Xerxes: Distributed Load Generator for Cloud-scale Experimentation

Version 1, June 2013

Author: Mukil Kesavan
Contact: mukilk@gatech.edu

Center for Experimental Research in Computer Systems (CERCS)
College of Computing, Georgia Institute of Technology

Table of Contents

Introduction

Xerxes [1] is a distributed load generation framework that can generate desired CPU and memory consumption patterns across a large number of machines. It is organized as a collection of individual load generators, one per machine, coordinated via a master node. It can be used to replay traces from datacenter machines, generate loads fitting statistical distributions and generate resource usage spikes, all at varying scales. The accuracy of aggregate resource consumption patterns decays gradually in line with the rate of experimentation node failures. This lets datacenter/cluster administrators and researchers achieve highly scalable load testing without having to deal with application logic specific nuances.

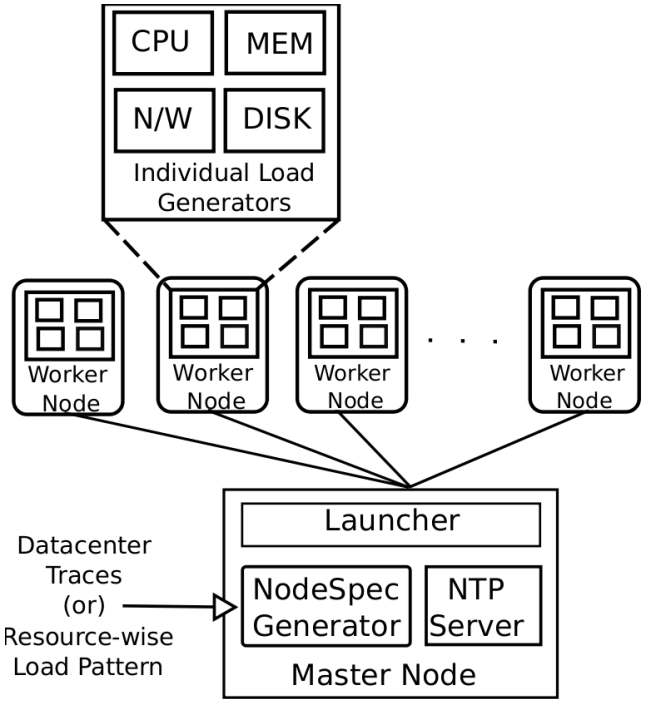

Figure 1. Xerxes Architecture

Architecture and Overview

Xerxes consists of a single **master** node and multiple **worker** nodes - one per server under evaluation as shown in Figure 1. The master node takes overall load generation specifications, converts them to individual worker node specifications that when executed together produces the global, large scale aggregate load pattern desired. The individual node specifications are generated by the **NodeSpec Generator** Module shown in the figure, and their execution is orchestrated by the **Launcher** Module through the Linux cron facility. The load specifications to

the master can be either in the form of (a) real-life datacenter traces specifying resource usages at various timestamps or (b) statistical distributions, such as normal distribution with a specified mean and deviation values, for example. In addition, these specifications can be at a per-server level or per-logical-job level, that maps to multiple servers, to generate a global resource usage pattern. The base specifications are extrapolated (in the current implementation, only proportionally) when the number of target servers is greater than the number of specification objects in the benchmark input, or, combined via aggregation in the opposite case.

Further, it is also possible to add usage volume spikes to the base load specifications by specifying the spike parameters as characterized by Bodik et. al. [2]: time-spike-start, time-peak-start, time-peak-end, time-spike-end, spike-magnitude-multiplier. In order to orchestrate a global resource usage pattern, the master is expected to run an **NTP server** that the worker nodes synchronize with periodically (typically once in days on modern machines), so that their individual timeofday values are not hugely divergent. However, once the simulation starts, the worker nodes need no further coordination with the master, or amongst each other, and use local high precision timers to transition between multiple timed load phases. The worker nodes are composed of **individual load generators**: a combined one for CPU and memory[1]. The worker node gets a load specification file for each resource at the start of the simulation from the master, which it then executes in isolation until completion.

Installation

1. Worker nodes need to be installed with the following items from the benchmark tarball:

<xerxes-root>/loadgen/wileE
<xerxes-root>/master/launcher/setupcron.sh

Please ensure that both of these files are in the same location in each worker node. To generate the “wileE” binary, just navigate to <xerxes-root>/loadgen/ and hit “make”. You may use the <xerxes-root>/master/launcher/shipfiles.py script to copy files/folders to worker nodes in parallel. For specifics on how to use shipfiles.py please refer to the next section.

Dependencies:

- GNU C Library (glibc)
- GNU C Compiler (gcc)

2. Master node needs the entire components of the xerxes tarball on a local directory.

[1] The Network and Storage load generators, proposed as part of the original work, are not available in this version.

Dependencies:

- Python
- NumPy for Python

Basic Workflow

1. Gather datacenter traces or statistical distribution parameters.
2. Edit the master's NodeSpec module load specification file to request specific load patterns and parameters.
3. Run the NodeSpec generator python script to generate individual worker node load specs.
4. Ship each worker's load spec to the worker and place it in the same folder as the load generator binary (wileE).
5. Configure the launch parameters (i.e. start time, end time etc.) and launch load generation.

Components & Configuration

Worker Node: CPU and Memory Load Generator - wileE

This is a CPU load generator and a memory allocator benchmark for Linux environments. It can be used standalone on a machine, as well as in conjunction with Xerxes. Over user specified time intervals, it generates (a) desired percentage of CPU usage by alternatively performing numerical computations over an integer array and sleeping, at microsecond granularity and, (b) allocates a desired fraction of system memory and "touches" it once. The current version does not include memory dirtying at a particular rate. If there is interest in this option, please email the author above to request the feature.

Syntax:

./wileE -l <minimum-loop-length> -f <load-spec-file> -t <num-threads> [--recalibrate]

Options:

Parameter	Name	Description
-l	Loop length	Specifies the smallest loop length in microseconds in which to achieve desired cpu utilization.
-f	Load spec file	Format of file entries: <time-in-secs>\t<cpu-load-percent>\t<system-memory-alloc-percent> WileE generates cpu load of specified value for specified amount of time. It also mallocs specified fraction of system memory and "touches" it so its

		all allocated (in virtualized systems for e.g.). You can specify multiple such lines and the benchmark just goes on simulating one case after the other.
-t	Num. threads	Number of parallel (identical) load generation threads to run. Useful for multi-core machines.
--recalibrate (optional)	Recalibration	CPU calibration computes the number of integer computation loops required to obtain 100% CPU usage and uses this to calibrate number of loops required for every other case. By default, this is performed once at the start of the program. Pass this option to recalibrate periodically. This is useful in virtualized environments where the amount of CPU available to a virtual machine varies with consolidation ratios.

If the load generator is intended to be used with Xerxes, one can configure these parameters above via Xerxes launcher configuration files as discussed below.

Master Node: NodeSpec Generator

As stated previously, the NodeSpec Generator module (<xerxes-root>/master/nodespec/generator.py) takes in global load specification parameters and converts it to individual worker node load specifications. It can either (a) convert cpu and memory usage traces from datacenter machines or (b) generate worker node specifications from statistical workload distributions. A global load specification file, <xerxes-root>/master/nodespec/loadspec.conf, can be customized for your case. If using statistical load generation, the NodeSpec module expects another separate input file with the statistical parameters such as mean and standard deviation for normal distribution, for example. Each line in this file is treated as a separate workload and the set of all workloads specified in the file is replayed through extrapolation or aggregation onto the set of target nodes, as required.

The global load specification file has the following options:

Section: “common”

Parameter	Description
TYPE	Type of global load specification. Options: trace, statistical. *trace* = use if you want to replay resource usage traces. *statistical* = use if you want load generation to follow common statistical distributions.

Section: “trace”

Parameter	Description
DATA_FILE	Path to the input trace file in csv format.
SEPARATOR	Separator used in input trace file, for each line. Options: comma, whitespace.
TIMESTAMP_COL	Column that identifies timestamps of samples in trace file. Numerical (0 - n).

ITEM_IDENTIFIER_COL	Column in trace that identifies machine or job from which utilization data has been collected. Numerical (0 - n).
CPU_COL	Column in trace that contains CPU util value. Numerical (0 - n).
CPU_UTIL_FORMAT	What format is utilization information specified? Options: absolute, relative. If “relative”, then the eventual worker node utilizations generated will be normalized to maximum observed value in the trace to get a [0 - 100] range. If the utilization value is already in percentage, then using “absolute” does not scaling except when aggregating, back to [0-100] range for each worker node.
MEM_COL	Column in trace that contains Mem util value. Numerical (0 - n).
MEM_UTIL_FORMAT	Same as for CPU. Note that if the memory utilizations are in MB or KB, use “relative” to generate percentage memory allocation specifications for worker nodes, that may have different amount of total memory than the trace machines.
MEM_LIMIT_PCT	Max amount of memory that the per-node load generator can touch in percentage. It is often desired that this be < 100% to leave room for OS and other apps. Numerical (0 - 100).

Section: “statistical”

Parameter	Description
DISTRIBUTION_CPU	Type of statistical distribution to be used for CPU load generation. Options: uniform, gaussian.
DISTRIBUTION_MEM	Type of statistical distribution to be used for Mem load generation. Options: uniform, gaussian.
WORKLOAD_DESC_FILE_CPU	Path to the separate statistical workload specification file. The information contained in this file is specific to the type of distribution set for the parameters above. Each line in the file is treated as a separate workload. Each line has to have the following information: Normal: Mean , Standard Deviation Uniform: Min, Max The utilizations are expected to be in 0 - 100 range.
SEPARATOR	Separator used in description file above, for each line. Options: comma, whitespace.
UNIFORM_MIN_COL	For “uniform” distribution, the column in the desc file containing the “min” value. Numerical (0 - n).
UNIFORM_MAX_COL	For “uniform” distribution, the column in the desc file containing the “max” value. Numerical (0 - n).
GAUSSIAN_MEAN_COL	For “gaussian” distribution, the column in the desc file containing the “mean” value. Numerical (0 - n).
GAUSSIAN_STDEV_COL	For “gaussian” distribution, the column in the desc file containing the “standard deviation” value. Numerical (0 - n).

WORKLOAD_DURATION_MINS	Total duration in minutes of the generated statistical workload in each worker node. Numerical (0 - n).
MEM_LIMIT_PCT	Max amount of memory that the per-node load generator can touch in percentage. It is often desired that this be < 100% to leave room for OS and other apps. Numerical (0 - 100).

Section: “target”

Parameter	Description
NODES	Number of worker nodes that will be used in experiments. Numerical (0 - n).
TIMEPOINT_DURATION_SECS	How long, in seconds, should each load value (cpu and memory) be generated in the resulting worker node specification. This is irrespective of the timestamps in the trace file (if chosen). In the case of statistical distribution, each randomly chosen utilization value fitting the desired distribution is simulated for this time.
OUTPUT_DIRECTORY	Path to the directory where worker node load specifications should be written to. Each generated worker node load spec will be of the format node<number>.dat.

Section: “spike”

Parameter	Description
SPIKES_DESC_FILE	Path to a separate file describing resource usage spike parameters. Each line in the file is treated as a separate spike in usage happening on a specific subset of total nodes for a specific duration during experiment. Each line has to have the following information: start node, end node, which-resource(cpu/mem), time-spike-start, time-peak-start, time-peak-end, time-spike-end, spike-magnitude-multiplier.
SEPARATOR	Separator used in description file above, for each line. Options: comma, whitespace.
START_NODE_COL	Start and end nodes must be between 0 and NODES value above (non-inclusive) in “target” section. The spike in resource usage is assumed to happen within nodes in this range. Numerical (0 - n).
END_NODE_COL	Same as above.
SPIKE_RESOURCE_COL	Column in the desc file containing which resource utilization is to be spiked. Numerical (0 - n). Valid options in file are “cpu” or “mem”.
TIME_SPIKE_START_COL	Column in desc file specifying what time point during load generation, the overall spike in resource usage has to start in worker nodes. Numerical (0 - n). In the file, specify time in seconds from start. Values will be rounded off to nearest

	timepoint_duration_secs.
TIME_PEAK_START_COL	Column in desc file specifying what time point during load generation, the spike peak resource usage has to start in worker nodes. Numerical (0 - n). In the file, specify time in seconds from start. Values will be rounded off to nearest timepoint_duration_secs.
TIME_PEAK_END_COLUMN	Column in desc file specifying what time point during load generation, the spike peak resource usage has to end in worker nodes. Numerical (0 - n). In the file, specify time in seconds from start. Values will be rounded off to nearest timepoint_duration_secs.
TIME_SPIKE_END_COL	Column in desc file specifying what time point during load generation, the overall spike in resource usage has to end in worker nodes. Numerical (0 - n). In the file, specify time in seconds from start. Values will be rounded off to nearest timepoint_duration_secs.
SPIKE_MAGNITUDE_MULTIPLIER_COL	Column in desc file specifying what the base utilization values observed for each worker node, has to be multiplied with to achieve the spike. Numerical (0 - n). In the file, specify as a number (0 - n). Note that the spikes will be clipped at 100% usage for CPU and MEM_LIMIT_PCT for memory i.e. using a very large multiplier that sends spike utilization beyond max values won't happen.

Master Node: Launcher

The Launcher module contains scripts to ship files/folders to worker nodes and also launch load generation on worker nodes.

Shipping Files/Folders

<xerxes-root>/master/launcher/shipfiles.py

This script ships (using scp) files and folders in parallel to nodes specified in a *hostfile*, to the desired location. It can also be used to ship load specification files to worker nodes. If used in this mode, it will use the logical node number information from the hostfile (format of hostfile discussed below) to ship to each node its load spec file. Note that the load spec file destination path in the remote node has to be the same location where the setupcron.sh script and the wileE load generation benchmark were installed.

The parameters controlling the shipfiles script can be configured using the template in <xerxes-root>/master/launcher/shipspec.conf. It has the following options:

Section: "common"

Parameter	Description
HOSTFILE	Path to file containing worker node DNS name/IPs, one per line. For each worker a logical node number is also expected to be specified in the file along with the DNS name/IP, in the same line.
SEPARATOR	Separator used in hostfile above, for each line. Options: comma, whitespace.
HOSTFILE_NODENUM_COL	Column number in the hostfile that specifies the logical node number of each worker or host. Numerical (0-n).
HOSTFILE_HOST_COL	Column in the hostfile that specifies the host's or worker's DNS name or IP. Numerical (0-n).
NUM_THREADS	Number of threads to use while shipping files/folders to workers. Numerical (1 - n).
LOCAL_SHIP_DATA_PATH	Path to the file/folder that is to be shipped to the workers.
REMOTE_SHIP_DATA_PATH	Destination path in each worker where the file/folder above needs to be shipped to.
SHIPPING_LOAD_SPECS	Options: yes/no. If set to "yes" then the LOCAL_SHIP_DATA_PATH parameter is expected to be the folder containing all the worker load specs created by the NodeSpec module. The script then ships to each file only its own load spec based on its logical number specified in hostfile. If set to "no" it is a standard "scp" of file/folder.
CONNECT_TIMEOUT_SECS	Timeout to use with "scp". If some worker nodes don't respond, then this will ensure that the script doesn't "hang" for a long time. Numerical (0-n) seconds.
REMOTE_USERNAME	The username to use to login to the workers. This should be identical for each worker and passwordless ssh should have been setup apriori between the machine shipping data and the workers.

Launching Load Generation

<xerxes-root>/master/launcher/launch.py

This script ssh's into worker nodes and uses the setupcron.sh script to setup local load generation at a specific time. All nodes will start at approximately the same time as long as NTP has been setup to sync with the master (out-of-band). It also gives you the ability to stop each node's load generation prematurely, at a specific time, if you don't want it to work through each line in the load spec file till the end.

The parameters controlling the load generation launch script can be configured using the template in <xerxes-root>/master/launcher/launchspec.conf. It has the following options:

Section: "common"

Parameter	Description

HOSTFILE	Path to file containing worker node DNS name/IPs, one per line. For each worker a logical node number is also expected to be specified in the file along with the DNS name/IP, in the same line.
SEPARATOR	Separator used in hostfile above, for each line. Options: comma, whitespace.
HOSTFILE_NODENUM_COL	Column number in the hostfile that specifies the logical node number of each worker or host. Numerical (0-n).
HOSTFILE_HOST_COL	Column in the hostfile that specifies the host's or worker's DNS name or IP. Numerical (0-n).
NUM_THREADS	Number of threads to use while shipping files/folders to workers. Numerical (1 - n).
INSTALL_PATH_REMOTE	Path in each worker where the load generation binary (wileE), load specification file and setupcron script have been installed. Expected to be the same for each worker.
LAUNCH_DATE	Date when load generation should start. Format: yyyy-mm-dd.
LAUNCH_TIME	Time when load generation should start. Format: HH:MM:SS (24 hour clock).
END_DATE	Date when load generation should end. Format: yyyy-mm-dd.
END_TIME	Time when load generation should end. Format: HH:MM:SS (24 hour clock).
CONNECT_TIMEOUT_SECS	Timeout to use with "ssh". If some worker nodes don't respond, then this will ensure that the script doesn't "hang" for a long time. Numerical (0-n) seconds.
REMOTE_USERNAME	The username to use to login to the workers. This should be identical for each worker and passwordless ssh should have been set up apriori between the machine shipping data and the workers.

Load generator specific parameters discussed a few sub-sections ago can be setup in the "loadgen" section of launchspec.conf file. The options are:

Section: "loadgen"

Parameter	Description
NUM_LGEN_THREADS	Number of identical threads to use in each worker for load generation. Numerical (1 - n).
SMALLEST_LOOP_LENGTH_US	Smallest loop length, in microseconds, in which to achieve desired CPU utilization. Numerical (1000 - n).
RECALIBRATE_PERIODICALLY	Toggle recalibrate option in the load generator binary (wileE). Options: yes/no.

Example Load Generation Scenarios

I. Replaying Datacenter Resource Usage Traces

Recently, Google Inc. released a large scale, anonymized production workload trace from one of its clusters [3], containing data worth over 6 hours with samples taken once every five minutes. Each line in the trace represents the normalized resource usage of a single task, belonging to one of four possible job types, at a given timestamp. This example shows how to replay the resource usage "pattern" of the four job types on a hypothetical set of 100 machines using xerxes. Refer to the Xerxes paper for additional details on this example.

Step 1: Download the google traces from [3]. You should get a large gzipped file from the site. Extract the "csv" file from it. Lets assume that this file is available at the following location: <xerxes-root>/master/nodespec/traces/google-cluster-data-1.csv

Step 2: Configure the NodeSpec generator parameters in the template file <xerxes-root>/master/nodespec/loadspec.conf as follows:

```
# Global And Per-node Load Specifications

[common]

# Options: trace, statistical
# trace = use if you want to replay resource usage traces
# statistical = use if you want load generation to follow
#               common statistical distributions
TYPE = trace

[trace]

# Path to the input resource usage trace file.
# NOTE: Lines beginning with "#" in file are ignored.
DATA_FILE = ./traces/google-cluster-data-1.csv

# Separator used in input trace file
# Valid options: comma, whitespace
SEPARATOR = whitespace

## Column numbers below begin from 0.

# Column that identifies timestamps of samples
TIMESTAMP_COL = 0

# Column in trace the identifies machine or job
# from which utilization data has been collected.
ITEM_IDENTIFIER_COL = 3

# Column in trace that contains CPU util value.
CPU_COL = 4

# What format is utilization information specified?
# Options: absolute, relative
CPU_UTIL_FORMAT = relative
```

```
# Column in trace that contains Mem util value.
MEM_COL = 5

# What format is utilization information specified?
# Options: absolute, relative
MEM_UTIL_FORMAT = relative

# Max amount of memory that the per-node load
# generator can touch. It is often desired
# that this be < 100% to leave room for OS
# and other apps.
MEM_LIMIT_PCT = 40

[target]

# Number of nodes that will run load generation.
NODES = 100

# How long should each timepoint in final spec last?
TIMEPOINT_DURATION_SECS = 300

# Output directory
OUTPUT_DIRECTORY = ./loadfiles
```

Note that the NodeSpec generator is set to output individual worker node files to the directory <xerxes-root>/master/nodespec/loadfiles/, which must have been created prior to invoking the generator script. The column numbers have been picked based on the first comment line in the trace file describing each column. The utilization formats for CPU and memory is set to relative since the values reported in the trace files are normalized using an unknown transformation. Relative will generate worker utilizations relative to the observed maximum utilization value in the file.

Step 3: Run the NodeSpec generator with the config file created above as input:

```
cd <xerxes-root>/master/nodespec/
python generator.py loadspec.conf
```

This may take a few minutes since the input trace file is quite large. Verify that the worker node load specs have been generated successfully by looking into the loadfiles/ folder. You should see files node0.dat to node99.dat for the 100 total workers.

Step 4: Now these worker load specification files need to be shipped to each worker. Configure the parameters for the ship files script using the template <xerxes-root>/master/launcher/shipspec.conf as follows:

```
# config for shipping files/folders to remote nodes

[common]

# Location of file containing list of host DNS names
# or IP addresses along with their logical
# node number for load generation.
# NOTE: Lines beginning with "#" in file are ignored.
HOSTFILE = ./hostfile

# Separator used in hostfile.
# Options: comma, whitespace
SEPARATOR = comma

# Column numbers pertaining to hostfile above
# begin from 0.

# Column containing logical node number in hostfile.
HOSTFILE_NODENUM_COL = 0

# Column containing host DNS names or IP
# addresses in hostfile.
HOSTFILE_HOST_COL = 1

# Number of threads to use while
# shipping scripts
NUM_THREADS = 4

# Path to the data that is to be
# shipped.
LOCAL_SHIP_DATA_PATH = ../nodespec/loadfiles

# Destination dir on remote node
# where data is to be shipped.
REMOTE_SHIP_DATA_PATH = ~

# If set to yes, script will only copy
# to each node the load spec generated
# for it. It expects the ship_data_path
# to be the root folder containing
# all the specs
SHIPPING_LOAD_SPECS = yes

# Timeout option for scp or ssh
# in seconds. This will allow to
# skip failed or faulty nodes
# quicker.
CONNECT_TIMEOUT_SECS = 10

# Remote node username.
# NOTE: passwordless ssh must have
# been setup apriori. Same username
# will be used for all nodes.
REMOTE_USERNAME = root
```

Note that the wileE binary and setupcron.sh script are assumed to be installed in the root user's home directory. We've set the SHIPPING_LOAD_SPECS parameter to "yes" and pointed the LOCAL_SHIP_DATA_PATH to the folder with the worker node specs generated in Step 2.

Create <xerxes-root>/master/launcher/hostfile as follows (in line with the column numbers specified in the spec above):

```
0, 10.2.2.0
1, 10.2.2.1
2, 10.2.2.2
3, 10.2.2.3
.
.
97, 10.2.2.97
98, 10.2.2.98
99, 10.2.2.99
```

Not all entries are shown for the sake of brevity.

Step 5: Ship the worker load spec files:

```
cd <xerxes-root>/master/launcher/
python shipfiles.py shipspec.conf
```

Watch out for any errors in communicating with remote worker nodes. You may need to manually resolve the communication issue with the erroneous remote nodes and re-copy their respective load spec files.

Step 6: Configure the load generation launch parameters using the template file <xerxes-root>/master/launcher/launchspec.conf as follows:

```
# hostfile, installation path, launch time, end time etc.

[common]

# Location of file containing list of host DNS names
# or IP addresses along with their logical
# node number for load generation.
# NOTE: Lines beginning with "#" in file are ignored.
HOSTFILE = ./hostfile

# Separator used in hostfile.
# Options: comma, whitespace
SEPARATOR = comma

# Column numbers pertaining to hostfile above
# begin from 0.
```

```
# Column containing logical node number in hostfile.
HOSTFILE_NODENUM_COL = 0

# Column containing host DNS names or IP
# addresses in hostfile.
HOSTFILE_HOST_COL = 1

# Location of (1) load generator,
# (2) load spec file & (3) cron script
# in remote node.
# Note: All files are assumed to be in
# the same location below.
INSTALL_PATH_REMOTE = ~

# What date/time should the load
# generation start?
# Format: yyyy-mm-dd
LAUNCH_DATE = 2013-6-1

# Format: HH:MM:SS (24 hr)
LAUNCH_TIME = 13:00:00

# If you want the load generation
# to end before all samples are
# through, what date/time do you
# want it to end?
# Format: yyyy-mm-dd
END_DATE = 2013-6-1

# Format: HH:MM:SS (24 hr)
END_TIME = 18:00:00

# Number of threads to use while
# shipping or launching scripts
NUM_THREADS = 4

# Timeout option for scp or ssh
# in seconds. This will allow to
# skip failed or faulty nodes
# quicker.
CONNECT_TIMEOUT_SECS = 10

# Remote node username.
# NOTE: passwordless ssh must have
# been setup apriori. Same username
# will be used for all nodes.
REMOTE_USERNAME = root

[loadgen]

# Number of threads to use in each
# remote node for load generation.
# Comes in handy for multi-core
# machines.
NUM_LGEN_THREADS = 2
```

```
# Smallest loop length in
# microseconds in which to
# achieve desired cpu utilization.
SMALLEST_LOOP_LENGTH_US = 1000000

# Should the load generator
# periodically re-calibrate cpu
# load generation loop? Comes in
# handy in virtualized systems
# where the time taken to perform
# a certain number of ops varies
# with consolidation ratios.
RECALIBRATE_PERIODICALLY = yes
```

The same hostfile created in Step 4 is now reused. The INSTALL_PATH_REMOTE parameter is set to the same folder where we shipped the worker node load spec file (also the same folder where wileE binary and setupcron.sh should be present).

Step 7: Launch setup of load generation experiment.

```
cd <xerxes-root>/master/launcher/
python launch.py launchspec.conf
```

Now at the specified start date and time, all the worker nodes will start generating CPU load and memory allocations. The transition between multiple utilization values specified in the worker load file (300 seconds as in loadspec.conf in Step 2) will happen in such a way that the overall resource usage “pattern” of the four job types in the original input trace file is largely preserved.

II. Generating Statistical Load Patterns

In this example, we define a set of workloads following a gaussian distribution of utilization values and have it replayed on a hypothetical set of 100 worker machines.

Step 1: Define a workload description file <xerxes-root>/master/nodespec/stat/gauss.txt as follows:

```
#workload_id, mean, deviation
0, 2, 0.5
1, 4, 1.5
2, 1, 1.5
3, 4, 1.0
4, 5, 1.0
5, 5, 1.0
6, 5, 1.0
7, 5, 1.0
8, 13, 6.5
9, 10, 8.5
```

Note that the utilization is assumed to be within 0 - 100 range and the standard deviation is in absolute terms.

Step 2: Configure the NodeSpec generator parameters in the template file <xerxes-root>/master/nodespec/loadspec.conf as follows:

```
# Global And Per-node Load Specifications

[common]

# Options: trace, statistical
# trace = use if you want to replay resource usage traces
# statistical = use if you want load generation to follow
#               common statistical distributions
TYPE = statistical

[statistical]

# Type of statistical distribution to be used for
# load generation.
# Options: uniform, gaussian
DISTRIBUTION_CPU = gaussian
DISTRIBUTION_MEM = gaussian

# Workload description file
# The format is specific to the distribution required.
WORKLOAD_DESC_FILE_CPU = ./stat/gauss.txt
WORKLOAD_DESC_FILE_MEM = ./stat/gauss.txt

# Separator used in workload desc file.
# Valid options: comma, whitespace
SEPARATOR = comma

## Column numbers below begin from 0

# gaussian
GAUSSIAN_MEAN_COL = 1
GAUSSIAN_STDEV_COL = 2

# How long should the workload last overall?
WORKLOAD_DURATION_MINS = 180

# Max amount of memory that the per-node load
# generator can touch. It is often desired
# that this be < 100% to leave room for OS
# and other apps.
MEM_LIMIT_PCT = 40


[target]

# Number of nodes that will run load generation.
NODES = 100
```

```
# How long should each timepoint in final spec last?
TIMEPOINT_DURATION_SECS = 300

# Output directory
OUTPUT_DIRECTORY = ./loadfiles
```

As before, the NodeSpec generator is set to output individual worker node files to the directory <xerxes-root>/master/nodespec/loadfiles/, which must have been created prior to invoking the generator script. Note that here we've used the same workload description file for both CPU and memory resources. You can use different files with different distribution parameters as required.

Steps 3 through 7: the same as the previous example.

III. Generating Resource Usage Spikes

Now suppose that we want to add some spikes in resource usage to the base utilizations example II. The steps are as follows:

Step 1: Define a spike description file <xerxes-root>/master/nodespec/spikes/example.txt as follows:

```
# start_node, end_node, resource, peak_start, peak_start, peak_end, spike_end, magnitude
1, 14, mem, 0, 600, 900, 1200, 3
30, 49, cpu, 3600, 4320, 5400, 5760, 4
```

Here we've described two spikes: (i) the first for memory resource occurring on worker nodes (1 through 14, inclusive) at the very start of load generation (0 seconds start) and ending altogether 1200 seconds or 20 minutes from start of experiment (important note: not relative to start of spike!), (ii) the second spike is for CPU occurring on worker nodes (30 through 49, inclusive) starting an hour after the start of the experiment and lasting altogether till an hour and a half from the start of the experiment (i.e. total duration is 30 minutes).

Step 2: Configure the NodeSpec generator parameters in the template file <xerxes-root>/master/nodespec/loadspec.conf as follows:

```
# Global And Per-node Load Specifications

[common]

# Options: trace, statistical
# trace = use if you want to replay resource usage traces
# statistical = use if you want load generation to follow
#               common statistical distributions
TYPE = statistical
```

```
[statistical]

# Type of statistical distribution to be used for
# load generation.
# Options: uniform, gaussian
DISTRIBUTION_CPU = gaussian
DISTRIBUTION_MEM = gaussian

# Workload description file
# The format is specific to the distribution required.
WORKLOAD_DESC_FILE_CPU = ./stat/gauss.txt
WORKLOAD_DESC_FILE_MEM = ./stat/gauss.txt

# Separator used in workload desc file.
# Valid options: comma, whitespace
SEPARATOR = comma

## Column numbers below begin from 0

# gaussian
GAUSSIAN_MEAN_COL = 1
GAUSSIAN_STDEV_COL = 2

# How long should the workload last overall?
WORKLOAD_DURATION_MINS = 180

# Max amount of memory that the per-node load
# generator can touch. It is often desired
# that this be < 100% to leave room for OS
# and other apps.
MEM_LIMIT_PCT = 40


[target]

# Number of nodes that will run load generation.
NODES = 100

# How long should each timepoint in final spec last?
TIMEPOINT_DURATION_SECS = 300

# Output directory
OUTPUT_DIRECTORY = ./loadfiles

[spike]

SPIKES_DESC_FILE = ./spikes/example.txt

# Separator used in spikes desc file.
# Valid options: comma, whitespace
SEPARATOR = comma

# Columns in description file for each
# of the parameters required to generate
# spikes. Please refer to Xerxes paper
```

```
# for more details.

# Start and end nodes must be between
# 0 and NODES - 1 value above.
START_NODE_COL = 0
END_NODE_COL = 1

# Options: cpu or mem.
SPIKE_RESOURCE_COL = 2

# Specify time in seconds from start.
# Values will be rounded off to
# nearest timepoint_duration_secs.
TIME_SPIKE_START_COL = 3
TIME_PEAK_START_COL = 4
TIME_PEAK_END_COL = 5
TIME_SPIKE_END_COL = 6

# Note that the magnitude for CPU spikes
# will clip at 100% and that of memory
# at the limit specified.
SPIKE_MAGNITUDE_MULTIPLIER_COL = 7
```

Most of the above configuration is the same as the previous example except for the “spike” section at the very end.

Steps 3 through 7: the same as the previous example.

References

1. M. Kesavan et al. Xerxes: Distributed load generator for cloud-scale experimentation. In Open Cirrus Summit, 2012.
2. P. Bodik et al. Characterizing, modeling, and generating workload spikes for stateful services. In Symposium on Cloud Computing SoCC ’10.
3. googleclusterdata - traces of google tasks running in a production cluster,” http://code.google.com/p/googleclusterdata/wiki/TraceVersion1.