it immediately in particular.

A state of server can be one of the following.

- inactive (0): server is healthy, but not on (i.e., powered off)
- booting (1): server is booting; an inactive server automatically boots as soon as a job is scheduled to (i.e., no explicit booting command)
- idle (2): server is on and no job is running; all jobs on the server have completed the execution
- active (3): server is on and one or more jobs are running; the server's actual availability for immediate job execution should be determined by waiting jobs and spare resources available at the time of scheduling.
- unavailable (4): server is unable to run any job due to primarily server failure

5 Configuration

Simulation settings are configured by an XML configuration file (see an example below and Appendices A and B) supplied as an argument. There are four main parts of simulation configuration: `servers`, `jobs`, `workload` and `termination`. In addition, the first element `config` should have `randomSeed` attribute. The random seed can be any integer between 1 and 2147483647. The actual values of attributes are generated in random uniform distribution based on value ranges specified in the configuration file and the random seed specified. If the same configuration file is used with the same random seed (on a particular machine), the same values will be generated each time a simulation is run. Alternatively, job file (ds-jobs.xml) and/or resource failure file (ds-res-failures.txt) can be specified to reproduce simulation results.

Servers can be either homogeneous or heterogeneous in terms of their attributes, such as resource capacity and hourly (rental) rate. The heterogeneity can be specified with respect to server types. For instance, if there is only one server type (i.e., one "server" element, e.g., small), servers that will be generated are all of that type with the same attributes, such as the same bootup time, hourly rate and resource capacity. Servers of the same type are identified by their id's, e.g., small 0 for the first server of small type, small 1 for the second server of small type. Although the usage (monetary) cost is specified by an hourly rate, the actual calculation is based on the real uptime including idle time periods at the second granularity.

```
<?xml version="1.0" encoding="UTF-8"?>
<config randomSeed="1024">
<servers>
   <server type="x" limit="2" bootupTime="60" hourlyRate="0.2" coreCount="2" memory="10" disk="20" />
                  .
                  .
                  .
</servers>
<jobs>
   <job type="small" minRunTime="1" maxRunTime="300" populationRate="60" />
                  .
                  .
                  .
</jobs>
<workload type="random" minLoad="10" maxLoad="90" />
<termination>
   <condition type="endtime" value="604800" />
   <condition type="jobcount" value="10" />
</termination>
</config>
```

Jobs can also be of one or more types determined primarily by their length (minRunTime and maxRunTime). The range of job runtime should be between 1 and 604800 seconds (or 30 days). Each type should be specified with a population rate (populationRate), the proportion of jobs of all types. Clearly, the sum of rates must be 100 (100%). The actual number of jobs for a particular simulation is determined by several factors, such as job runtime ranges, workload (minLoad and maxLoad) and the termination condition. For example, a simulation with a job type with its runtime range from 1 to 30 and a load range of 80 and 100, there will be relatively much more jobs within any particular time interval, compared to that with a job type with its runtime range from 600 to 3600 and a load range of 10 and 20.

Workload varies between minimum and maximum load settings (minLoad and maxLoad). The load level indicates the busyness of the entire system. The load level dictates how many jobs to be generated/submitted considering their runtimes and the size of system (the number of servers). The current implementation of

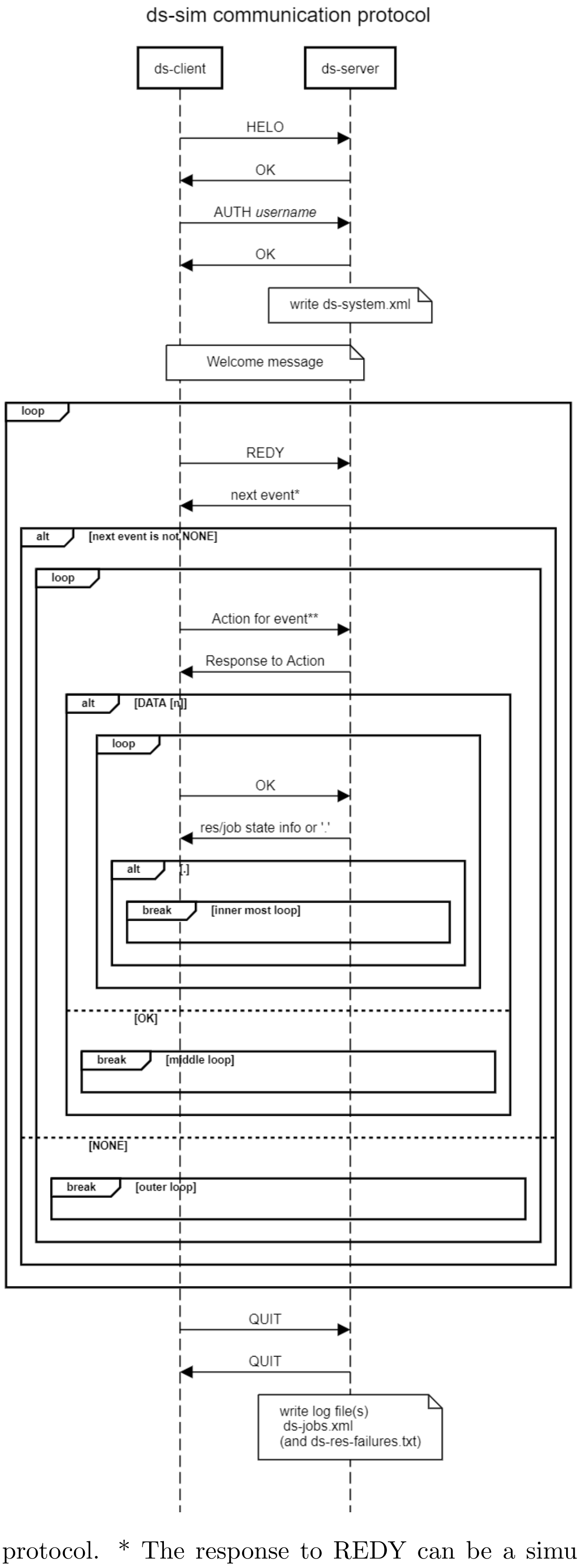

Figure 4: ds-sim simulation protocol. * The response to REDY can be a simulation event, such as JOBN, JCPL, RESF and NONE. ** Action as the response to event can vary depending on the event; for instance, GETS or SCHD is sent for JOBN. Note that sending REDY implies the response to the latest event is ignored (JCPL, RESF and RESR) or postponed (JOBN and JOBP).

ds-sim uses the ratio of total job core count to total server core count. The actual load shows some alternating pattern. In particular, the load continues to increase until it reaches the max load (maxLoad). Afterwards, it continues to decrease towards the min load (minLoad). This sequence iterates during the simulation. Note it might take a short time to reach minLoad at the start of simulation.

The simulation terminates upon meeting any of termination conditions specified in the `termination` element, more precisely, `endtime` and `jobcount`. For example, if endtime and jobcount are set to 36000 (10 hours) and 100, respectively, the simulation terminates either when the current simulation time is greater or equal to 36000 seconds or the 100th job has been processed. As the simulation end time is determined by the submission time of last job, the actual simulation end time may be greater (or sometimes far greater) than the set end time depending on several factors, such as scheduling decisions, server loads and/or server failure rates.

The values of server and job attributes, such as limit, bootupTime, minRunTime and maxRunTime must be within valid ranges as specified in Sections 4.1 and 4.2. Type names, except that in termination are all user-defined names.

6 Simulation protocol

ds-sim is designed to be language independent using the client-server model. ds-client (or any name the user chooses to use) must follow the communication protocol as shown in Figure 4.

6.1 Detailed steps

1. C(lient): sends HELO to S(erver)
2. S: replies with OK (e.g., 250 in SMTP)
3. C: sends AUTH with authentication information* "AUTH xxx" to S
4. S: replies with OK after printing out a welcome message and writing system info (ds-system.xml)
5. C: sends REDY to S, probably after reading ds-system.xml
6. S: sends one of the following:
   - JOBN: a normal job for scheduling for the first time
   - JOBP: a (preempted) job resubmitted for scheduling due to, e.g., server failure/termination or job termination (KILJ), a job dequeued (DEQJ)
   - JCPL: information on the latest job completion
   - RESF: information on the latest server failed
   - RESR: information on the latest server recovered
   - CHKQ: some jobs in the queue while no more new jobs to be submitted
   - NONE: there are no more jobs to schedule
7. C: sends one of the following or go to Step 12 for NONE
   - GETS: the request for server state information
   - SCHD: the actual scheduling decision
   - ENQJ: the request to enqueue a job to the global queue
   - DEQJ: the request to dequeue a job from the global queue
   - LSTQ: the request to get job information in the global queue
   - CNTJ: the request for job count on a specified server, of a particular job state
   - EJWT: the request for the sum of estimated waiting time
   - LSTJ: the request for information on running and waiting jobs on a particular server
   - MIGJ: the request to migrate a job
   - KILJ: the signal to kill a specific job
   - TERM: the request to terminate a server
8. S: sends one of the following corresponding messages to C
   - DATA n (for GETS, 'LSTQ *' and LSTJ): the preparation message that indicates the number of records (n), each of which is a set of state data fields per server, such as available resources and the current server state

- OK (for SCHD, ENQJ, DEQJ, MIGJ and KILJ): action is successfully done
- *n* (for CNTJ and 'LSTQ #'): the number of jobs
- time for EJWT: the sum of estimated waiting time on a specified server
- "*n* jobs killed" for TERM: the number of jobs killed due to server termination
- ERR (for any of scheduling actions): invalid request; invalid scheduling decision (in SCHD message) or invalid resource/job information request (in GETS/LSTJ message); for example, an invalid server ID or an invalid job ID has been used

9. C: takes one of the following actions upon receiving a message from S
   - send OK for DATA
   - go to Step 5 for OK, or
   - go to Step 7 for "ERR"
10. S: sends one of the following
    - resource information
    - information of jobs on a server, or
    - '.' if no more information to send
11. C: takes one of the following actions
    - reads the resource/jobs information, sends "OK" to S and go to Step 10, or
    - go to Step 7 for '.'
12. C: sends "QUIT" to S
13. S: sends "QUIT" to C and exits
14. C: exits

* The authentication information can be a username. It can be any arbitrary character string as the current implementation of ds-sim does no actual authentication. It is simply part of 'handshaking'.

7 ds-server options

The following are a list of command line options for ds-server. Many of these, such as -f, -n and -r can be specified in a configuration file. In particular, it is better to specify simulation parameters in a configuration file rather than in the command line. These command line argument options are also shown if "-h usage" is used.

- **b** - break simulation at job n; the simulation can run in an interactive mode still with ds-client
  -b *job_ID*
- **c** - use of a configuration file
  -c *configuration_ file_name (.xml)*
- **d** - simulation duration/end time
  -d n (in seconds)
- **f** - failure distribution model (e.g., teragrid, nd07cpu and websites02)
  -f teragrid|nd07cpu|websits02|g5k06|pl05|ldns04
- **g** - time granularity for resource failures
  -g n (in second)
- **h** - show help
  -h all|usage|limits|states
- **i** - run the simulation in the interactive mode without ds-client
  -i

- **j** - set job count
  -j *n* (max #jobs to generate)
- **l** - the number of servers (uniform for all "hard-coded" server types)
  -l *n*
- **n** - use the newline character ('\n') at the end of each message; this option is particularly useful for ds-client written in Java (or other languages)
  -n
- **o** - omit to send server failure event messages (RESF and RESR) for fast simulation; it is effective only if failures are simulated (i.e., "-f" is used)
  -o
- **p** - TCP/IP port number; note that when running multiple simulations concurrently with the '-p' option, each simulation (a particular ds-server and ds-client pair with the same port number) needs to run in a different directory as some simulation files, such as ds-system.xml and ds-jobs.xml are identical for every simulation. (min: 49152, max: 65535 and default: 50000)
  -p *n*
- **r** - set a random seed -r *n*
- **s** - set a scale factor for resource failures between 1 and 100; 1 for nearly no failures and 100 for original
  -s *n* (in percentage)
- **v** - verbose
  with **all**: show all messages; with **brief**: show a bit less than that with **all**, e.g., server status details in response to a GETS command are omitted; with **jobs**: show job status messages only; with **stats**: show only statistical data, e.g., server usage and the number of jobs completed.

  -v all|brief|jobs|stats[+]

8 Commands

ds-sim commands are designed in a similar manner to those for Simple Mail Transfer Protocol (SMTP). In particular, the length of most ds-sim commands is four characters, such as HELO, REDY and JOBN. In this section, we first give a table of all ds-sim commands in Table 4 followed by detailed description for each and every command.

HELO - greet ds-server

SYNOPSIS
    HELO

DESCRIPTION
    The HELO command greets ds-server at the very beginning of simulation. Up on the successful connection with ds-server, ds-client sends the HELO command to start the communication and simulation.

EXAMPLE
    HELO

OK - acknowledge the validity of the command received

SYNOPSIS
    OK

DESCRIPTION
    The OK command acknowledges the validity of the command received from either ds-server or ds-client.

Category	Command	Description
Connection	HELO	Initial message from client
	AUTH	Authentication information
	QUIT	Simulation termination
Preparation	REDY	Client signals server for a job
	DATA	The indicator for the actual information to be sent
Simulation event	JOBN	Job submission information
	JOBP	Job resubmission information after pre-emption and/or dequeuing
	JCPL	Job completion
	RESF	Server failure notice
	RESR	Server recovery notice
	CHKQ	Some jobs in the queue while there are no new jobs to be submitted
	NONE	No more jobs to schedule
Client action	GETS	Server information request
	SCHD	Scheduling decision
	ENQJ	Enqueue job
	DEQJ	Dequeue job
	LSTQ	List job information in a queue (currently, only the “global” queue)
	CNTJ	The number of jobs on a specified server with a particular state
	EJWT	The sum of estimated waiting time on a given server
	LSTJ	Job list of a server, i.e., all pending jobs (waiting and running jobs)
	MIGJ	Migrate a job from a source server to a destination server
	KILJ	Kill a job
	TERM	Server termination
Error	ERR	Error message
Acknowledgement	**OK**	Response to a valid command

Table 4: ds-sim command list. Commands in blue, red and black are used by ds-client, ds-server and **both**, respectively.

EXAMPLE
OK

AUTH - authenticate user

SYNOPSIS
AUTH *username*

DESCRIPTION
The AUTH command authenticates the user specified by *username*.

EXAMPLE
AUTH hojoo

REDY - signal ds-server for a next simulation event

SYNOPSIS
REDY

DESCRIPTION
The REDY command signals ds-server for a next simulation event.

EXAMPLE
REDY

JOBN - send a normal job

SYNOPSIS

JOBN *jobID submitTime core memory disk estRuntime*

DESCRIPTION

The JOBN command sends a normal job that is submitted for the first time. All fields are integer values. jobID starts from 0. Resource requirements (the number of CPU cores, the amount of memory and the amount of disk) do not exceed the resource capacity of a server of the largest type. The estRuntime is only an estimate.

EXAMPLE

JOBN 4 173 2 50 120 320 // a normal job of ID 2 submitted at time 173 (or the 172rd second from the start of simulation) with an estimated runtime of 320 seconds; the job requires 2 CPU cores, 50MB RAM and 120MB disk space.

JOBP - resend a job

SYNOPSIS

JOBP *jobID submitTime core memory disk estRuntime*

DESCRIPTION

The JOBP command resends a job that was previously submitted but failed to complete. A job submitted by JOBP is due to one of the following reasons: failed (currently the server on which it was scheduled was failed), killed (e.g., by KILJ) and pre-empted (a backfilled job was pre-empted due to priority). The job must start from the beginning, i.e., no checkpointing. All fields are integer values. jobID starts from 0. Resource requirements (the number of CPU cores, the amount of memory and the amount of disk) do not exceed the resource capacity of a server of the largest type. The estRuntime is only an estimate.

EXAMPLE

JOBP 12 2142 4 250 800 750

JCPL - provide the information on most recent job completion

SYNOPSIS

JCPL *endTime jobID serverType serverID*

DESCRIPTION

The JCPL command sends the information of most recent job completion. The response from ds-client can vary depending on its scheduling policy. For example, a job on another server can migrate (MIGJ) to the server that the job just completed on, or the server can be terminated (TERM) if no other jobs are waiting/running, to save costs.

EXAMPLE

JCPL 8297 12 medium 2

RESF - notify a server failure

SYNOPSIS

RESF *serverType serverID timeOfFailure*

DESCRIPTION

The RESF command notifies a server failure. Jobs on the failed server, regardless of their state (e.g., Waiting, Running and Suspended) are all terminated. No state will be stored, i.e., no checkpointing for resuming their execution. All terminated jobs will be resubmitted by the JOBP command with their submission time being set to the time of failure (timeOfFailure). Failures can be simulated either based on a failure model specified by the -f command line option or failureModel attribute in the configuration file, or a resource failures log file (e.g., ds-res-failures.txt).

EXAMPLE

RESF MQxlarge 122 20272

RESR - notify server recovery

SYNOPSIS

RESR *serverType serverID timeOfRecovery*

DESCRIPTION

The RESR command notifies the recovery of a server after its failure. The time to recover (mean time to recovery or MTTR) is randomised and thus, unable to be accurately estimated. The state of server will be set to inactive.

EXAMPLE

RESR MQxlarge 122 21159

CHKQ - indicate no new jobs, but some in the queue

SYNOPSIS

CHKQ

DESCRIPTION

The CHKQ command indicates there are one or more jobs in the queue while there are no more new jobs to be submitted. The jobs in the queue have been resulted from ENQJ or pre-emption, e.g., jobs failed/killed/terminated/pre-empted.

EXAMPLE

CHKQ

NONE - indicate no more jobs

SYNOPSIS

NONE

DESCRIPTION

The NONE command indicates there are no more jobs to be scheduled including any failed/killed/terminated/pre-empted jobs.

EXAMPLE

NONE

GETS - query server state information

SYNOPSIS

GETS All|Type *serverType*|Capable *core memory disk*|Avail *core memory disk*

DESCRIPTION

The GETS command queries server state information at the current simulation time. The All option requests the information on all servers regardless of their state including inactive and unavailable. The Type option requests the information on servers of a specified type (serverType) regardless of their state, too. The Capable and Avail options make requests for server state information based on initial resource capacity and the current resource availability, respectively. For instance, GETS Capable 3 500 1000 and GETS Avail 3 500 1000 are different in that the response to the former is all servers that can "eventually" provide 3 cores, 500MB of memory and 1000MB of disk regardless of their current availability. Meanwhile, the response to the latter is all servers that can "immediately" provide 3 cores, 500MB of memory and 1000MB of disk. With the Avail option, if there are insufficient available resources and/or waiting jobs, the server is not available for the job.

In general, it is recommended to use the Capable option than the Avail option as the system is often busy, i.e., all servers are running one or more jobs at any given point in time. In the case of no servers are available (Avail), the message right after the DATA message will be ‘.’.

EXAMPLE

```
    JOBN 3 101 2 900 2500 380 // ds-server
    GETS Capable 2 900 2500 // ds-client; consider the current time is 101
    DATA 5 123 // ds-server
    OK // ds-client
    juju 0 booting 120 0 2500 13100 1 0 // ds-server; the entire server state information
    juju 1 booting 156 0 2500 13900 1 0 // is sent as a single message delimited by
    joon 0 active 97 1 15300 60200 0 1 // a newline character (‘\n’).
    joon 1 inactive -1 4 16000 64000 0 0
    super-silk 0 inactive -1 16 64000 512000 0 0
    OK // ds-client
    . // ds-server
```

The state information on each server is formatted in *serverType serverID state curStartTime core memory disk #wJobs #rJobs [#failures totalFailtime mttf mttr madf lastStartTime]*

Server ***state*** is one of five states, inactive, booting, idle, active and unavailable. While an inactive (powered off) server can be used to run a job, an unavailable server cannot run any job, e.g., the server is being recovered. An idle server is one that has been powered on, but it has no jobs waiting/running; hence, it is immediately available. An active server is one that is currently running one or more jobs. Note that for a job to be scheduled, an active server with sufficient available/spare resources is not able to immediately run the job if there is any waiting job since no backfilling is allowed.

The current start time of a server (***curStartTime***) is the start time of current use; it is the time of actual use after booting as in 120 and 156 for juju 0 and juju 1 above, respectively. If a server is inactive (e.g., joon 1) or unavailable, the current start time will be unknown (-1). Obviously, the current start time changes if a server is rebooted after a termination (TERM) or a failure. The available time of a server can be estimated based on the current start time or the current (simulation) time and estimated runtimes of running/waiting jobs. In particular, the available time of an inactive server is the current time + bootup time (e.g., 60 seconds for juju and joon, respectively and 80 seconds for super-silk). For a given job, the available time of an idle server is instant as long as the server is capable of running the job whereas that of an active server is based on not simply its spare/available resources, but also the existence of waiting jobs. The available time of an active server cannot be accurately estimated (1) if the active server has not sufficient immediately available resources and (2) the active server has one or more waiting jobs. In the meantime, if an active server has sufficient available resources for a given job, its available time is the same as the current simulation time, i.e., it is instantly available. The available time of a booting server is similar to that of an active server with one exception that the server is only available once the booting completes regardless of the resource availability.

The resources fields (***core memory disk***) are the available resources. Note that this availability should not be directly interpreted as the readiness for immediate job execution; see the discussion above on the available time.

The last two fields (***#wJobs and #rJobs***) are the number of waiting jobs and the number of running jobs. Their details, such as job ID, state, start time, estimated runtime and resource requirements can be found by the LSTJ command. In the above example, although all servers are capable of running the job (job 3), juju 0, juju 1 and joon 0 are currently not immediately available since each of the first two has a waiting job and joon 0 is running a job.

If resource failures are simulated (either with the “-f” command line option or by specifying failure details in the configuration file), there are six more fields, #failures, total amount of failure times, mean time to failure, mean time to recovery, mean absolute deviation of failure and the last start time of a server (either the initial start time of server or the start after a recovery).

* For the DATA message, see the DATA command description below.

SCHD - schedule a job

SYNOPSIS

SCHD *jobID serverType serverID*

DESCRIPTION

The SCHD command schedules a job (jobID) to the server (serverID) of serverType.

EXAMPLE

```
SCHD 3 joon 1
```

ENQJ - enqueue a job

SYNOPSIS

```
ENQJ queue_name // e.g., 'GQ'
```

DESCRIPTION

The ENQJ command places the current job to a specified queue. The name used for the global queue is GQ.

EXAMPLE

```
ENQJ GQ
```

DEQJ - dequeue a job

SYNOPSIS

```
DEQJ queue_name qID // e.g., 'GQ'
```

DESCRIPTION

The DEQJ command gets the job at the specified queue position (qID) from the specified queue.

EXAMPLE

```
DEQJ GQ 2 // job at the third place (qID of 2 starting from 0) of the global queue
```

LSTQ - get job information in a queue

SYNOPSIS

```
LSTQ queue_name i|$|#|*
```

DESCRIPTION

The LSTQ command gets job information in the specified queue.

EXAMPLE

```
LSTQ GQ 3
0 3 173 173 79 2 500 1000 // jobID qID job_submit_time queued_time estimated_runtime #cores memory disk

LSTQ GQ #
14 // 14 jobs in the queue

LSTQ GQ *
DATA 4 55 // 4 jobs and the message length of each job is 55 characters
OK
3 0 57 57 107 3 400 1000
5 1 61 61 82 2 100 300
11 2 219 219 37 1 200 200
47 3 1029 1029 520 4 2000 1000
OK
.
```

CNTJ - query job count of a state

SYNOPSIS

CNTJ *serverType serverID jobState*

DESCRIPTION

The CNTJ command queries the number of jobs of a specified state, on a specified server. The job state is specified by one of state codes, except 0 for 'submitted'.

EXAMPLE

CNTJ joon 0 2 // query the number of running jobs on joon 0
1 // the response from ds-server, i.e., 1 running job on joon 0

EJWT - query the total estimated waiting time

SYNOPSIS

EJWT *serverType serverID*

DESCRIPTION

The EJWT command queries the sum of estimated waiting times on a given server. It does not take into account the remaining runtime of running jobs. Note that the calculation should not be considered to be accurate because (1) it is based on estimated runtimes of waiting jobs and (2) more importantly, it does not consider the possibility of parallel execution of waiting jobs.

EXAMPLE

EJWT MQ2xlarge 3
218 // the response from ds-server

LSTJ - query the list of running/waiting jobs

SYNOPSIS

LSTJ *serverType serverID*

DESCRIPTION

The LSTJ command queries the list of running and waiting jobs on a given server. The response to LSTJ is formatted in *jobID jobState submitTime startTime estRunTime core memory disk.* The job state is sent as a state code either 1 or 2 for waiting and running, respectively. The response will be a sequence of DATA, a series of job information and OK message pairs and '.'.

EXAMPLE

LSTJ medium 3
DATA 3 59 // 3 jobs and the length of each message is 59 character long
OK
2 2 139 1208 172 2 100 200
7 2 192 1224 328 1 120 450
11 1 324 -1 49 4 380 1000 // -1 for unknown start time since the job 11 is waiting
OK
.

MIGJ - migrate a job

SYNOPSIS

MIGJ *jobID srcServerType srcServerID tgtServerType tgtServerID*

DESCRIPTION

The MIGJ command migrates a job specified by jobID on srcServerID of srcServerType to tgtServerID of tgtServerType. The job can be of waiting, running or suspended. The successful migration results in the same behaviour of normal scheduling action. In particular, the job's state on the target server is determined by the common criteria of job execution, such as the resource availability and running/waiting jobs of the target server. The migrated job will "restart" on the target server.

EXAMPLE

MIGJ 572 small 2 medium 7
OK

KILJ - kill a job

SYNOPSIS

KILJ *serverType serverID jobID*

DESCRIPTION

The KILJ command kills a job. The killed job is pushed back to the queue with the killed time as a new submission time. The job will be resubmitted with JOBP.

EXAMPLE

KILJ tiny 17 249
OK

TERM - terminate a server

SYNOPSIS

TERM *serverType serverID*

DESCRIPTION

The TERM command terminates a server. All waiting/running jobs are killed and re-submitted for scheduling with JOBP. The server is then put into the inactive state.

EXAMPLE

TERM veryveryveryLarge 5
4 jobs killed // the response from ds-server

DATA - indicate one or more messages to be sent

SYNOPSIS

DATA *nRecs recLen*

DESCRIPTION

The DATA command is sent in response to either GETS or LSTJ. The response to GETS is in the format of DATA nRecs recLen. The actual message following is a set of nRecs records as a single message; these records are delimited by the newline character (‘`\n`’). recLen is the maximum length of one record in bytes including spaces between fields. recLen should not be explicitly used to read individual records. Rather the delimiter (the newline character, ‘`\n`’) should be used to separate records. In other words, recLen should be used to estimate/allocate space needed to read the message, the batch message in the case GETS in particular.

EXAMPLE

DATA 5 123

ERR - indicate an error

SYNOPSIS

ERR: *error message*

DESCRIPTION

The ERR command sends an error message from ds-server. Two typical causes of an error are sending a message in a wrong format by ds-client and sending an infeasible command.

EXAMPLE

ERR: Server incapable of running such a job

QUIT - quit simulation

SYNOPSIS

QUIT

DESCRIPTION

The QUIT command terminates the simulation. When there are no more jobs to schedule indicated by NONE, ds-client sends QUIT to ds-server signalling to terminate the simulation. ds-server then responds with QUIT and performs housekeeping, such as writing ds-jobs.xml and ds-res-failures.txt.

EXAMPLE

QUIT

9 How to use ds-sim

The usage of ds-sim is simple as shown in the following.

1. Download/clone ds-sim from the git repository. To clone, type the following:
   `git clone https://github.com/distsys-MQ/ds-sim.git`[6]
   Once cloned, you will see the ds-sim directory under the current directory.

2. Open two terminals by either clicking the terminal icon or pressing the shortcut key (Ctrl+Alt+t in Windows or control+command+t in MacOS).

3. Go to the pre-compiled directory (ds-sim/src/pre-compiled/) and make sure both ds-server and ds-client[7] have the 'execute' permission[8].

4. Run ds-server in one terminal, e.g., `./ds-server -c sample-config01.xml -v brief`. ds-server will show the welcome message and wait ds-client to connect.

5. Run ds-client in other terminal, e.g., `./ds-client`. By default, ds-client will use the ATL algorithm (all jobs assigned to the first server of the largest type). If you'd like to use a different scheduling algorithm, use the "-a" option, e.g., ./ds-client -a ff. The simulation then runs. It typically completes in seconds unless exceptional cases are expected, such as server failures.

If you would like to compile ds-sim (ds-server in particular) yourself, you may need to install required libraries as follows.

1. Make sure you have downloaded/cloned ds-sim including the simulator source files

2. Check whether the following libraries are installed on your machine, e.g., `apt list --installed` on Ubuntu: libxml2 and libxml2-dev.

3. If they're not installed, install with the following command in a terminal window.
   `sudo apt-get install libxml2 libxml2-dev`

4. Go to the `src` directory

5. run the make command (`make`); make sure source files (e.g., ds-server.c) and `Makefile` exist

[6] If you don't have git installed already, you will be asked to install it first.

[7] ds-client from the repository is a reference implementation with baseline scheduling algorithms, such as ATL, LRR, FF, BF, WF and FC. To implement your own scheduling algorithms, you have to write your own *ds-client* (or any name of your choice).

[8] You can check permissions with the `ls` command. To give the execute permission, use the chmod command, e.g., `chmod +x ds-server`.

10 Assumptions

- Resource requirements of any one job do not exceed resource capacities of a server of the largest type.
- Servers are homogeneous in terms of processing power; that is, the runtime of a job stays the same regardless of the scheduled server.
- A server can run one or more jobs in parallel without performance degradation (from resource contention) as long as it can satisfy resource requirements of all co-located jobs.
- A job is only schedulable on a single server; that is, jobs are not to be executed across multiple servers.
- The runtime of a job is only an estimate; the exact runtime is unknown.

Appendix A: XML Schema for ds-sim configuration

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="terminationType">
        <xs:all>
            <xs:element name="condition">
        </xs:all>
        <xs:attribute name="type" type="xs:string" use="required"/>
        <xs:attribute name="value" type="xs:integer" use="required"/>
    </xs:complexType>

    <xs:complexType name="workload">
        <xs:simpleContent>
            <xs:attribute name="type" type="xs:string" use="required"/>
            <xs:attribute name="minLoad" type="xs:integer" use="required"/>
            <xs:attribute name="maxLoad" type="xs:integer" use="required"/>
        </xs:simpleContent>
    </xs:complexType>

    <xs:complexType name="jobType">
        <xs:simpleContent>
            <xs:attribute name="type" type="xs:string" use="required"/>
            <xs:attribute name="minRunTime" type="xs:integer" use="required"/>
            <xs:attribute name="maxRunTime" type="xs:integer" use="required"/>
            <xs:attribute name="populationRate" type="xs:integer" use="required"/>
        </xs:simpleContent>
    </xs:complexType>

    <xs:complexType name="jobsType">
        <xs:all>
            <xs:element name="job" type="jobType"/>
        </xs:all>
        <xs:attribute name="file" type="xs:string" use="optional"/>
    </xs:complexType>

    <xs:complexType name="serverType">
        <xs:simpleContent>
            <xs:attribute name="type" type="xs:string" use="required"/>
            <xs:attribute name="limit" type="xs:integer" use="required"/>
            <xs:attribute name="bootupTime" type="xs:integer" use="required"/>
            <xs:attribute name="hourlyRate" type="xs:double" use="required"/>
            <xs:attribute name="cores" type="xs:integer" use="required"/>
            <xs:attribute name="memory" type="xs:integer" use="required"/>
            <xs:attribute name="disk" type="xs:integer" use="required"/>
        </xs:simpleContent>
        </xs:complexType>

    <xs:complexType name="serversType">
        <xs:all>
            <xs:element name="server" type="serverType"/>
        </xs:all>
        <xs:attribute name="failureFile" type="xs:string" use="optional"/>
        <xs:attribute name="failureModel" type="xs:string" use="optional"/>
        <xs:attribute name="failureGrain" type="xs:integer" use="optional"/>
    </xs:complexType>

    <xs:complexType name="configType">
        <xs:all>
            <xs:element name="servers" type="serversType"/>
            <xs:element name="jobs" type="jobsType"/>
            <xs:element name="workload" type="workloadType"/>
            <xs:element name="termination" type="terminationType"/>
        </xs:all>
    </xs:complexType>

    <xs:element name="config" type="configType"/>
</xs:schema>
```

Appendix B: sample configuration files

sample-config01.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by: Y. C. Lee -->
<config randomSeed="1024">
  <servers>
       <server type="juju" limit="2" bootupTime="60" hourlyRate="0.2" cores="2" memory="4000" disk="16000" />
       <server type="joon" limit="2" bootupTime="60" hourlyRate="0.4" cores="4" memory="16000" disk="64000" />
       <server type="super-silk" limit="1" bootupTime="80" hourlyRate="0.8" cores="16" memory="64000" disk="512000" />
  </servers>
  <jobs>
       <job type="short" minRunTime="1" maxRunTime="300" populationRate="60" />
       <job type="medium" minRunTime="301" maxRunTime="1800" populationRate="30" />
       <job type="long" minRunTime="1801" maxRunTime="100000" populationRate="10" />
  </jobs>
  <workload type="moderate" minLoad="30" maxLoad="70" />
  <termination>
       <condition type="endtime" value="604800" />
       <condition type="jobcount" value="10" />
  </termination>
</config>
```

sample-config02.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by: Y. C. Lee -->
<config randomSeed="1024">
  <servers>
       <server type="offload" limit="2" bootupTime="40" hourlyRate="0.2" cores="2" memory="4000" disk="16000" />
       <server type="highway" limit="2" bootupTime="60" hourlyRate="0.4" cores="4" memory="16000" disk="64000" />
       <server type="silkroad" limit="2" bootupTime="80" hourlyRate="0.8" cores="8" memory="64000" disk="512000" />
  </servers>
  <jobs>
       <job type="short" minRunTime="1" maxRunTime="300" populationRate="30" />
       <job type="medium" minRunTime="301" maxRunTime="1800" populationRate="50" />
       <job type="long" minRunTime="1801" maxRunTime="100000" populationRate="20" />
  </jobs>
  <workload type="moderate" minLoad="50" maxLoad="90" />
  <termination>
       <condition type="endtime" value="604800" />
       <condition type="jobcount" value="100" />
  </termination>
</config>
```

sample-config03.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by: Y. C. Lee -->
<config randomSeed="1024">
  <servers>
       <server type="tiny" limit="16" bootupTime="60" hourlyRate="0.1" cores="1" memory="2000" disk="16000" />
       <server type="small" limit="16" bootupTime="60" hourlyRate="0.2" cores="2" memory="4000" disk="32000" />
       <server type="medium" limit="8" bootupTime="60" hourlyRate="0.4" cores="4" memory="8000" disk="64000" />
       <server type="large" limit="4" bootupTime="60" hourlyRate="0.8" cores="8" memory="16000" disk="128000" />
       <server type="xlarge" limit="4" bootupTime="60" hourlyRate="1.6" cores="16" memory="32000" disk="256000" />
  </servers>
  <jobs>
       <job type="instant" minRunTime="1" maxRunTime="30" populationRate="10" />
       <job type="short" minRunTime="31" maxRunTime="180" populationRate="30" />
       <job type="medium" minRunTime="181" maxRunTime="600" populationRate="30" />
       <job type="long" minRunTime="601" maxRunTime="1800" populationRate="20" />
       <job type="verylong" minRunTime="1801" maxRunTime="100000" populationRate="10" />
  </jobs>
  <workload type="medium" minLoad="60" maxLoad="90" />
  <termination>
       <condition type="endtime" value="986400" />
       <condition type="jobcount" value="2000" />
  </termination>
</config>
```

sample-config04.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by: Y. C. Lee -->
<config randomSeed="1024">
  <servers>
       <server type="small" limit="10" bootupTime="60" hourlyRate="0.2" cores="2" memory="4000" disk="16000" />
```

```
      <server type="medium" limit="10" bootupTime="60" hourlyRate="0.4" cores="4" memory="16000" disk="64000" />
      <server type="large" limit="10" bootupTime="80" hourlyRate="0.8" cores="16" memory="64000" disk="512000" />
      <server type="MQ_Cloud_large" limit="10" bootupTime="70" hourlyRate="0.6" cores="8" memory="32000" disk="256000" />
      <server type="very_very_very_very_very_very_large" limit="10" bootupTime="120" hourlyRate="3.2" cores="64" memory="256000"
          ↪  disk="2049000" />
  </servers>
  <jobs>
      <job type="instant" minRunTime="1" maxRunTime="30" populationRate="10" />
      <job type="short" minRunTime="31" maxRunTime="180" populationRate="30" />
      <job type="medium" minRunTime="181" maxRunTime="600" populationRate="30" />
      <job type="long" minRunTime="601" maxRunTime="1800" populationRate="20" />
      <job type="verylong" minRunTime="1801" maxRunTime="100000" populationRate="10" />
  </jobs>
  <workload type="towardshigh" minLoad="60" maxLoad="80" />
  <termination>
      <condition type="endtime" value="2592000" />
      <condition type="jobcount" value="1000" />
  </termination>
</config>
```

sample-config05.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by: Y. C. Lee -->
<config randomSeed="202102">
  <servers>
      <server type="tiny" limit="100" bootupTime="60" hourlyRate="0.1" cores="1" memory="2000" disk="8000" />
      <server type="small" limit="80" bootupTime="60" hourlyRate="0.2" cores="2" memory="4000" disk="16000" />
      <server type="medium" limit="40" bootupTime="60" hourlyRate="0.4" cores="4" memory="8000" disk="32000" />
      <server type="16xlarge" limit="10" bootupTime="120" hourlyRate="25.6" cores="256" memory="512000" disk="4096000" />
      <server type="large" limit="20" bootupTime="80" hourlyRate="0.8" cores="8" memory="16000" disk="64000" />
      <server type="xlarge" limit="20" bootupTime="80" hourlyRate="1.6" cores="16" memory="32000" disk="128000" />
      <server type="2xlarge" limit="10" bootupTime="100" hourlyRate="3.2" cores="32" memory="64000" disk="256000" />
      <server type="4xlarge" limit="10" bootupTime="100" hourlyRate="6.4" cores="64" memory="128000" disk="512000" />
      <server type="8xlarge" limit="10" bootupTime="120" hourlyRate="12.8" cores="128" memory="256000" disk="1024000" />
  </servers>
   <jobs>
      <job type="instant" minRunTime="1" maxRunTime="20" populationRate="10" />
      <job type="short" minRunTime="60" maxRunTime="1800" populationRate="30" />
      <job type="medium" minRunTime="3600" maxRunTime="10800" populationRate="30" />
      <job type="long" minRunTime="18000" maxRunTime="604800" populationRate="30" />
  </jobs>
  <workload type="overloaded" minLoad="80" maxLoad="100" />
  <termination>
      <condition type="endtime" value="2592000" />
      <condition type="jobcount" value="5000" />
  </termination>
</config>
```

ds-sample-config05-failureFile.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by: Y. C. Lee -->
<config randomSeed="2000">
  <servers failureFile="ds-sample-config05-failures.txt">
      <server type="tiny" limit="20" bootupTime="60" hourlyRate="0.1" cores="1" memory="1000"
      disk="4000" />
      <server type="small" limit="20" bootupTime="60" hourlyRate="0.2" cores="2" memory="4000"
      disk="16000" />
      <server type="medium" limit="20" bootupTime="60" hourlyRate="0.4" cores="4" memory="16000"
      disk="64000" />
      <server type="large" limit="20" bootupTime="60" hourlyRate="0.8" cores="8" memory="32000"
      disk="256000" />
      <server type="xlarge" limit="20" bootupTime="60" hourlyRate="1.6" cores="16" memory="64000"
      disk="512000" />
  </servers>
  <jobs>
      <job type="instant" minRunTime="1" maxRunTime="30" populationRate="15" />
      <job type="short" minRunTime="11" maxRunTime="180" populationRate="40" />
      <job type="medium" minRunTime="301" maxRunTime="900" populationRate="30" />
      <job type="long" minRunTime="1801" maxRunTime="43200" populationRate="10" />
      <job type="verylong" minRunTime="3601" maxRunTime="100000" populationRate="5" />
  </jobs>
  <workload type="light" minLoad="10" maxLoad="30" />
  <termination>
      <!-- simulation terminates whichever condition meets first -->
      <!-- 60 seconds * 60 minutes * 24 hours * 3 days = 129600 seconds -->
      <condition type="endtime" value="129600" />
      <condition type="jobcount" value="500" />
  </termination>
</config>
```

* ds-sample-config05-failures.txt should have been produced with the same configuration prior to using it here, but without the “failureFile” attribute, e.g., ./ds-server -c ds-sample-config05.xml -f teragrid. The actual failures file name is ds-res-failures.txt that has been used after changing its name to ds-sample-config05-failures.txt.

ds-sample-config05-failures_jobs.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by: Y. C. Lee -->
<config randomSeed="2000">
  <servers failureFile="ds-sample-config05-failures.txt">
       <server type="tiny" limit="20" bootupTime="60" hourlyRate="0.1" cores="1" memory="1000"
       disk="4000" />
       <server type="small" limit="20" bootupTime="60" hourlyRate="0.2" cores="2" memory="4000"
       disk="16000" />
       <server type="medium" limit="20" bootupTime="60" hourlyRate="0.4" cores="4" memory="16000"
       disk="64000" />
       <server type="large" limit="20" bootupTime="60" hourlyRate="0.8" cores="8" memory="32000"
       disk="256000" />
       <server type="xlarge" limit="20" bootupTime="60" hourlyRate="1.6" cores="16" memory="64000"
       disk="512000" />
  </servers>
  <jobs file="ds-sample-config05-jobs.xml"/>
</config>
```

* ds-sample-config05-jobs.xml should have been produced with the same configuration prior to using it here. Like ds-sample-config05-failures.txt, after each simulation run, ds-jobs.xml is generated. In other words, ds-sample-config05-jobs.xml has been created simply by changing the file name from ds-jobs.xml.

Appendix C: a sample system information file

ds-system.xml for sample-config01.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- system information generated for ds-sim@MQ, 28-Feb, 2023 @ MQ - client-server -->
<system>
       <servers>
              <server type="juju" limit="2" bootupTime="60" hourlyRate="0.20" cores="2" memory="4000" disk="16000" />
              <server type="joon" limit="2" bootupTime="60" hourlyRate="0.40" cores="4" memory="16000" disk="64000" />
              <server type="super-silk" limit="1" bootupTime="80" hourlyRate="0.80" cores="16" memory="64000" disk="512000" />
       </servers>
</system>
```

* ds-system.xml will be overwritten each time you run a simulation. ds-client may read it.

Appendix D: a sample jobs file

ds-jobs.xml for sample-config01.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- job list for ds-sim, 28-Feb, 2023 @ MQ - client-server -->
<jobs>
       <type name="short" minRunTime="1" maxRunTime="300" populationRate="60" />
       <type name="medium" minRunTime="301" maxRunTime="1800" populationRate="30" />
       <type name="long" minRunTime="1801" maxRunTime="100000" populationRate="10" />
       <job id="0" type="medium" submitTime="37" estRunTime="653" cores="3" memory="700" disk="3800" />
       <job id="1" type="medium" submitTime="60" estRunTime="2025" cores="2" memory="1500" disk="2900" />
       <job id="2" type="medium" submitTime="96" estRunTime="343" cores="2" memory="1500" disk="2100" />
       <job id="3" type="medium" submitTime="101" estRunTime="380" cores="2" memory="900" disk="2500" />
       <job id="4" type="short" submitTime="137" estRunTime="111" cores="1" memory="100" disk="2000" />
       <job id="5" type="short" submitTime="156" estRunTime="8" cores="3" memory="2700" disk="2600" />
       <job id="6" type="medium" submitTime="198" estRunTime="1074" cores="4" memory="4000" disk="7600" />
       <job id="7" type="medium" submitTime="225" estRunTime="442" cores="2" memory="500" disk="2100" />
       <job id="8" type="medium" submitTime="249" estRunTime="926" cores="1" memory="100" disk="800" />
       <job id="9" type="medium" submitTime="308" estRunTime="2010" cores="2" memory="600" disk="1500" />
</jobs>
```

* ds-jobs.xml will be overwritten each time you run a simulation.

Appendix E: a sample simulation log

The following simulation log is gathered from ./ds-server -c sample-config01.xml -v all and ./ds-client -a bf with the best-fit (BF) scheduling algorithm.

```
# ds-sim server 14-Jan, 2024 @ MQ - client-server
# Server-side simulator started with './ds-server -c ../sample-configs/sample-config01.xml -v all'
# Waiting for connection to port 50000 of IP address 127.0.0.1
RCVD HELO
SENT OK
RCVD AUTH yclee
# Welcome yclee!
# The system information can be read from 'ds-system.xml'
SENT OK
RCVD REDY
t 37: job 0 -- CPU: 3, Mem: 700, Disk: 3800, Est Runtime: 653 SUBMITTED
      SENT JOBN 0 37 3 700 3800 653
      RCVD GETS Capable 3 700 3800
      SENT DATA 3 124
      RCVD OK
      SENT joon 0 inactive -1 4 16000 64000 0 0
           joon 1 inactive -1 4 16000 64000 0 0
           super-silk 0 inactive -1 16 64000 512000 0 0
      RCVD OK
      SENT .
      RCVD SCHD 0 joon 0
      job 0 (waiting) on joon 0 (booting) SCHEDULED
      SENT OK
      RCVD REDY
t 60: job 1 -- CPU: 2, Mem: 1500, Disk: 2900, Est Runtime: 2025 SUBMITTED
      SENT JOBN 1 60 2 1500 2900 2025
      RCVD GETS Capable 2 1500 2900
      SENT DATA 5 124
      RCVD OK
      SENT juju 0 inactive -1 2 4000 16000 0 0
           juju 1 inactive -1 2 4000 16000 0 0
           joon 0 booting 97 1 15300 60200 1 0
           joon 1 inactive -1 4 16000 64000 0 0
           super-silk 0 inactive -1 16 64000 512000 0 0
      RCVD OK
      SENT .
      RCVD LSTJ joon 0
      SENT DATA 1 59
      RCVD OK
      SENT 0 1 37 97 653 3 700 3800
      RCVD OK
      SENT .
      RCVD SCHD 1 juju 0
      job 1 (waiting) on juju 0 (booting) SCHEDULED
      SENT OK
      RCVD REDY
t 96: job 2 -- CPU: 2, Mem: 1500, Disk: 2100, Est Runtime: 343 SUBMITTED
      SENT JOBN 2 96 2 1500 2100 343
      RCVD GETS Capable 2 1500 2100
      SENT DATA 5 124
      RCVD OK
      SENT juju 0 booting 120 0 2500 13100 1 0
           juju 1 inactive -1 2 4000 16000 0 0
           joon 0 booting 97 1 15300 60200 1 0
           joon 1 inactive -1 4 16000 64000 0 0
           super-silk 0 inactive -1 16 64000 512000 0 0
      RCVD OK
      SENT .
      RCVD LSTJ juju 0
      SENT DATA 1 59
      RCVD OK
      SENT 1 1 60 120 2025 2 1500 2900
      RCVD OK
      SENT .
      RCVD LSTJ joon 0
      SENT DATA 1 59
      RCVD OK
      SENT 0 1 37 97 653 3 700 3800
      RCVD OK
      SENT .
      RCVD SCHD 2 juju 1
      job 2 (waiting) on juju 1 (booting) SCHEDULED
      SENT OK
      RCVD REDY
t 97: job 0 on joon 0 RUNNING
t 101: job 3 -- CPU: 2, Mem: 900, Disk: 2500, Est Runtime: 380 SUBMITTED
       SENT JOBN 3 101 2 900 2500 380
       RCVD GETS Capable 2 900 2500
       SENT DATA 5 124
       RCVD OK
       SENT juju 0 booting 120 0 2500 13100 1 0
            juju 1 booting 156 0 2500 13900 1 0
            joon 0 active 97 1 15300 60200 0 1
            joon 1 inactive -1 4 16000 64000 0 0
            super-silk 0 inactive -1 16 64000 512000 0 0
       RCVD OK
       SENT .
       RCVD LSTJ juju 0
       SENT DATA 1 59
       RCVD OK
       SENT 1 1 60 120 2025 2 1500 2900
       RCVD OK
       SENT .
       RCVD LSTJ juju 1
       SENT DATA 1 59
       RCVD OK
       SENT 2 1 96 156 343 2 1500 2100
       RCVD OK
       SENT .
       RCVD SCHD 3 joon 1
       job 3 (waiting) on joon 1 (booting) SCHEDULED
       SENT OK
       RCVD REDY
t 120: job 1 on juju 0 RUNNING
t 137: job 4 -- CPU: 1, Mem: 100, Disk: 2000, Est Runtime: 111 SUBMITTED
       SENT JOBN 4 137 1 100 2000 111
       RCVD GETS Capable 1 100 2000
       SENT DATA 5 124
       RCVD OK
       SENT juju 0 active 120 0 2500 13100 0 1
            juju 1 booting 156 0 2500 13900 1 0
            joon 0 active 97 1 15300 60200 0 1
            joon 1 booting 161 2 15100 61500 1 0
            super-silk 0 inactive -1 16 64000 512000 0 0
       RCVD OK
       SENT .
       RCVD LSTJ juju 1
       SENT DATA 1 59
       RCVD OK
       SENT 2 1 96 156 343 2 1500 2100
       RCVD OK
       SENT .
       RCVD LSTJ joon 1
       SENT DATA 1 59
       RCVD OK
       SENT 3 1 101 161 380 2 900 2500
       RCVD OK
       SENT .
       RCVD SCHD 4 joon 0
       job 4 (running) on joon 0 (active) SCHEDULED
       job 4 on joon 0 RUNNING
       SENT OK
       RCVD REDY
t 156: job 2 on juju 1 RUNNING
       job 5 -- CPU: 3, Mem: 2700, Disk: 2600, Est Runtime: 8 SUBMITTED
       SENT JOBN 5 156 3 2700 2600 8
       RCVD GETS Capable 3 2700 2600
       SENT DATA 3 124
       RCVD OK
       SENT joon 0 active 97 0 15200 58200 0 2
            joon 1 booting 161 2 15100 61500 1 0
            super-silk 0 inactive -1 16 64000 512000 0 0
       RCVD OK
       SENT .
       RCVD LSTJ joon 1
       SENT DATA 1 59
       RCVD OK
       SENT 3 1 101 161 380 2 900 2500
       RCVD OK
       SENT .
       RCVD SCHD 5 super-silk 0
       job 5 (waiting) on super-silk 0 (booting) SCHEDULED
       SENT OK
       RCVD REDY
t 161: job 3 on joon 1 RUNNING
t 198: job 6 -- CPU: 4, Mem: 4000, Disk: 7600, Est Runtime: 1074 SUBMITTED
       SENT JOBN 6 198 4 4000 7600 1074
       RCVD GETS Capable 4 4000 7600
       SENT DATA 3 124
       RCVD OK
       SENT joon 0 active 97 0 15200 58200 0 2
            joon 1 active 161 2 15100 61500 0 1
            super-silk 0 booting 236 13 61300 509400 1 0
       RCVD OK
       SENT .
       RCVD LSTJ super-silk 0
       SENT DATA 1 59
       RCVD OK
       SENT 5 1 156 236 8 3 2700 2600
       RCVD OK
       SENT .
       RCVD SCHD 6 super-silk 0
       job 6 (waiting) on super-silk 0 (booting) SCHEDULED
       SENT OK
       RCVD REDY
t 225: job 7 -- CPU: 2, Mem: 500, Disk: 2100, Est Runtime: 442 SUBMITTED
       SENT JOBN 7 225 2 500 2100 442
       RCVD GETS Capable 2 500 2100
       SENT DATA 5 124
       RCVD OK
       SENT juju 0 active 120 0 2500 13100 0 1
           juju 1 active 156 0 2500 13900 0 1
           joon 0 active 97 0 15200 58200 0 2
           joon 1 active 161 2 15100 61500 0 1
           super-silk 0 booting 236 9 57300 501800 2 0
       RCVD OK
       SENT .
       RCVD LSTJ super-silk 0
       SENT DATA 2 59
       RCVD OK
       SENT 5 1 156 236 8 3 2700 2600
 6 1 198 236 1074 4 4000 7600
       RCVD OK
       SENT .
       RCVD SCHD 7 joon 1
       job 7 (running) on joon 1 (active) SCHEDULED
       job 7 on joon 1 RUNNING
       SENT OK
       RCVD REDY
 t 236: job 5 on super-silk 0 RUNNING
       job 6 on super-silk 0 RUNNING
 t 249: job 8 -- CPU: 1, Mem: 100, Disk: 800, Est Runtime: 926 SUBMITTED
       SENT JOBN 8 249 1 100 800 926
       RCVD GETS Capable 1 100 800
       SENT DATA 5 124
       RCVD OK
       SENT juju 0 active 120 0 2500 13100 0 1
           juju 1 active 156 0 2500 13900 0 1
           joon 0 active 97 0 15200 58200 0 2
           joon 1 active 161 0 14600 59400 0 2
           super-silk 0 active 236 9 57300 501800 0 2
       RCVD OK
       SENT .
       RCVD SCHD 8 super-silk 0
       job 8 (running) on super-silk 0 (active) SCHEDULED
       job 8 on super-silk 0 RUNNING
       SENT OK
       RCVD REDY
 t 257: job 5 on super-silk 0 COMPLETED
       SENT JCPL 257 5 super-silk 0
       RCVD REDY
 t 303: job 4 on joon 0 COMPLETED
       SENT JCPL 303 4 joon 0
       RCVD REDY
 t 308: job 9 -- CPU: 2, Mem: 600, Disk: 1500, Est Runtime: 2010 SUBMITTED
       SENT JOBN 9 308 2 600 1500 2010
       RCVD GETS Capable 2 600 1500
       SENT DATA 5 124
       RCVD OK
       SENT juju 0 active 120 0 2500 13100 0 1
           juju 1 active 156 0 2500 13900 0 1
           joon 0 active 97 1 15300 60200 0 1
           joon 1 active 161 0 14600 59400 0 2
           super-silk 0 active 236 11 59900 503600 0 2
       RCVD OK
       SENT .
       RCVD SCHD 9 super-silk 0
       job 9 (running) on super-silk 0 (active) SCHEDULED
       job 9 on super-silk 0 RUNNING
       SENT OK
       RCVD REDY
 t 575: job 7 on joon 1 COMPLETED
       SENT JCPL 575 7 joon 1
       RCVD REDY
 t 642: job 2 on juju 1 COMPLETED
       SENT JCPL 642 2 juju 1
       RCVD REDY
 t 796: job 6 on super-silk 0 COMPLETED
       SENT JCPL 796 6 super-silk 0
       RCVD REDY
 t 1073: job 8 on super-silk 0 COMPLETED
        SENT JCPL 1073 8 super-silk 0
        RCVD REDY
 t 1215: job 1 on juju 0 COMPLETED
        SENT JCPL 1215 1 juju 0
        RCVD REDY
 t 1232: job 3 on joon 1 COMPLETED
        SENT JCPL 1232 3 joon 1
        RCVD REDY
 t 1337: job 0 on joon 0 COMPLETED
        SENT JCPL 1337 0 joon 0
        RCVD REDY
 t 1778: job 9 on super-silk 0 COMPLETED
        SENT JCPL 1778 9 super-silk 0
        RCVD REDY
        SENT NONE
        RCVD QUIT
        SENT QUIT
 # -------------------------------------------------------------------------------------
 # 2 juju servers used with a utilisation of 100.00 at the cost of $0.09
 # 2 joon servers used with a utilisation of 100.00 at the cost of $0.26
 # 1 super-silk servers used with a utilisation of 100.00 at the cost of $0.34
 # ==================================== [ Summary ] ====================================
 # actual simulation end time: 1778, #jobs: 10 (failed 0 times)
 # total #servers used: 5, avg util: 100.00% (ef. usage: 100.00%), total cost: $0.69
 # avg waiting time: 35, avg exec time: 728, avg turnaround time: 763
```

* The effective usage (ef. usage) is calculated based on only the server usage of successfully completed jobs; that is, the server usage of pre-empted jobs won't be accounted for.

Appendix F: a sample resource failure file

ds-res-failures.txt for sample-config01.xml (when "-f websites02" is used)

```
#base_trace: websites02, config_file: ds-sample-config01.xml, total_servers: 5, total_time: 604800,
time_dist_mean: 2.430000, time_dist_stdev: 0.300000, node_dist_mean: 1.650000, node_dist_stdev: 1.220000
0 143 juju 0
239 515 super-silk 0
264 847 joon 1
1495 1964 juju 0
1559 1978 super-silk 0
1665 1878 joon 1
2465 3830 joon 1
3226 3440 juju 0
4189 4371 joon 1
```

References

[1] "Discrete-event simulation," https://en.wikipedia.org/wiki/Discrete-event_simulation, accessed on 28 Jan 2022.

[2] J. King, Y. K. Kim, **Lee, Young Choon**, and S.-H. Hong, "Visualisation of distributed systems simulation made simple," in *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2019, pp. 309–312.