ds-sim: A Distributed Systems Simulator

(Developed by Young Choon Lee, Young Ki Kim and Jayden King)

User Guide

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 for 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 (Figure 2). 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.

At high level, ds-sim performs a simulation repeating three key steps:

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

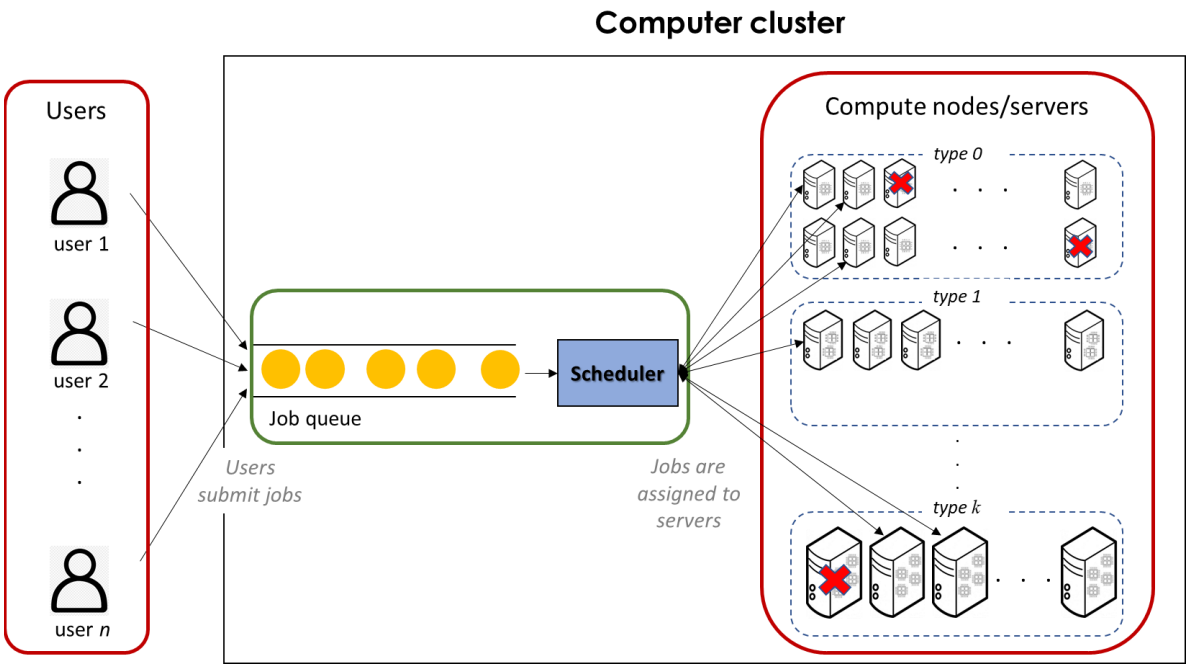


Figure 1. Scheduling in a computer cluster as a typical distributed system.

Initialise system state variables by reading

- Simulation configuration file (e.g., `ds-config01.xml`) in the server-side simulator
- System information (`ds-system.xml`) in the client-side simulator.

Initialise Clock (usually starts at simulation time zero)

While (there is a job to schedule) **do**

- Get a job to schedule
- Set clock to next event time (based on the submission time of the job in our case)
- Schedule the job
- Update statistics

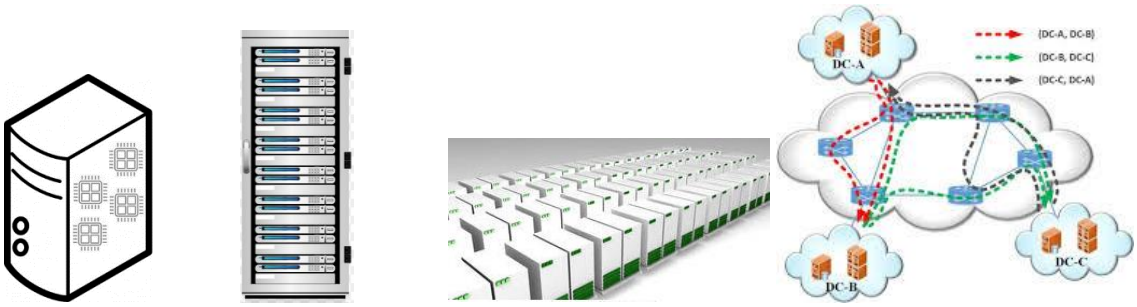
End while

Generate statistical report (e.g., job completion times, system utilisation and costs)

Figure 2. Discrete event simulation logic.

2. Background

Distributed (computing) systems come in various sizes and scale (Figure 3). 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, 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) Single machine with multiple CPUs (b) a rack of servers (c) a data centre (d) a federated cloud

Figure 3. 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.

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.

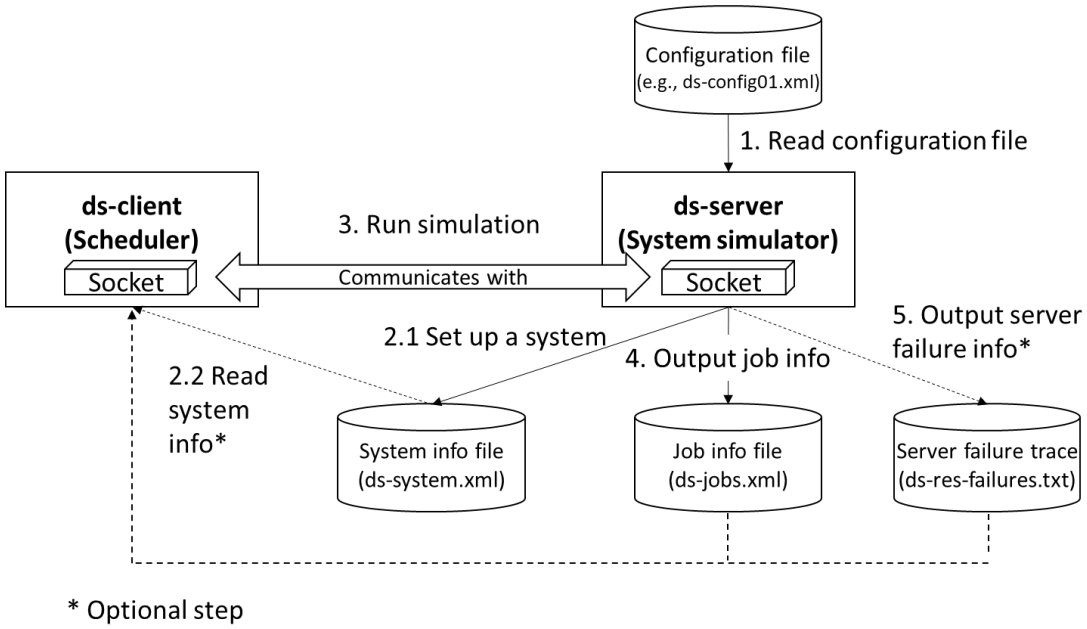


Figure 4. ds-sim architecture.

3. Architecture

ds-sim consists of two major components, ds-server and ds-client (Figure 4). 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-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*** [1]. 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.

A job in ds-sim is attributed by the following.

Description		XML attribute name	Data type	Data range
Identification	Number	id	Integer	0..999999
	Type	type	Char array/string	64; 0..127 *
Timing	Submission time	submitTime	Integer	0.. 604799**
	Estimated run time (sec)	estRunTime	Integer	1..604800***
Resource requirements	CPU cores	core	Integer	1..256
	Memory (MB)	memory	Integer	1..8192000
	Disk (MB)	disk	Integer	1..8192000

* While the length of job type name is limited to 64 characters, the number job types is limited by 128.
** The range of job submission time is between 0 and 604799 seconds (or 7 days – 1 second).
*** The range of job runtime (both estimated and actual runtimes) is between 1 and 604800 seconds (or 7 days).

Job attribute names are used within the job element in a simulation configuration file.

id: a sequence number based on the submission time
type: an identifier of job category based on run time
submitTime: submission time; it can be the time of initial submission or the time of re-submission after pre-emption/failure.
estRunTime: estimated run time (in seconds)
core: the number of CPU cores
memory: the amount of RAM (in MB)
disk: the amount of disk space (in MB)

The actual resource limits 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 256, 8192000 and 8192000. In addition, memory/disk limit per core is 200 (MB).

Job states are as follows.
0: submitted, 1: waiting, 2: running, 3: suspended, 4: completed, 5: pre-empted, 6: failed, 7: killed

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.

A server in ds-sim is attributed by the following.

Description		XML attribute name	Data type	Data range
Identification*	Number	id	Integer	0..999
	Type	type	Char array/string	64; 0..99**
Timing	Max #servers	limit	Integer	1..1000
	Bootup time (s)	bootupTime	Integer	0..600
	Hourly rental rate ($)	hourlyRate	Float	0..1000000
Resource requirements	CPU cores	core	Integer	1..256
	Memory (MB)	memory	Integer	1..8192000
	Disk (MB)	disk	Integer	1..8192000

* Servers are identified by the type and id, e.g., small, 2 and large, 2.
** While the length of server type name is limited to 64 characters, the number server types is limited by 100; hence, the max number of servers in total is 100,000, i.e., 1000 servers * 100 types.

Server attribute names are used within the job element in a simulation configuration file.

id: a sequence number based on the submission time
type: an identifier of job category based on run time
limit: the number of servers of a particular type
bootupTime: the amount of time taken to boot a server of a particular type
hourlyRate: the monetary cost for renting a server of a particular type per hour
core: 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 internally maintains a number of lists, one for each job state, except the 'submitted' state. In general, as soon as a job is submitted and sent to the scheduler (ds-client), it will be scheduled to a particular server and placed in one of those lists, e.g., a waiting list/queue or a running job list, resulting in job state being changed. In particular, each server in ds-sim uses "a local waiting queue". There are three cases ds-sim keeps a job in its global waiting queue: (1) there is no available server that has sufficient resource capacity due primarily to server/resource failures, (2) the scheduler (intentionally) skips the scheduling of that job, and (3) the job is required to be re-submitted (killed/pre-empted/failed).

Server states are as follows.

0: inactive, 1: booting, 2: idle, 3: active, 4: unavailable

* Units used in ds-sim for time, monetary cost and memory/disk are the second, the dollar and Mega Byte (MB), respectively.

5. Configuration

Simulation settings are configured by an XML configuration file (see Appendices C and D) supplied as an argument. There are four main parts of simulation configuration: Server, Job, Workload and Termination condition. The values of these elements and attributes must be within valid ranges as specified in the previous two Sections above. ***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.

Jobs can also be of one or more types determined primarily be their length (runtime, `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, the proportion of jobs of all types. Clearly, the sum of rate values 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 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.

termination defines one or two termination conditions (`endtime` and `jobcount`). A simulation terminates whichever condition meets first. 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 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.

Type names, except that in `termination` are all user-defined names.

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.

Simulation protocol

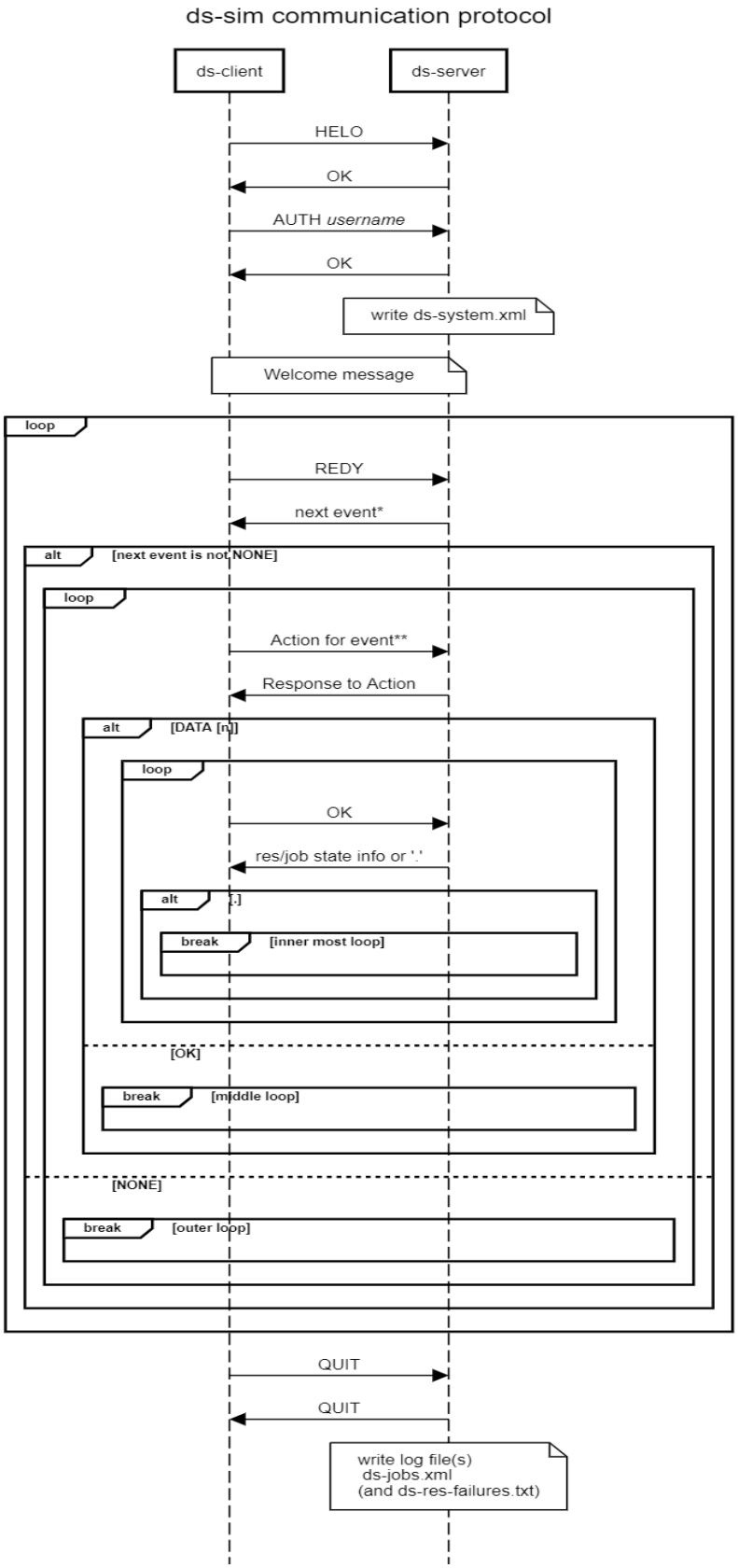


Figure 5. ds-sim simulation protocol.

* Simulation Event (from ds-server), e.g., JOBN, JOBP, JCPL, RESF, RESR and NONE.
** Action as response to Event, e.g., GETS, SCHD, CNTJ, EJWT, LSTJ, PSHJ, MIGJ, KILJ, TERM and REDY. Note that sending REDY implies the response to the latest simulation event is ignored (JCPL, RESF and RESR) or postponed (JOBN and JOBP).

Detailed steps of the protocol

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)
    - JCPL: information on the latest job completion
    - RESF: information on the latest server failed
    - RESR: information on the latest server recoverd
    - NONE: when 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
    - 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
    - PSHJ: the request to get the next job skipping the current job without scheduling
    - 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: 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, PSHJ, MIGJ and KILJ): action is successfully done
    - *n* for CNTJ: the number of jobs
    - *time* for EJWT: the sum of estimated waiting time on a specified server
    - DATA for LSTJ: the indicator for the requested information will be sent in the successive messages
    - "*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, or
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. In the current implementation of ds-sim, it can be any character string. As no actual authentication takes place, it is more or less for the purpose of 'handshaking'.

6. Simulation 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 configuration files.

-b *n* (job ID)
    b(reak) simulation at job *n*; the simulation can run in an interactive mode still with ds-client

-c configuration file name (.xml)
    c(onfiguration file): use of configuration file

-d *n* (in seconds)
    d(uration): simulation duration/end time

-f teragrid|nd07cpu|websits02|g5k06|pl05|ldns04
    f(ailure): failure distribution model (e.g., teragrid, nd07cpu and websites02)

-g *n* (in second)
    g(ranularity): time granularity for resource failures

-h all|usage|limits|states
    h(elp): usage

-i
    i(nteractive): the simulation can run in a *fully* interactive mode without ds-client

-j *n* (max #jobs to generate)
    j(ob): set job count

-l *n*

l(imit of #servers): the number of servers (uniform to all server types)

-n

n(ewline): newline character (\n) at the end of each message
This option is helpful for schedulers written in Java (or other languages with newline terminators)

-o

o(mit): omit to send server failure event messages (RESF and RESR) for fast simulation; it is effective only if failures are simulated (i.e., the use of '-f')

-p *n*

p(ort number): 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)

-r *n* (some integer)

r(andom seed): random seed

-s *n* (in percentage)

s(cale factor for resource failures): between 1 and 100;
1 for nearly no failures and 100 for original

-v all|brief|stats[+]

v(erbose): verbose

* These command line argument options are also shown if "-h usage" is used.

7. Commands

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.

EXAMPLE

OK

AUTH – authenticate user

SYNOPSIS

AUTH USERNAME

DESCRIPTION

The AUTH command “authenticates” the user specified by the 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 submitTime jobID estRuntime core memory disk

DESCRIPTION

The JOBN command sends a *normal* job that is submitted for the first time. All fields are integer values. jobID starts from 0. The estRuntime is only an estimate. 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.

EXAMPLE

JOBN 172 4 320 2 50 120

JOBP – resend a job

SYNOPSIS

JOBP submitTime jobID estRuntime core memory disk

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. The estRuntime is only an estimate. 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.

EXAMPLE

JOBP 2142 12 750 4 90 350

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 MQ_Cloud_large 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.

EXAMPLE

RESR MQ_Cloud_large 122 21159

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 101 3 380 2 900 2500	// 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
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
silk-the-invincible 0 inactive -1 16 64000 512000 0 0
	// ds-server; the entire server state information
	// is sent as **a single message** delimited by
	// a newline character ('\n').

OK	// ds-client
.	// ds-server

The state information on each server is formatted in
serverType serverID state curStartTime core mem 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, respectively above. 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 silk-the-invincible). 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 and 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

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 seven state codes, 1 - 7 except 0 of submitted.

EXAMPLE

```
CNTJ joon 0 2                    // the number of running jobs on joon 0

1                                // the response from ds-server
```

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 MQ_Cloud_large 2

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 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                    // ds-client

DATA   1 52                      // ds-server
```

OK // ds-client

2 2 1208 172 2 100 200 // ds-server

OK // ds-client

7 2 1224 328 1 120 450 // ds-server

OK // ds-client

11 1 1259 49 4 380 1000 // ds-server

OK // ds-client

. // ds-server

PSHJ – Push the current job to back of the next job

SYNOPSIS

PSHJ

DESCRIPTION

The PSHJ command pushes the current job to back of the next job. In particular, it forces to skip the scheduling of the current job and to get the next job. The current job's submission time is then set to one time unit (one second) larger than that of next job. This command can be used primarily when all servers capable of running the current job are unavailable, e.g., failed.

EXAMPLE

PSHJ

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.

EXAMPLE

MIGJ 572 small 2 medium 4

OK // the response from ds-server

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 MQ_Cloud_large 0 1208

OK // the response from ds-server

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 very_very_very_very_very_very_large 3

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'). In the meantime, the response to LSTJ* is a series of individual messages; hence nRecs is always 1. Each message should be responded with OK by ds-client, except '.', the last message. 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.

** In the future, the format of LSTJ's response may change to the same format as that of GETS; this reduces communication overhead significantly.*

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

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: simulator manual

* The following can be also obtained by running ds-server with -h all, i.e., `ds-server -h all`.

Basic usage

For the (ds-sim) server side,
`ds-server` [*OPTION*] . . .

For the (ds-sim) client side,
`ds-client` [*OPTION*] . . .

Command line options for ds-server
Please see Section 6 Simulation Options.

Command line options for ds-client

`-a` scheduling algorithm name (this is only a mandatory argument to be implemented)

Appendix B: Command list

Category	Cmd	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 (failed, due to server failure/termination, or killed)
	JCPL	Job completion
	RESF	Server failure notice
	RESR	Server recovery notice
	NONE	No more jobs to schedule
Client Action	GETS	Resource information request: - GETS All: the information of all servers, in the system, regardless of their state. - GETS Type: the information of all servers of a particular server type. - GETS Avail: the information of servers that are available (i.e., inactive, booting and idle) for the job with sufficient resources. - GETS Capable: the information of servers that are capable of running the job, based on initial resource capacity, not current resource availability.
	SCHD	Scheduling decision
	CNTJ	The number of jobs on a specified server with a particular state
	EJWT	The sum of estimated waiting time on a given server; note that the calculation shouldn't be considered to be accurate because (1) it's based on estimated runtimes of waiting jobs and (2) more importantly, it doesn't consider the possibility of parallel execution of waiting jobs
	LSTJ	Job list of a server, i.e., all pending jobs (waiting and running jobs)
	PSHJ	Force to get the next job to schedule skipping the current job; this command can be used primarily when all servers capable of running the current job are failed; When sent, the current job is pushed back to the job queue by making its submission time 1 time unit later than the next job
	MIGJ	
	KILJ	Kill a job; the killed job is pushed back to the queue with the killed time as a new submission time

	TERM	Server termination (all waiting/running jobs are killed and re-submitted for scheduling); the server will be put into the Inactive state.
Error	ERR	Invalid message received
Acknowledgement	OK	Response to a valid command

* Commands in **blue** are used by client, those in **green** by server and those in **orange** by both.

Appendix C: XML Schema for ds-sim configuration files

```
<?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="coreCount" 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 D: sample configuration files

ds-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" coreCount="2" memory="4000" disk="16000" />
      <server type="joon" limit="2" bootupTime="60" hourlyRate="0.4" coreCount="4" memory="16000" disk="64000" />
      <server type="silk-the-invincible" limit="1" bootupTime="80" hourlyRate="0.8" coreCount="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>
```

ds-config02.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by: Y. C. Lee -->
<config randomSeed="1024">
  <servers>
      <server type="tiny" limit="10" bootupTime="60" hourlyRate="0.1" coreCount="1" memory="2000" disk="16000" />
      <server type="small" limit="10" bootupTime="60" hourlyRate="0.2" coreCount="2" memory="4000" disk="32000" />
      <server type="medium" limit="10" bootupTime="60" hourlyRate="0.4" coreCount="4" memory="8000" disk="64000" />
      <server type="large" limit="10" bootupTime="60" hourlyRate="0.8" coreCount="8" memory="16000" disk="128000" />
      <server type="xlarge" limit="10" bootupTime="60" hourlyRate="1.6" coreCount="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="40" maxLoad="60" />
  <termination>
      <condition type="endtime" value="86400" />
      <condition type="jobcount" value="500" />
  </termination>
</config>
```

ds-config03.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by: Y. C. Lee -->
<config randomSeed="1024">
  <servers>
      <server type="small" limit="20" bootupTime="60" hourlyRate="0.2" coreCount="2" memory="4000" disk="16000" />
      <server type="medium" limit="20" bootupTime="60" hourlyRate="0.4" coreCount="4" memory="16000" disk="64000" />
      <server type="large" limit="10" bootupTime="80" hourlyRate="0.8" coreCount="16" memory="64000" disk="512000" />
      <server type="MQ_Cloud_large" limit="30" bootupTime="70" hourlyRate="0.6" coreCount="8" memory="32000" disk="256000" />
      <server type="very_very_very_very_very_very_large" limit="10" bootupTime="120" hourlyRate="3.2" coreCount="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>
```

ds-config04.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by: Y. C. Lee -->
<config randomSeed=" 202008">
  <servers>
      <server type="tiny" limit="100" bootupTime="60" hourlyRate="0.1" coreCount="1" memory="2000" disk="8000" />
      <server type="small" limit="100" bootupTime="60" hourlyRate="0.2" coreCount="2" memory="4000" disk="16000" />
      <server type="medium" limit="100" bootupTime="60" hourlyRate="0.4" coreCount="4" memory="8000" disk="32000" />
      <server type="16xlarge" limit="10" bootupTime="120" hourlyRate="25.6" coreCount="256" memory="512000" disk="4096000" />
      <server type="large" limit="40" bootupTime="80" hourlyRate="0.8" coreCount="8" memory="16000" disk="64000" />
      <server type="xlarge" limit="40" bootupTime="80" hourlyRate="1.6" coreCount="16" memory="32000" disk="128000" />
      <server type="2xlarge" limit="20" bootupTime="100" hourlyRate="3.2" coreCount="32" memory="64000" disk="256000" />
      <server type="4xlarge" limit="20" bootupTime="100" hourlyRate="6.4" coreCount="64" memory="128000" disk="512000" />
      <server type="8xlarge" limit="20" bootupTime="120" hourlyRate="12.8" coreCount="128" memory="256000" disk="1024000" />
  </servers>
  <jobs file="normalworkload.xml"/>
</config>
```

ds-config05.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by: Y. C. Lee -->
<config newline="true" randomSeed="12345">
  <servers failureFile="infreqeuntfailures.txt">
      <server type="tiny" limit="100" bootupTime="60" hourlyRate="0.1" coreCount="1" memory="2000" disk="8000" />
      <server type="small" limit="100" bootupTime="60" hourlyRate="0.2" coreCount="2" memory="4000" disk="16000" />
      <server type="medium" limit="100" bootupTime="60" hourlyRate="0.4" coreCount="4" memory="8000" disk="32000" />
      <server type="16xlarge" limit="10" bootupTime="120" hourlyRate="25.6" coreCount="256" memory="512000" disk="4096000" />
      <server type="large" limit="40" bootupTime="80" hourlyRate="0.8" coreCount="8" memory="16000" disk="64000" />
      <server type="xlarge" limit="40" bootupTime="80" hourlyRate="1.6" coreCount="16" memory="32000" disk="128000" />
      <server type="2xlarge" limit="20" bootupTime="100" hourlyRate="3.2" coreCount="32" memory="64000" disk="256000" />
      <server type="4xlarge" limit="20" bootupTime="100" hourlyRate="6.4" coreCount="64" memory="128000" disk="512000" />
      <server type="8xlarge" limit="20" bootupTime="120" hourlyRate="12.8" coreCount="128" memory="256000" disk="1024000" />
```

```
  </servers>
  <jobs file="normalworkload.xml"/>
</config>
```

** Note that infreqeuntfailures.txt should have been produced with the same configuration, but without the "failureFile" attribute. In other words, the use of failures file is to reproduce results.*

Appendix E: a sample system information file

ds-system.xml for ds-config01.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- system information generated for ds-sim@MQ, 18-August, 2020 @ MQ - client-server -->
<system>
      <servers>
            <server type="juju" limit="2" bootupTime="60" hourlyRate="0.2" coreCount="2" memory="4000" disk="16000" />
            <server type="joon" limit="2" bootupTime="60" hourlyRate="0.4" coreCount="4" memory="16000" disk="64000" />
            <server type="silk-the-invincible" limit="1" bootupTime="80" hourlyRate="0.8" coreCount="16" memory="64000"
disk="512000" />
      </servers>
</system>
```

Appendix F: a sample jobs file

ds-jobs.xml for ds-config01.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- job list for ds-sim, 18-August, 2020 @ 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>
```

Appendix G: a sample simulation log (from `./ds-server -c ds-config01.xml -v all` and `./ds-client -a bf` with the best-fit (BF) scheduling algorithm.

```
# ds-sim server 18-August, 2020 @ MQ - client-server
# Server-side simulator started with './ds-server -c ds-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
SENT JOBN 37 0 653 3 700 3800
RCVD GETS Capable 3 700 3800
SENT DATA 3
RCVD OK
SENT joon 0 inactive -1 4 16000 64000 0 0
joon 1 inactive -1 4 16000 64000 0 0
silk-the-invincible 0 inactive -1 16 64000 512000 0 0
RCVD OK
SENT .
RCVD SCHD 0 joon 0
t:         37 job     0 (waiting) on # 0 of server joon (booting) SCHEDULED
SENT OK
RCVD REDY
SENT JOBN 60 1 2025 2 1500 2900
RCVD GETS Capable 2 1500 2900
SENT DATA 5
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
silk-the-invincible 0 inactive -1 16 64000 512000 0 0
RCVD OK
SENT .
RCVD SCHD 1 juju 0
t:         60 job     1 (waiting) on # 0 of server juju (booting) SCHEDULED
SENT OK
RCVD REDY
SENT JOBN 96 2 343 2 1500 2100
RCVD GETS Capable 2 1500 2100
SENT DATA 5
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
silk-the-invincible 0 inactive -1 16 64000 512000 0 0
RCVD OK
SENT .
RCVD SCHD 2 juju 1
t:         96 job     2 (waiting) on # 1 of server juju (booting) SCHEDULED
SENT OK
RCVD REDY
t:         97 job     0 on # 0 of server joon RUNNING
SENT JOBN 101 3 380 2 900 2500
RCVD GETS Capable 2 900 2500
SENT DATA 5
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
silk-the-invincible 0 inactive -1 16 64000 512000 0 0
RCVD OK
SENT .
RCVD SCHD 3 joon 1
t:        101 job     3 (waiting) on # 1 of server joon (booting) SCHEDULED
SENT OK
RCVD REDY
t:        120 job     1 on # 0 of server juju RUNNING
SENT JOBN 137 4 111 1 100 2000
RCVD GETS Capable 1 100 2000
SENT DATA 5
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
silk-the-invincible 0 inactive -1 16 64000 512000 0 0
RCVD OK
SENT .
RCVD SCHD 4 joon 0
t:        137 job     4 (running) on # 0 of server joon (active) SCHEDULED
t:        137 job     4 on # 0 of server joon RUNNING
SENT OK
RCVD REDY
t:        156 job     2 on # 1 of server juju RUNNING
SENT JOBN 156 5 8 3 2700 2600
RCVD GETS Capable 3 2700 2600
SENT DATA 3
RCVD OK
SENT joon 0 active 97 0 15200 58200 0 2
joon 1 booting 161 2 15100 61500 1 0
silk-the-invincible 0 inactive -1 16 64000 512000 0 0
RCVD OK
SENT .
RCVD SCHD 5 silk-the-invincible 0
t:        156 job     5 (waiting) on # 0 of server silk-the-invincible (booting)
SCHEDULED
SENT OK
RCVD REDY
t:        161 job     3 on # 1 of server joon RUNNING
SENT JOBN 198 6 1074 4 4000 7600
RCVD GETS Capable 4 4000 7600
SENT DATA 3
RCVD OK
SENT joon 0 active 97 0 15200 58200 0 2
joon 1 active 161 2 15100 61500 0 1
silk-the-invincible 0 booting 236 13 61300 509400 1 0
RCVD OK
SENT .
RCVD SCHD 6 joon 0
t:        198 job     6 (waiting) on # 0 of server joon (active) SCHEDULED
SENT OK
RCVD REDY
SENT JOBN 225 7 442 2 500 2100
RCVD GETS Capable 2 500 2100
SENT DATA 5
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 1 2
joon 1 active 161 2 15100 61500 0 1
silk-the-invincible 0 booting 236 13 61300 509400 1 0
RCVD OK
SENT .
```

```
RCVD SCHD 7 joon 1
t:        225 job     7 (running) on # 1 of server joon (active) SCHEDULED
t:        225 job     7 on # 1 of server joon RUNNING
SENT OK
RCVD REDY
t:        236 job     5 on # 0 of server silk-the-invincible RUNNING
SENT JOBN 249 8 926 1 100 800
RCVD GETS Capable 1 100 800
SENT DATA 5
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 1 2
joon 1 active 161 0 14600 59400 0 2
silk-the-invincible 0 active 236 13 61300 509400 0 1
RCVD OK
SENT .
RCVD SCHD 8 silk-the-invincible 0
t:        249 job     8 (running) on # 0 of server silk-the-invincible (active) SCHEDULED
t:        249 job     8 on # 0 of server silk-the-invincible RUNNING
SENT OK
RCVD REDY
t:        257 job     5 on # 0 of server silk-the-invincible COMPLETED
SENT JCPL 257 5 silk-the-invincible 0
RCVD REDY
t:        303 job     4 on # 0 of server joon COMPLETED
SENT JCPL 303 4 joon 0
RCVD REDY
SENT JOBN 308 9 2010 2 600 1500
RCVD GETS Capable 2 600 1500
SENT DATA 5
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 1 1
joon 1 active 161 0 14600 59400 0 2
silk-the-invincible 0 active 236 15 63900 511200 0 1
RCVD OK
SENT .
RCVD SCHD 9 silk-the-invincible 0
t:        308 job     9 (running) on # 0 of server silk-the-invincible (active) SCHEDULED
t:        308 job     9 on # 0 of server silk-the-invincible RUNNING
SENT OK
RCVD REDY
t:        575 job     7 on # 1 of server joon COMPLETED
SENT JCPL 575 7 joon 1
RCVD REDY
t:        642 job     2 on # 1 of server juju COMPLETED
SENT JCPL 642 2 juju 1
RCVD REDY
t:       1073 job     8 on # 0 of server silk-the-invincible COMPLETED
SENT JCPL 1073 8 silk-the-invincible 0
RCVD REDY
t:       1215 job     1 on # 0 of server juju COMPLETED
SENT JCPL 1215 1 juju 0
RCVD REDY
t:       1232 job     3 on # 1 of server joon COMPLETED
SENT JCPL 1232 3 joon 1
RCVD REDY
t:       1337 job     0 on # 0 of server joon COMPLETED
t:       1337 job     6 on # 0 of server joon RUNNING
SENT JCPL 1337 0 joon 0
RCVD REDY
t:       1778 job     9 on # 0 of server silk-the-invincible COMPLETED
SENT JCPL 1778 9 silk-the-invincible 0
```

```
RCVD REDY
t:       1897 job     6 on # 0 of server joon COMPLETED
SENT JCPL 1897 6 joon 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 $0.32
# 1 silk-the-invincible servers used with a utilisation of 100.00 at the cost of $0.34
# ================================== [ Summary ] ================================
# actual simulation end time: 1897, #jobs: 10 (failed 0 times)
# total #servers used: 5, avg util: 100.00% (ef. usage: 100.00%), total cost: $0.75
# avg waiting time: 145, avg exec time: 728, avg turnaround time: 873
```

Appendix H: a sample resource failure file

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

```
#base_trace: websites02, config_file: ds-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 silk-the-invincible 0
264 847 joon 1
1495 1964 juju 0
1559 1978 silk-the-invincible 0
1665 1878 joon 1
2465 3830 joon 1
3226 3440 juju 0
4189 4371 joon 1
```

References

[1] Jayden King, Young Ki Kim, Young Choon Lee, Seok-Hee Hong, Visualisation of Distributed Systems Simulation Made Simple, In Proc. CloudCom, 309-312, 2019.
[2] Y. C. Lee, J. King and S.H. Hong, Holistic Approach for Studying Resource Failures at Scale, in Proc. The IEEE International Symposium on Network Computing and Applications (NCA 2019), 2019.