Benchmarking Suite Documentation

Release 3.1.0

Gabriele Giammatteo

Jan 22, 2019

Contents

The Benchmarking Suite is an all-in-one solution for benchmarking cloud services simulating different typical application behaviours and comparing the results on different cloud providers. It wraps a set of representative, de-facto standard and widely used third-party benchmarking tools and relies on them for the workload simulation and performance measurement.

The Benchmarking Suite automates the benchmarking process managing the allocation and de-allocation of necessary resources, the installation and execution of the benchmarking tools and the storage of data.

It has been designed to be extendible and allow an easy integration of new third-party benchmarking tools and cloud services. Data collected and stored during the tests execution is homogenized and aggregated on different higher-level metrics (e.g. average value) allowing performance comparisons among different providers and/or different dates.

The Benchmarking Suite development has been funded by two European reasearch and innovation projects: ARTIST[1] and CloudPerfect[2].

[1] http://www.artist-project.eu/
[2] https://cloudperfect.eu/

CHAPTER 1

License

The Benchmarking Suite is an open source product released under the Apache License v2.0[3].

[3] https://www.apache.org/licenses/LICENSE-2.0

CHAPTER 2

Topics

2.1 Quick Start

2.1.1 Install

The Benchmarking Suite is package and distributed through PyPI[1].

Important: The Benchmarking Suite requires Python 3.5+. If it is not the default version in you system, it is recommended to create a virtualenv:

```
virtualenv -p /usr/bin/python3.5 benchmarking-suite
source benchsuite/bin/activate
```

Let's start by installing the command line tool and the standard library:

```
$ pip install benchsuite.stdlib benchsuite.cli
```

This will make available the `benchsuite` bash command and will copy the standard benchmark tests configuration into the default configuration location (located under `~/.config/benchmarking-suite/benchmarks`).

2.1.2 Configure

Before executing a benchmark, we have to configure at least one Service Provider. The `benchsuite.stdlib` provides some template (located under `~/.config/benchmarking-suite/providers`).

For instance, for Amazon EC2 we can start from the template and complete it:

```
cp ~/.config/benchmarking-suite/providers/amazon.conf.example my-amazon.conf
```

[1] https://python.org/pypi/benchsuite.core/

Open and edit `my-amazon.conf`

```
[provider]
class = benchsuite.provider.libcloud.LibcloudComputeProvider
type = ec2

access_id = <your access_id>
secret_key = <your secret_key>

[ubuntu_micro]
image = ami-73f7da13
size = t2.micro
```

In this case we will provide this file directly to the command line tool, but we can also configure our own configuration directory, put all our service providers and benchmarking tests configuration there and refer to them by name.

(Full specification of the configuration files syntax, can be found in the "Service Providers" sections).

2.1.3 Run!

Now you can execute your first benchmark test:

```
benchsuite multiexec --provider my-amazon.conf --service ubuntu_micro ycsb-
↪mongodb:workloada
```

2.1.4 Go REST

Enable the REST server is very simple:

```
pip install benchsuite.rest
benchsuite-rest start
tail -f benchsuite-rest.log
```

2.1.5 References

2.2 Architecture

The Benchmarking Suite is composed by five main components summarized in the following diagram:

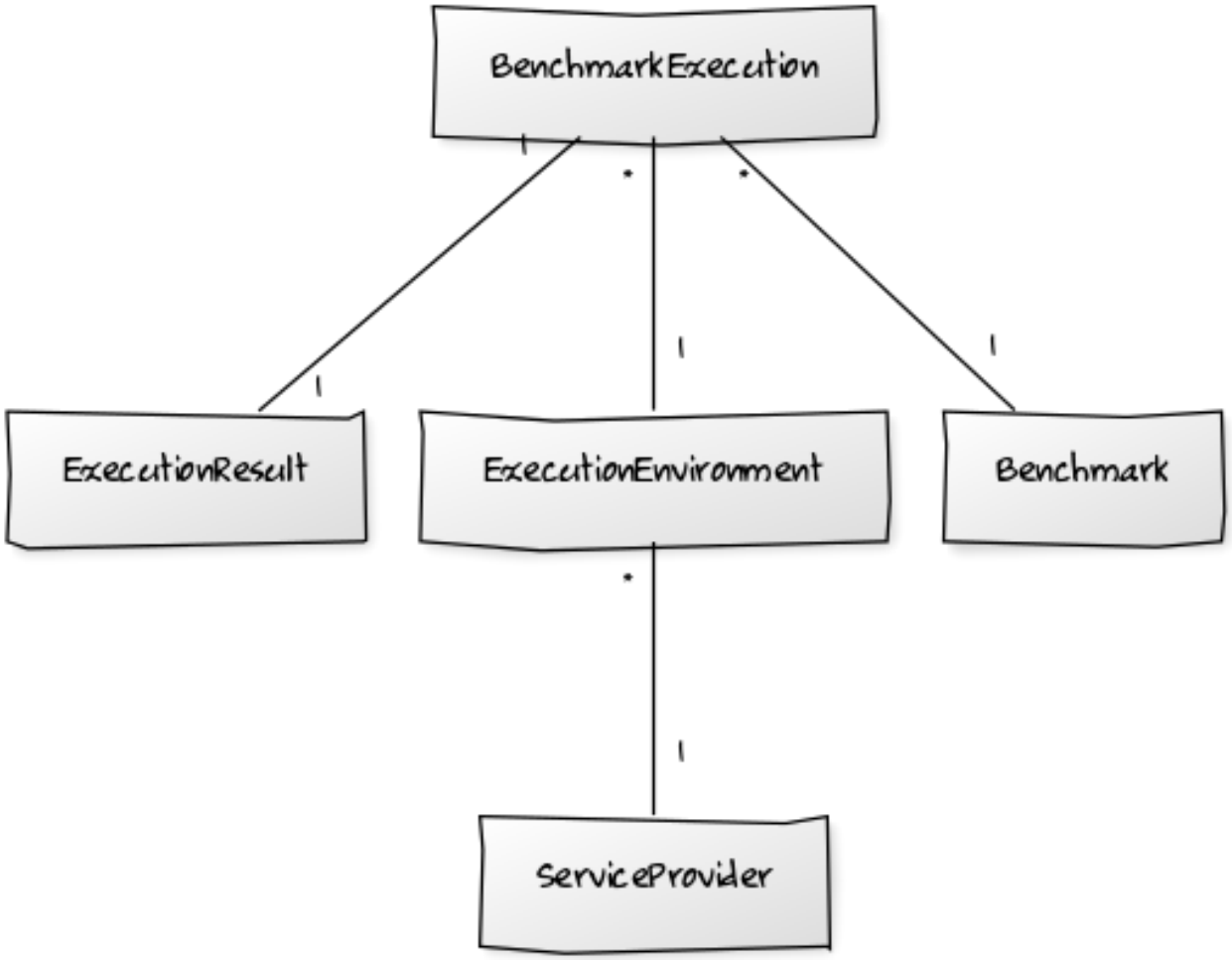
- **Core**: defines the main model, the extension framework for the benchmarks and the Cloud providers and the main data representation types;
- **REST**: a REST server to access all the Benchmarking Suite functionalities;
- **CLI**: a command line tool (bash-compatible) to access all the Benchmarking Suite functionalities;
- **Std Library**: a set of selected benchmark tools, including their configurations and the implementation of the required wrapping scripts for the execution and the parsing of results;
- **Backend Connectors**: a set of connectors to store the Benchmarking Suite executions results on different storage technologies (e.g. MySQL, MongoDB).
- **Scheduler**: a service that periodically executes a benchmarking session;

The *Core* component is the only required component, the other components are optional. However the Benchmarking Suite installation will miss the functionalities of not-installed modules (e.g. if the *Backend Connectors* is not installed, the execution results will not be stored).

The *User's Cloud Configuration* is the required configuration of the Cloud Providers that the Benchmarking Suite needs to be able to access the *Target Cloud Provider*. It can be specified either as configuration file or as parameter in the execution requests (through the REST or CLI components). Refer to section *Service Providers* for further details

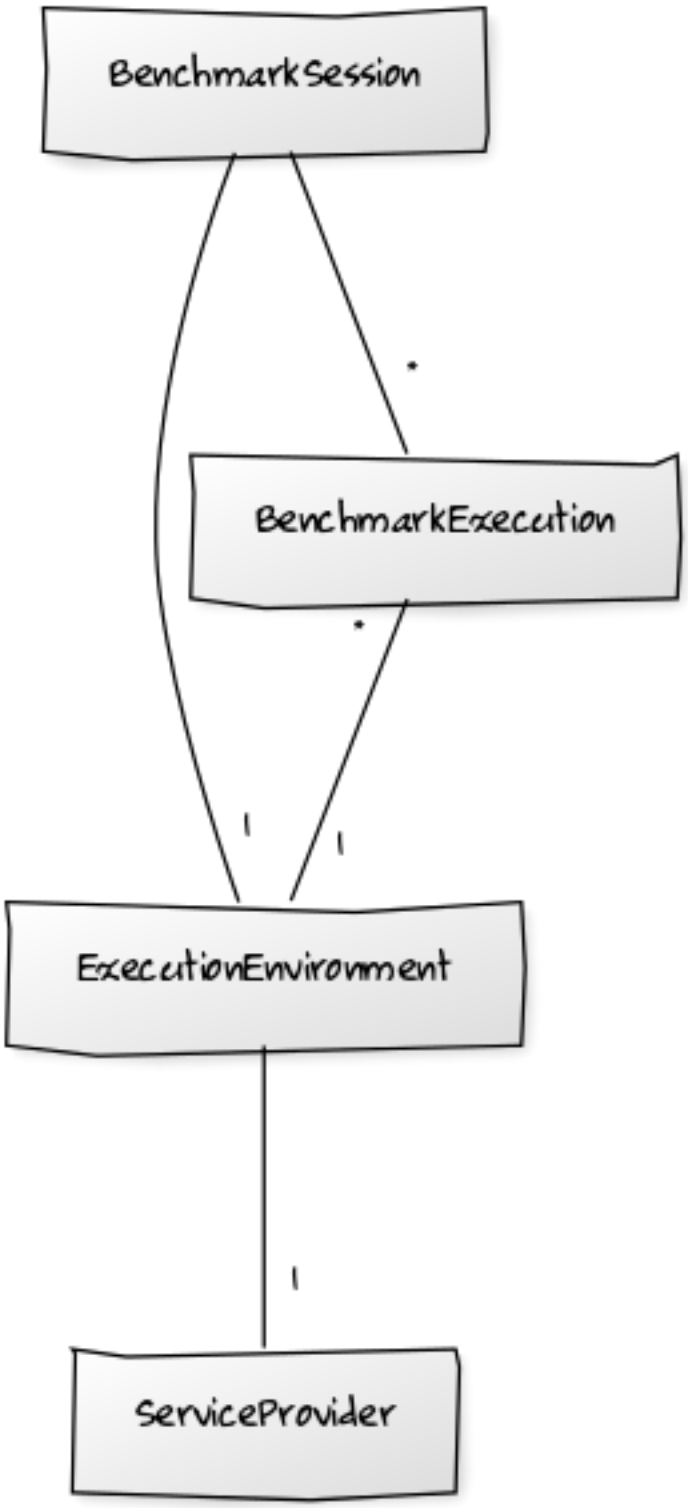
2.2.1 Domain Model

The core concept in the Benchmarking Suite is the **BenchmarkExecution**. It represents the execution of a **Benchmark** test against an **ExecutionEnvironment** provided from a **ServiceProvider** and produces an **ExecutionResult**.



Note: For instance, following this model we can easily model the execution of YCSB.WorkloadA (the *Benchmark*) on the Virtual Machine with ip=50.1.1.1 (the *ExecutionEnvironment*) provided by Amazon EC2 (the *ServiceProvider*).

Since it is frequent to execute multiple tests against the same Service Provider, the Benchmarking Suite has also the concept of **BenchmarkingSession**. that can gropu one or more executions of the ServiceProvider, using the same ExecutionEnvironment.



2.2.2 Software Modules

In order to address all the different use cases and the installation necessities, the Benchmarking Suite is distributed in six different software modules that can be installed separately:

`benchsuite.core`	the core library (all other modules depend on it) with the definition of types and the fundamental framework for the extension of the Benchmarking Suite
`benchsuite.stdlib`	a collection of benchmark tests configuration files and support for some Cloud Providers
`benchsuite.cli`	a bash command line tool to manage tests and results
`benchsuite.rest`	an HTTP server and a REST API to interact with the Benchmarking Suite
`benchsuite.backend`	connectors for the supported storage backends
`benchsuite.scheduler`	a service to automatically execute periodic benchmarks

2.3 Benchmarks

The Benchmarking Suite comes with a set of third-party benchmarking tools, each of them with a set of different test configurations ready to be executed. The tools are:

- **CFD**: a tool realized in the CloudPerfect EU project[1] that uses OpenFOAM to run a waterbox simulation. Can be configured with different solvers, number of iterations and write to disk strategies. It is primarily a CPU intensive benchmark;
- **DaCapo**: a tool for Java benchmarking simulating real world applications with non-trivial memory loads. It is mainly a CPU and memory intensive benchmark;
- **Filebench**: a powerful and flexible tool able to generate and execute a variety of filesystem workloads to simulate applications like Web servers, File servers, Video services. It is mainly a Disk intensive benchmark;
- **Iperf**: is a tool for active measurements of the maximum achievable bandwidth on IP networks;
- **Sysbench**: a tool to test CPU, memory, file I/O, mutex performance and MySQL on Linux systems;
- **YCSB**: a tool for database benchmarking that supports several database technologies. In the Benchmarking Suite, tests for Mysql and MongoDB are provided. It is primarily a Disk intensive benchmark;
- **WebFrameworks**: tests common web frameworks workloads like fetching and inserting data in a database or create/parse json objects. It is mainly a Memory and Network intensive benchmark;

The following table summarizes the tools available and their compatibility with different operating system.

Table 1: Test-OS compatibility matrix

Tool	Version	CentOS	Ubuntu 14	Ubuntu 16
CFD	1.0		✓	
DaCapo	9.12	✓	✓	
Filebench	1.4.9.1	✓	✓	✓
Iperf	2.0.5		✓	✓
Sysbench	2.1.0		✓	✓
YCSB-MySQL	0.12.0	✓	✓	
YCSB-MongoDB	0.11.0	✓	✓	
WebFrameworks	master		✓	✓

[1] CloudPerect project homepage: http://cloudperfect.eu/

2.3.1 CFD

The CFD benchmarking tool has been realized in the context of the CloudPerfect EU project[1] and released open source on GitHub[2]. The tool executes a CFD simulation on a waterbox geometry allowing to customize several parameters in order to simulate different simulations.

The following combination of parameters is used in the Benchmarking Suite tests:

100iterGAMG	100 iterations using the GAMG solver
100iterWriteAtLast	100 iterations using the GAMG solver and not writing intermediate results on the disk
500iterGAMG	500 iterations using the GAMG solver
500iterGAMGWriteAtLast	500 iterations using the GAMG solver and not writing intermediate results on the disk
500iterICCG	500 iterations using the ICCG solver
500iterPCG	500 iterations using the PCG solver

All the tests uses all the CPUs available in the machine.

Metrics

Metric	Unit	Description
duration	s	The overall duration of the simulation

2.3.2 DaCapo

DaCapo[3] as a tool for Java benchmarking by the programming language, memory management and computer architecture communities. It consists of a set of open source, real world applications with non-trivial memory loads. Tests implemented by the tool are:

[2] CFD Benchmark Case code: https://github.com/benchmarking-suite/cfd-benchmark-case
[3] DaCapo homepage: http://www.dacapobench.org/

Table 2: DaCapo tests (source: http://www.dacapobench.org/)

avrora	simulates a number of programs run on a grid of AVR microcontrollers
batik	produces a number of Scalable Vector Graphics (SVG) images based on the unit tests in Apache Batik
eclipse	executes some of the (non-gui) jdt performance tests for the Eclipse IDE
fop	takes an XSL-FO file, parses it and formats it, generating a PDF file.
h2	executes a JDBCbench-like in-memory benchmark, executing a number of transactions against a model of a banking application, replacing the hsqldb benchmark
jython	inteprets a the pybench Python benchmark
luindex	Uses lucene to indexes a set of documents; the works of Shakespeare and the King James Bible
lusearch	Uses lucene to do a text search of keywords over a corpus of data comprising the works of Shakespeare and the King James Bible
pmd	analyzes a set of Java classes for a range of source code problems
sunflow	renders a set of images using ray tracing
tomcat	runs a set of queries against a Tomcat server retrieving and verifying the resulting webpages
tradebeans	runs the daytrader benchmark via a Jave Beans to a GERONIMO backend with an in memory h2 as the underlying database
tradesoap	runs the daytrader benchmark via a SOAP to a GERONIMO backend with in memory h2 as the underlying database
xalan	transforms XML documents into HTML

Each test is executed multiple times, until the exectuions duration converge (variance is <= 3.0 in the latest 3 executions).

Metrics

Metric	Unit	Description
timed_duration	ms	the duration of the latest execution
warmup_iters	num	the number of executions that were necessary to converge

2.3.3 Filebench

Filebench[4] is a very powerful tool able to generate a variety of filesystem- and storage-based workloads. It implements a set of basic primitives like *createfile*, *readfile*, *mkdir*, *fsync*, ... and provide a language (the Workload Model Language - WML) to combine these primitives in complex workloads.

In the Benchmarking Suite, a set of pre-defined workloads have been used to simulate different services:

[4] Filebench homepage: https://github.com/filebench/filebench/wiki

Table 3: Filebench workloads (source: https://github.com/filebench/filebench/wiki/Predefined-personalities)

file-server	Emulates simple file-server I/O activity. This workload performs a sequence of creates, deletes, appends, reads, writes and attribute operations on a directory tree. 50 threads are used by default. The workload generated is somewhat similar to SPECsfs.
webproxy	Emulates I/O activity of a simple web proxy server. A mix of create-write-close, open-read-close, and delete operations of multiple files in a directory tree and a file append to simulate proxy log. 100 threads are used by default.
web-server	Emulates simple web-server I/O activity. Produces a sequence of open-read-close on multiple files in a directory tree plus a log file append. 100 threads are used by default.
videoserver	This workloads emulates a video server. It has two filesets: one contains videos that are actively served, and the second one has videos that are available but currently inactive. One thread is writing new videos to replace no longer viewed videos in the passive set. Meanwhile $nthreads threads are serving up videos from the active video fileset.
var-mail	Emulates I/O activity of a simple mail server that stores each e-mail in a separate file (/var/mail/ server). The workload consists of a multi-threaded set of create-append-sync, read-append-sync, read and delete operations in a single directory. 16 threads are used by default. The workload generated is somewhat similar to Postmark but multi-threaded.

Metrics

Metric	Unit	Description
duration	s	The overall duration of the test
ops	num	The sum of all operations (of any type) executed
ops_throughput	ops/s	The average number of operations executed per second
throughput	MB/s	The average number of MBs written/read during the test
cputime	µs	The average cpu time taken by each operation
latency_avg	µs	The average duration of each operation

2.3.4 Iperf

IPerf[5] is a benchmarking tool to measure the maximum achievable bandwidth on IP networks. It provides statistics both for TCP and UDP protocols.

In the Benchmarking Suite, the following pre-defined workloads have been created:

tcp_10_1	transfer data over a single TCP connections for 10 seconds
tcp_10_10	transfer data over 10 parallel TCP connections for 10 seconds
udp_10_1_1	transfer UDP packets over a single connection with a maximum bandwidth limited at 1MBit/s
udp_10_1_10	transfer UDP packets over a single connection with a maximum bandwidth limited at 10MBit/s
udp_10_10_10	transfer UDP packets over 10 parallel connections with a maximum bandwidth limited at 1MBit/s

Metrics

For the TCP workloads:

[5] IPerf homepage: https://iperf.fr/

Metric	Unit	Description
duration	s	The overall duration of the test
transferred_x	bytes	data transferred for the connection x
bandwidth_x	bit/s	bandwidth fo the connection x
transferred_sum	bytes	sum of data transferred in all connections
bandwidth_sum	bit/s	sum of bandwidth of all connections

For the UDP workloads:

Metric	Unit	Description
duration	s	The overall duration of the test
transferred_x	bytes	data transferred over connection x
bandwidth_x	bit/s	bandwidth of connection x
total_datagrams_x	num	number of UDP packets sent over connection x
lost_datagrams_x	num	number of lost UDP packets over connection x
jitter_x	ms	latency of connection x
outoforder_x	num	number of packets received by the server in the wrong order
transferred_avg	bytes	average data transferred by each connection
bandwidth	bit/s	average bandwidth of each connection
total_datagrams_avg	num	average number of packets sent over each connection
lost_datagrams_avg	num	average number of packets lost for each connection
jitter_avg	ms	average latency
outoforder_avg	num	average number of packets received in the wrong order

2.3.5 Sysbench

SysBench[6] is a modular, cross-platform and multi-threaded benchmark tool for evaluating CPU, memory, file I/O, mutex performance, and even MySQL benchmarking. At the moment, in the Benchmarking Suite only the CPU benchmarking capabilities are integrated.

cpu_10000	Verifies prime numbers between 0 and 20000 by doing standard division of the number by all numbers between 2 and the square root of the number. This is repeated 1000 times and using 1, 2, 4, 8, 16 and 32 threads

Metrics

Metric	Unit	Description
events_rate_X	num/s	the number of times prime numbers between 0 and 20000 are verified each second with X threads
total_time_X	s	total number of seconds it took to execute the 1000 cycles with X threads
latency_min_X	ms	minimum time it took for a cycle
latency_max_X	ms	maximum time it took for a cycle
latency_avg_X	ms	average time the 1000 cycles took. It gives a good measure of the cpu speed
latency_95_X	ms	95th percentile of the latency times.

[6] Sysbench homepage: https://github.com/akopytov/sysbench

2.3.6 YCSB

YCSB[7] is a database benchmarking tool. It has the support for several database technologies and provides a configuration mechanism to simulate different usages.

In the Benchmarking Suite, YCSB is used to benchmark two of the most popular database servers: **MySQL** and **MongoDB**.

For each database, the following workloads are executed:

workloada	Simulates an application that performs read and update operations with a ratio of 50/50 (e.g. recent actions recording)
workloadb	Simulates an application that performs read and update operations with a ratio of 95/5 (e.g. photo tagging)
workloadc	Simulates a read-only databases (100% read operations)
workloadd	Simulates an application that performs read and insert operations with a ratio of 95/5 (e.g. user status update)
workloade	Simulates an application that performs scan and insert operations with a ratio of 95/5 (e.g. threaded conversations)
workloadf	Simulates an application that performs read and read-modify-write operations with a ratio of 50/50 (e.g. user database)

Metrics

Metric	Unit	Description
duration	s	The overall duration of the test
read_ops	num	THe number of read operations executed
read_latency_avg	µs	The average latency of the read operations
read_latency_min	µs	The minimum latency of the read operations
read_latency_max	µs	The maximum latency of the read operations
read_latency_95	µs	The maximum latency for the 95% of the read operations
read_latency_99	µs	The maximum latency for the 99% of the read operations
insert_ops	num	THe number of insert operations executed
insert_latency_avg	µs	The average latency of the insert operations
insert_latency_min	µs	The minimum latency of the insert operations
insert_latency_max	µs	The maximum latency of the insert operations
insert_latency_95	µs	The maximum latency for the 95% of the insert operations
insert_latency_99	µs	The maximum latency for the 99% of the insert operations
update_ops	num	THe number of update operations executed
update_latency_avg	µs	The average latency of the update operations
update_latency_min	µs	The minimum latency of the update operations
update_latency_max	µs	The maximum latency of the update operations
update_latency_95	µs	The maximum latency for the 95% of the update operations
update_latency_99	µs	The maximum latency for the 99% of the update operations

[7] YCSB homepage: https://github.com/brianfrankcooper/YCSB/wiki

2.3.7 WebFrameworks

This is an open source tool[8] used to compare many web application frameworks executing fundamental tasks such as JSON serialization, database access, and server-side template composition. The tool has been developed and it is used to run the tests that generate the results available at: https://www.techempower.com/benchmarks/.

Currently, in the Benchmarking Suite the framework supported are: **Django**, **Spring**, **CakePHP**, **Flask**, **FastHttp** and **NodeJS**.

For each framework the following tests are executed:

Table 4: Test types (source: https://www.techempower.com/benchmarks/#section=code&hw=ph)

json	This test exercises the framework fundamentals including keep-alive support, request routing, request header parsing, object instantiation, JSON serialization, response header generation, and request count throughput.
query	This test exercises the framework's object-relational mapper (ORM), random number generator, database driver, and database connection pool.
fortunes	This test exercises the ORM, database connectivity, dynamic-size collections, sorting, server-side templates, XSS countermeasures, and character encoding.
db	This test uses a testing World table. Multiple rows are fetched to more dramatically punish the database driver and connection pool. At the highest queries-per-request tested (20), this test demonstrates all frameworks' convergence toward zero requests-per-second as database activity increases.
plaintext	This test is an exercise of the request-routing fundamentals only, designed to demonstrate the capacity of high-performance platforms in particular. Requests will be sent using HTTP pipelining.
update	This test exercises the ORM's persistence of objects and the database driver's performance at running UPDATE statements or similar. The spirit of this test is to exercise a variable number of read-then-write style database operations.

For the types *json*, *query*, *fortunes* and *db* the tool executes six different burst of requests. Each burst last 15 seconds and have a different concurrency level (number of requests done concurrently): 16, 32, 64, 128, 256 and 512.

For the type *plaintext*, the tool executes four burst of 15 seconds each with the following concurrency levels: 256, 1024, 4096 and 16384.

For the type *update*, the tool executes five burst of 15 seconds each with a 512 concurrency level, but different number of queries to perform: 1, 5, 10, 15 and 20.

Metrics

Metric	Unit	Description
duration	s	The overall duration of the test
duration_N	s	The overall duration for the N concurrency level*. It is fixed to 15 seconds by default
totalRequests_N	num	The overall number of requests processed during the 15 seconds test at the N concurrency level*
timeout_N	num	The number of requests that went in timeout for the N concurrency level*
latencyAvg_N	s	the average latency between a request and its response for the N concurrency level*
latencyMax_N	s	the maximum latency between a request and its response for the N concurrency level*
latencyStdev_N	s	the standard deviation measure for the latency for the N concurrency level*

[8] Web Framewoks Benchmarking code: https://github.com/TechEmpower/FrameworkBenchmarks

2.4 Service Providers

2.4.1 Configuration

Each provider has its own configuration file that provides all the information to access and use the provider plus the configuration of the services offered by the provider (e.g. VM creation). The configuration is specified in a properties file (also in JSON format). The list of supported properties is shown below (see on GitHub[1]):

```
#
# PROVIDER SECTION
#
#

[provider]

#
# MANDATORY FIELDS
#

# the Benchmarking Suite class that implement the access to this provider
class = benchsuite.stdlib.provider.libcloud.LibcloudComputeProvider

# custom human-readable name for the provider. Used only for displaying purposes
name = ote-testbed

# the driver that Libcloud should use to access the provider
driver = openstack

# Access credentials
access_id = admin
secret_key = xxxxx

# Authentication URL for the target Cloud
auth_url = http://cloudpcntlr:5000/


#
# OPTIONAL FIELDS
#

# If not specified RegionOne is used
region = RegionOne

# if not specified 2.0_password and 3.x_password will be attempted
auth_version = 3.x_password

# if not specified, a new security group "benchsuite_sg" will be created (if
# not exist)
security_group = mysg

# Automatically selected if multiple exist
network = mynet

# If not specified the access_id is used
tenant = admin
```

(continues on next page)

[1] https://github.com/benchmarking-suite/benchsuite-stdlib/blob/master/data/providers/openstack.conf.example

(continued from previous page)

```
# If specified do not attemp to assign a public IP
benchsuite.openstack.no_floating_ip = true


# after a new VM is created, a connection test is executed to check that everything
# is ok and to execute the post-creation scripts.
new_vm.connection_retry_period = 30
new_vm.connection_retry_times = 10


#
# SERIVCE TYPE SECTIONS
#
#

[ubuntu_large]

#
# MANDATORY FIELDS
#

image = base_ubuntu_14.04
size = m1.large


#
# OPTIONAL FIELDS
#

# name of the keypair to assign to the VM. If not specified, a new keypair will
# be created and then removed after the test ends.
key_name = ggiammat-key

# file that contains the private key. Alternatively the private key can be
# specified in this config file directly with the ssh_private_key property
key_path = /home/ggiammat/credentials/filab-vicenza/ggiammat-key.pem

# optional way to specify the key directly in the configuration instead
# of by filename
ssh_private_key = -----BEGIN RSA PRIVATE KEY-----
    MIIEowIBAAKCAQEAkadPr5n1NSOyHloajvovCD05M5Gz36NN4UouSWmId8QuTwXx
    Hw6m9aOXJmYHdkSYLrNs+y65EDpUkw1DXNDEJ146ZK9PxAQEdcngwPk76a4A/ybz
    [...]
    x+GRpQ9o/4EAzpBw9NVNNJ9Glbd7SSFqhpHR5pn5OBG/fdPJV8DzjUET528o8Jd9
    gynGwAYRed38UtCE7gn+u1RSvmYUveDwQ7Cf2KIohI2jlzR6YLea
    -----END RSA PRIVATE KEY-----

# If not specified, the Benchmarking Suite will try to guess them
vm_user = ubuntu
platform = ubuntu

# any command to run just after the VM has been created
post_create_script =
    sudo hostname localhost
```

2.5 Command line tool

2.5.1 Install

The Benchmarking Suite Command line tool can be installed with:

```
pip install benchsuite.cli
```

If the installation was successful, the `benchsuite` command should be in your path.

2.5.2 Configuration

The Benchmarking Suite has a configuration folder where the providers, benchmarks and storage backends configuration files are located. If not explicitly set, the configuration folder is the located in the system default configuration folder (e.g. `/home/<user>/.config` in Linux). It can be customized setting the `BENCHSUITE_CONFIG_FOLDER` env variable.

The Benchmarking Suite stores the active benchmarking sessions data on the filesystem. By default it is located under the system default data folder (e.g. `/Home/<user>/.local/share` in Linux). It can be customized setting the `BENCHSUITE_DATA_FOLDER` env variable.

2.5.3 Usage and Examples

Create a new session

To create a new session, the Benchmarking Suite needs two information: the provider configuration and the service type. The command line tool offers multiple options to specify these parameters.

Provider Configuration

There are different alternatives:

1. specify a name with the `--provider` option (e.g. `--provider myamazon`). In this case, a provider configuration file named `<name>.conf` will be searched in the configuration path;
2. specify a filename in the `--provider` option (e.g. `--provider /path/myamazon.conf`). The configuration file specified will be used;
3. store the provider configuration in the `BENCHSUITE_PROVIDER` environment variable.

As example, the following invocations load the same provider configuration:

```
$ benchsuite new-session --provider my-amazon.conf ...
```

```
$ benchsuite new-session --provider $BENCHSUITE_CONFIG_FOLDER/providers/my-amazon.
↪conf ...
```

```
$ export BENCHSUITE_PROVIDER=```cat $BENCHSUITE_CONFIG_FOLDER/providers/my-amazon.
↪conf```
$ benchsuite new-session ...
```

Service Type

The service type can be specified using the `--service-type` option (e.g. `--service-type ubuntu_micro`). The value of the service type must be one of the ones defined in the provider configuration. Alternatively the ‘‘BENCHSUITE_SERVICE_TYPE‘ enviornment variable can be used.

If neither the `--service-type` option nor the ‘‘BENCHSUITE_SERVICE_TYPE‘ enviornment variable are specified and the provider configuration defines only ONE service type, that one will be used, otherwise the invocation will fail.

2.5.4 Command Line Tool Documentation

This is an autogenerated documentation from the Python `argparse` options.

```
usage: benchsuite [-h] [--verbose] [--quiet] [--config CONFIG]
                  {shell,new-session,new-exec,prepare-exec,cleanup-exec,run-exec,list-
↪sessions,list-providers,list-benchmarks,destroy-session,list-execs,collect-exec,
↪multiexec}
                  ...
```

Named Arguments

--verbose, -v print more information (3 levels)

--quiet, -q suppress normal output

Default: False

--config, -c foo help

Sub-commands:

shell

Starts an interactive shell

```
benchsuite shell [-h]
```

new-session

Creates a new benchmarking session

```
benchsuite new-session [-h] [--provider PROVIDER]
                       [--service-type SERVICE_TYPE] [--property PROPERTY]
                       [--user USER] [--tag TAG]
```

Named Arguments

--provider, -p The name for the service provider configuration or the filepath of the provider configuration file. Alternatively, the provider configuration can be specified in the environment variable BENCHSUITE_PROVIDER (the content of the variable must be the actual configuration not the filepath)

--service-type, -s The name of one of the service types defined in the provider configuration. Alternatively, it can be specified in the BENCHSUITE_SERVICE_TYPE environment varaible

--property, -P	Add a user defined property to the session. The property must be expressed in the format <name>=<value>
--user, -u	sets the "user" property. It is a shortcut for "–property user=<name>
--tag, -t	sets one or more session tags. Internally, tags are stored as properties

Example: benchsuite new-session -p myamazon -s centos_tiny

new-exec

Creates a new execution

```
benchsuite new-exec [-h] session tool workload
```

Positional Arguments

session	a valid session id
tool	a valid benchmarking tool
workload	a valid benchmarking tool workload

Example: benchsuite new-exec 73cff747-d31a-488c-98f5-a70b9a77a11f filebench varmail

prepare-exec

Executes the install scripts for an execution

```
benchsuite prepare-exec [-h] id
```

Positional Arguments

id	a valid id of the execution

Example: benchsuite prepare-exec 4a5a86d4-88b6-11e7-9f96-742b62857160

cleanup-exec

Executed the cleanup script for an execution

```
benchsuite cleanup-exec [-h] id
```

Positional Arguments

id	a valid id of the execution

Example: benchsuite cleanup-exec 4a5a86d4-88b6-11e7-9f96-742b62857160

run-exec

Executes the execute scripts for an execution

```
benchsuite run-exec [-h] [--storage-config STORAGE_CONFIG] [--async] id
```

Positional Arguments

id	a valid id of the execution

Named Arguments

--storage-config, -r	Specify a custom location for the storage configuration file
--async	start the execution of the scripts and return (do not wait for the execution to finish) Default: False

Example: benchsuite run-exec 4a5a86d4-88b6-11e7-9f96-742b62857160

list-sessions

a help

```
benchsuite list-sessions [-h]
```

list-providers

a help

```
benchsuite list-providers [-h]
```

list-benchmarks

a help

```
benchsuite list-benchmarks [-h]
```

destroy-session

a help

```
benchsuite destroy-session [-h] id
```

Positional Arguments

id	bar help

list-execs

lists the executions

```
benchsuite list-execs [-h]
```

collect-exec

collects the outputs of an execution

```
benchsuite collect-exec [-h] id
```

Positional Arguments

id the execution id

multiexec

Execute multiple tests in a single benchmarking session

```
benchsuite multiexec [-h] [--provider PROVIDER] [--service-type SERVICE_TYPE]
                     [--storage-config STORAGE_CONFIG] [--property PROPERTY]
                     [--user USER] [--tag TAG] [--failonerror] [--keep-env]
                     [--max-retry MAX_RETRY]
                     tests [tests ...]
```

Positional Arguments

tests one or more tests in the format <tool>[:<workload>]. If workload is omitted, all workloads defined for that tool will be executed

Named Arguments

--provider, -p The name for the service provider configuration or the filepath of the provider configuration file

--service-type, -s The name of one of the service types defined in the provider configuration. If not specified, all service types will be used

--storage-config, -r Specify a custom location for the storage configuration file

--property, -P Add a user defined property to the session. The property must be expressed in the format <name>=<value>

--user, -u sets the "user" property. It is a shortcut for "–property user=<name>

--tag, -t sets one or more session tags. Internally, tags are stored as properties

--failonerror, -e If set, exits immediately if one of the tests fail. It is false by default

Default: False

--keep-env, -k If set, doesn't destroy the session after the end of the tests

Default: False

--max-retry, -m If a test fails, retry for a maximum of max-retry times (default is 1)

Example: benchsuite multiexec -p myamazon -s centos_tiny cfd:workload1 ycsb:workloada ycsb:workloadb

2.6 REST Server

2.6.1 Quick Start

This short tutorial shows how to use the API to perform a step-by-step benchmarking test.

First, we need to create a new session. This can be done making a `POST` request to `/api/v1/sessions` providing the provider name and service type.

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/
↪json' -d '{\
    "provider": "my-provider",\
    "service": "my-service-type"\
}' http://localhost:5000/api/v1/sessions
```

Alternatively, the provider configuration can be provided directly in the request payload. For instance, a typical request to create a benchmarking session for Amazon EC2 would be:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/
↪json' -d '{\
    "config": {\
        "provider": {\
                "class": "benchsuite.stdlib.provider.libcloud.LibcloudComputeProvider
↪",\
                    "name": "ec2-ggiammat",\
                    "driver": "ec2",\
                    "access_id": "<your_access_id>",\
                    "secret_key": "<your_key>",\
                    "region": "us-west-1"\
            },\
            "centos_micro": {\
                    "image": "ami-327f5352",\
                    "size": "t2.micro",\
                    "vm_user": "ec2-user",\
                    "platform": "centos_6",\
                    "key_name": "<keypair_name>",\
                    "ssh_private_key": "----BEGIN RSA PRIVATE KEY-----\nIIE... [...] .
↪..6alL\n-----END RSA PRIVATE KEY-----"\
            }\
    }\
}' http://localhost:5000/api/v1/sessions/
```

Important: Providing the configuration directly in the request payload, your credentials will be sent over the network unencrypted. Do it only when the server is running in a trusted environment!

Note: The ssh private key msut be provided on a sinlge line (json does not support multiline values), but the line ends must be preserved. A convenient method to generate this string in bash is:

```
sed -E ':a;N;$!ba;s/\r{0,1}\n/\\n/g' my-key.pem
```

The response will contain the `id` of the session created:

```
{
    "id": "58920c6c-c57c-4c55-a227-0ab1919e83be",
    [...]
}
```

Now we can create a new benchmarking test execution in the session (note that the id of the session is used in the request URL:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/
↪json' -d '{ \
    "tool": "idle", \
    "workload": "idle30" \
}' http://localhost:5000/api/v1/sessions/58920c6c-c57c-4c55-a227-0ab1919e83be/
↪executions/
```

The response will contain (along with other execution details) the `id` of the execution:

```
{
    "id": "253d9544-b3db-11e7-8bc2-742b62857160",
    [...]
}
```

With this execution id we can now invoke the *prepare* step that will create the resources on the provider, install the necessary tools and load the workloads:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/
↪json' http://localhost:5000/api/v1/executions/253d9544-b3db-11e7-8bc2-742b62857160/
↪prepare
```

Finally, we can invoke the *run* step:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/
↪json' http://localhost:5000/api/v1/executions/253d9544-b3db-11e7-8bc2-742b62857160/
↪run
```

The response of the *prepare* and *run* steps contain the start time and the duration of the operation:

```
{
    "started": "2017-10-18 08:18:33",
    "duration": "32.28253793716431"
}
```

The same session can be used to run multiple executions. At the end, the session and the resources created (e.g. VM) can be destroyed using the DELETE operation:

```
curl -X DELETE --header 'Accept: application/json' http://localhost:5000/api/v1/
↪sessions/58920c6c-c57c-4c55-a227-0ab1919e83be
```

2.6.2 Swagger Doc

This documentation is autogenerated from the Swagger API Specification using sphinx-swaggerdoc.

A better documentation for the REST API can be found directly in the REST Server:

1. Launch the server
2. Open http://localhost:5000/api/v1/

sessions

GET /sessions/

Parameters

Name	Position	Description	Type
X-Fields	header	An optional fields mask	string

POST /sessions/

Parameters

Name	Position	Description	Type
payload	body		
X-Fields	header	An optional fields mask	string

GET /sessions/{session_id}

Parameters

Name	Position	Description	Type
X-Fields	header	An optional fields mask	string
session_id	path	The id of the session	string

DELETE /sessions/{session_id}

Parameters

Name	Position	Description	Type
session_id	path	The id of the session	string

GET /sessions/{session_id}/executions/

Parameters

Name	Position	Description	Type
X-Fields	header	An optional fields mask	string
session_id	path		string

POST /sessions/{session_id}/executions/

Parameters

Name	Position	Description	Type
payload	body		
X-Fields	header	An optional fields mask	string
session_id	path		string

---

executions

GET /executions/

Parameters

Name	Position	Description	Type
X-Fields	header	An optional fields mask	string

POST /executions/{exec_id}/run

Parameters

Name	Position	Description	Type
X-Fields	header	An optional fields mask	string
exec_id	path		string

GET /executions/{exec_id}

Parameters

Name	Position	Description	Type
X-Fields	header	An optional fields mask	string
exec_id	path		string

POST /executions/{exec_id}/prepare

Parameters

Name	Position	Description	Type
X-Fields	header	An optional fields mask	string
exec_id	path		string

---

benchmarks

GET /benchmarks/

Parameters

Name	Position	Description	Type
X-Fields	header	An optional fields mask	string

GET /benchmarks/{benchmark_id}

Parameters

Name	Position	Description	Type
X-Fields	header	An optional fields mask	string
benchmark_id	path		string

---

providers

GET /providers/

Parameters

Name	Position	Description	Type
X-Fields	header	An optional fields mask	string

2.7 Docker

The Benchmarking Suite is also distributed in two different Docker containers. They are available at https://cloud.docker.com/app/benchsuite/repository/list.

2.7.1 benchsuite-multiexec

This container can be used to run benchmarks in batch mode.

Get (or update) the image with:

```
docker pull benchsuite/benchsuite-multiexec
```

Run the container binding the provider and storage (optional) configuration files stored in the local machine and passing the list of tests to execute as parameters (e.g. `idle:idle5`):

```
docker run -v /home/mypc/amazon.conf:/provider.conf -v /home/mypc/storage.conf:/
↪storage.conf benchsuite/benchsuite-multiexec:dev -p provider.conf -s centos_micro␣
↪idle:idle5
```

In case the storage service is running on the local machine, it could be necessary to use the `--net=host` option to reach it.

Alternatively, provider and storage configurations can be specified through environment variables: `BENCHSUITE_PROVIDER` and `BENCHSUITE_STORAGE_CONFIG` respectively.

```
docker run -e BENCHSUITE_PROVIDER="[myconf]...." -e BENCHSUITE_SERVICE_TYPE="centos_
↪micro" -v /home/mypc/storage.conf:/storage.conf benchsuite/benchsuite-multiexec:dev␣
↪idle:idle5
```

2.7.2 benchsuite-rest-service

This image contains the Benchmarking Suite REST SERVER (see rest-server-doc section). When started, the container exposes the REST service on port 5000.

To run the container, just use the Docker CLI:

```
docker run benchsuite/benchsuite-rest-server
```

The service reads the Benchmarking Suite bs-configuration from the `/` directory of the container. For instance, to provide a configuration for the storage (to persist results in the db) mount a file in the container named `/storage.conf` or `/storage.json`:

```
docker run -v my-storage.conf:/storage.conf benchsuite/benchsuite-rest-server
```

Also providers configuration files can be mounted in the container in the same way:

```
docker run -v my-provider.conf:/providers/my-provider.conf benchsuite/benchsuite-rest-
↪server
```

2.8 Scheduler

The Benchsuite Scheduler allows to schedule the execution of benchmarking tests at pre-fixed intervals. It needs:

- a MongoDB instance to load the **schedules** (see below), keep its state, log the executions and save the results
- a Docker Swarm instance to launch the tests (the *benchsuite-multiexec* Docker image is used) and to run the scheduler itself

The scheduler works in this way:

1. loads from a MongoDB collection the schedules and creates a job for each schedule (it uses APScheduler[1] under the hood). The jobs are kept in sync and refreshed periodically
2. sets-up a timer for each job accordingly with the time interval defined in the schedule

[1] https://apscheduler.readthedocs.io/en/latest/

3. when its the time to execute a job, launchs a *benchsuite-multiexec* and configure it to execute the needed tests and to store the results on another MongoDB collection

2.8.1 Schedules

The **schedules** are the main input to the scheduler and models the tests that needs to be scheduled. Each schedule contains two types of information: the parameters of the tests and the timing information. Each schedule is expected to be a MongoDB document with this structure:

```
{
    "id" : "ec2-123asd-filebench",
    "active": true,
    "provider_config_secret" : "ec2",
    "username" : "ggiammat",
    "tests" : [
        "filebench",
        "ycsb-mysql",
        "dacapo"
    ],
    "tags" : [
        "scheduled"
    ],
    "properties" : {
        "prop1" : "val1",
        "prop2" : "prop2"
    },
    "env" : {
        "MYVAR": "val"
    },
    "interval" : {
        "hours" : 1
    },
    "benchsuite_additional_opts": ["-v", "--another-opt],
    "docker_additional_opts": {
        "hosts": {"myhost":"10.1.0.1"}
    }
}
```

It contains:

- `id`: a unique id
- `active`: defined whether this schedule should be considered by the scheduler or not
- `provider_config_secret`: the name (or the id) of the Docker secret that contains the Cloud Provider configuration. It uses the Docker secrets because the configuration also contains the user credentials to access the Cloud Provider
- `username`: an identifier of the user that is requesting the execution. It will be saved also in the benchmarking results
- `tests`: a list of test names to execute to be passed to the benchsuite `multiexec` command (see *Command Line Tool Documentation*)
- `tags`: a list of tags to assign to the results
- `properties`: a list of key-value properties that will be assigned to the results
- `env`: key-value pairs that define enviornment variables to be available in the execution environment during the execution

- `interval`: the time interval between two executions. The accepted keys are: `weeks`, `days`, `hours`, `minutes` and `seconds`. Multiple keys can be combined and if not specified, the default value is 0
- `benchsuite_additional_opts`: a list of string that will be appended to the benchsuite-multiexec command line
- `docker_additional_opts`: a dictionary of additional options to use when creating new Docker services (see DockerCreateServiceReference[2] for a reference of available options)

New schedules are automatically loaded and they are rescheduled if a change is detected.

2.8.2 Configuration

The schduler accepts multiple parameters. Some of them are mandatory, while some other have a default value.

All the parameters can be specified in a config file in the format

```
PARAM1=val1
PARAM2=val2
...
```

or specified as environment variable (the latter overrides the former).

The list of mandatory parameters are:

- `DB_HOST`: the connection string to the MongDB (e.g. "mongodb://localhost:27017"). It can be omitted only if the `SCHEDULES_DB_HOST`, `JOBS_DB_HOST` and `EXEC_DB_HOSTS` are provided
- `DOCKER_STORAGE_SECRET`: the name of the secret that contains the Benchsuite Storage configuration (used to store results of the tests)

The optional parameters (or the ones that have a default value) are:

- `SCHEDULES_SYNC_INTERVAL` (default: 60): it the number of seconds between two refresh of the schedules in the MongoDB collection
- `SCHEDULES_JOBS_PRINT_INTERVAL` (default: 60): interval time in seconds to print on the console a report of the scheduled and running jobs
- `DB_NAME` (default: "benchmarking"): the name of the MongoDB database to use
- `SCHEDULES_DB_HOST`: if set, overrides the `DB_HOST` value for the MongoDB instance used to load the schedules
- `SCHEDULES_DB_NAME`: if set, overrides the `DB_NAME` value for the database used to load the schedules
- `SCHEDULES_DB_COLLECTION` (default: "scheduling"): the name of the collection that contains the schedules
- `JOBS_DB_HOST`: if set, overrides the `DB_HOST` value for the MongoDB instance used to store the internal state of the scheduler
- `JOBS_DB_NAME`: if set, overrides the `DB_NAME` value for the database used to store the internal state of the scheduler
- `JOBS_DB_COLLECTION` (default: "_apjobs"): the name of the collection that contains the internal state of the scheduler
- `EXEC_DB_HOST`: if set, overrides the ‘‘DB_HOST‘ value for the MongoDB instance used to log the executions
- `EXEC_DB_NAME`: if set, overrides the `DB_NAME` value for the database used to log the executions

[2] https://docker-py.readthedocs.io/en/stable/services.html

- EXEC_DB_COLLECTION (default: "_apexec"): the name of the collection that contains the logs of the executions
- DOCKER_HOST (default: "localhost:2375"): the host and port of the Docker Swarm instance (used to create containers though the Docker API)
- DOCKER_BENCHSUITE_IMAGE (default: "benchsuite/benchsuite-multiexec"): the name of the benchsuite-multiexec image to use
- DOCKER_GLOBAL_ENV: a comma separated list of environment variables that will be set in the benchsuite-multiexec container (e.g. "VAR1=val1,var_2=val2"). Useful to set the an http proxy if necessary. Use ',' to insert a comma in the variables names or values.
- BENCHSUITE_GLOBAL_TAGS: a comma separated list of string that will be set as tags in the benchmarking results (e.g. "test1,scheduled,automatic")
- DOCKER_ADDITIONAL_OPTS: a comma separated list of options in the format 'KEY=VAL' that will be added to the Docker service create invocation. VAL is evaluated using json.loads() function. See DockerCreateServiceReference[2] for a reference of available options (e.g. 'hosts={"myhost":"10.1.0.1"}')
- BENCHSUITE_ADDITIONAL_OPTS: additional options that will be set on the benchsuite-multiexec command line (e.g. "-vvv –failonerror")

2.8.3 Benchsuite Scheduler Docker image

The simplest way to run the Benchsuite Scheduler is to run the benchsuite/benchsuite-scheduler Docker image specifying the configuration parameters as envrionment variables:

```
docker run -e DB_HOST=mongodb://172.17.0.1:27017/ -e DOCKER_STORAGE_SECRET=storage -e␣
↪DOCKER_HOST=172.17.0.1:2375 benchsuite/benchsuite-scheduler
```

Alternatively, the configuration can be specified in the /tmp/config file.

```
docker run -v /home/mypc/scheduler.conf:/tmp/config benchsuite/benchsuite-scheduler
```

The two approaches can be also be mixed.

2.9 API Reference

class benchsuite.core.controller.**BenchmarkingController**(*config_folder=None*, *storage_config_file=None*)

The facade to all Benchmarking Suite operations

class benchsuite.core.model.benchmark.**Benchmark**(*tool_id*, *workload_id*, *tool_name*, *workload_name*, *workload_categories*, *workload_description*)

A Benchmark

2.10 Changelog

This Changelog reports the main changes occuring in the Benchmarking Suite. The versions of the Benchmarking Suite (also called Milestones) refers to the versions of the Docker containers and the Documentation, while the versions of the single modules are reported in each entry of the changelog.

The *Unreleased* section contains changes already released in the Python modules, but not yet included in any Milestone.

2.10.1 Unreleased

2.10.2 Benchmarking Suite v. 3.1.0

Release date: 2019-01-17

- [stdlib-2.7.0] reduced number of testing queries for YCSB benchmarks (reducing the overall duration of the tests)
- [stdlib-2.7.0] added Ubuntu 14 compatibility for Web Framework benchmark
- [stdlib-2.7.0] more robust ssh connections (retry on network failure for a fixed number of times)
- [stdlib-2.7.0] new version of Paramiko ssh library that brings security patches
- [stdlib-2.7.0] increasing waiting time in YCSB test for MongoDB to start
- [stdlib-2.7.0] implemented multi-node benchmark tests with custom number and names of nodes
- [stdlib-2.7.0] added IPerf benchmark with various pre-defined workloads
- [stdlib-2.7.0] added Sysbench benchmark with various pre-defined workloads
- [core-2.6.0] added option to keep the environment after a multiexec execution
- [core-2.6.0] added option in multiexec mode to retry a failed test
- [core-2.6.0] added option to retry (–max-retry) tests if they fail
- [core-2.6.0] improved handling of logs in multi-node tests
- [core-2.6.0] improved execution of clean-up scripts
- [scheduler-1.5.0] improved logging of executions
- [scheduler-1.5.0] added debug option that not delete containers after they end
- [backends-2.5.0] updated result records schema to version 2: include all logs of multi-node tests
- [cli-2.3.0] improved log messages

2.10.3 Benchmarking Suite v. 3.0.0

Release date: 2018-08-08

- [stdlib-2.6.0] added Web Frameworks Benchmarking tool
- [stdlib-2.6.0] added category and keywords for each workload
- [stdlib-2.6.0] auto-discovery (when possible) of networks, security groups and platforms
- [stdlib-2.6.0] add async executions to avoid timeout exceptions
- [stdlib-2.6.0] support creation of key pairs if not provided in the configuration
- [scheduler-1.4.0] updated to docker-py version 3.0.0
- [scheduler-1.4.0] improved logging of exceptions
- [scheduler-1.4.0] added “properties” field in schedules to store custom data in generated results
- [cli-2.2.0] introduced autocomplete for some commands

- [backends-2.4.0] store execution errors and logs in the backend
- [backends-2.4.0] changed results schema
- [core-2.5.0] fixed crash if the storage was not properly configured
- [core-2.5.0] allowed to use wildecards in workload names in multiexec mode

2.10.4 Benchmarking Suite v. 2.7.0

Release date: 2018-02-14

- [rest-2.3.0] added options to listen on specific host and port
- [sdtlib-2.5.0] customizable retries time for connection to new VMs
- [stdlib-2.5.0] delete the VMs created in case of an unhandled exception during the creation
- [stdlib-2.5.0] fixed empty values in configuration parsing
- [core-2.4.0] added possibility to use custom sessions storage file
- [scheduler-1.3.1] fixed invalid characters in containers name

2.10.5 Benchmarking Suite v. 2.6.1

Release date: 2018-01-22

- [stdlib-2.4.3] support for 'auth_url', 'auth_version' and 'region' provider config parameters

2.10.6 Benchmarking Suite v. 2.6.0

Release date: 2018-01-16

- [scheduler-1.3.0] added configuration parameters to add additional Docker options to the containers created by the scheduler

2.10.7 Benchmarking Suite v. 2.5.1

Release date: 2018-01-15

- fixed the URL to download the DaCapo benchmark

2.10.8 Benchmarking Suite v. 2.5.0

Release date: 2017-12-18

- [backends-2.3.0] MongoDB - storing start time as date object (previously it was a timestamp)
- [scheduler-1.2.0] Support for using Docker unix socket instead of the tcp port
- [core-2.3.1] Fixed DEFAULT section not read in the Json configuration files
- [stdlib-2.4.1] Fixed serialization issue of the LibcloudComputeProvider objects
- [cli-2.1.2] Improvements and fixes to the "shell" command

2.10.9 Benchmarking Suite v. 2.4.0

Release date: 2017-12-04

- [stdlib-2.4.0] randomize names of VMs created by the Benchmarking Suite
- [stdlib-2.4.0] set security groups in openstack
- [scheduler-1.1.0] add config parameter to add global env, tags and additional params
- [scheduler-1.1.0] added the "active" parameter in the schedules
- [core-2.3.0, backends-2.2.0] added storage of execution errors in the database

2.10.10 Benchmarking Suite v. 2.3.1

Release date: 2017-11-21

- [core-2.2.4] fixed support for the –failonerror parameter from the command line

2.10.11 Benchmarking Suite v. 2.3.0

Release date: 2017-11-20

- [core-2.2.2] considering only providers configuration files with extension .json and .conf
- [core-2.2.3] duration is now considered as a metric
- [stdlib-2.3.0] metrics renamed to make them coherent in different tests
- [stdlib-2.3.0] added multiple workloads in the CFD benchmark
- [cli-2.1.1] added –failonerror for the multiexec command. The option allows to not continue with next test if the current one fails
- [scheduler-1.0.0] first release of the Benchsuite Scheduler

2.10.12 Benchmarking Suite v. 2.2.2

Release date: 2017-10-20

This patch release fixes some minor bugs found in the code:

- fixed creation of new sessions if the provider configuration is in json format
- fixed default error handling in the REST server (now the full exception message - and not only "Internal Server Error" is sent back to the caller)
- fixed parsing of "network" and "security_group" parameters: now they can be either the id or the name of the object
- fixed crash of some Filebench workloads on Amazon EC2 using the micro instances

2.10.13 Benchmarking Suite v. 2.2.1

Release date: 2017-10-18

This patch release fixes an outdated information in the REST server documentation page

2.10.14 Benchmarking Suite v. 2.2.0

Release date: 2017-10-18

This minor release introduces following improvements:

- support for json configuration files (only for providers and storage at the moment)
- better handling of network configuration parameters in the provider configuration

2.10.15 Benchmarking Suite v. 2.1.0

Release date: 2017-10-13

This minor release introduces some new functionalities and improvement to the tool:

- support for MongoDB backend
- list of available benchmarks and cloud providers (in Cli and REST)
- field "name" in workload sections in configuration files
- return node_id (in case of OpenStack) in the REST calls
- accept provider configuration as string parameter
- add tags to sessions/executions (e.g. for the user-id in the QET)
- provider and storage configurations can be also specified via command line or environment variable
- improvement and tuning of YCSB, Filebench and DaCapo benchmarks

2.10.16 Benchmarking Suite v. 2.0.0

Release date: 2017-08-01

This is a major release version of the Benchmarking Suite that introduces several changes and improvements with respect to the Benchmarking Suite 1.x versions.

In the Core library:

- a complete refactoring of the code to improve the parameterization and modularization
- introduction of benchmarking sessions

In the StdLib library:

- **for Benchmarks:**
  - NEW CFD Benchmark
  - Updated Filebench and YCSB tools versions
- **for Cloud Providers:**
  - NEW FIWARE FILAB connector
  - Updated Amazon EC2 to work with VPCs

The Cli and REST modules are completely new and the previous implmentation have been abandoned.

2.11 Development

This section explains the development, integration and distritbuion process of the Benchmarking Suite. Intended readers are developers.

2.11.1 Continuous Integration

TBD

2.11.2 Release Steps

Checklist to release the Benchmarking Suite

1. Commit any not-committed file on the workspace
2. Identify whcih modules needs to be released (see commits since the latest release, see the changelog)
3. Update the changelog file if not done (use messages in the commits as reference)

Modules Release

For each module to release:

1. increase the version number in the `__init__py` file
2. create the source distribution package and upload on PYPI Testing (remove the `-r pypitest` to upload on the official PYPI)

```
python setup.py sdist upload -r pypitest
```

3. to test the release from PYPI test:

```
# create a new virtual env
virtualenv -p /usr/bin/python3.5 venvXX

# activate the virtualenv
source venvXX/bin/activate


# install the modules to test
pip install -v -i https://testpypi.python.org/pypi --extra-index-url https://pypi.
↪python.org/simple/ -U benchsuite.core
```

4. upload the distribution packages on PYPI

```
python setup.py sdist upload
```

3. commit and push everything on GitHub
4. create a release on GitHub (this will also create a tag)

Milestone Release

1. Check all the modules and the versions that will be included. Release modules if necessary
1. In `benchsuite-docs`, update the version in conf.py
2. Update the changelog.rst with the changelog for this milestone
6. Commit the documentation on GitHub and create a tag. This will also create a new tag in readthedocs
7. Update the ".release" Dockerfiles in benchsuite-docker project
8. Commit benchsuite-docker and create a new tag. This will trigger the creation of a new tag for docker images

2.11.3 Documentation

Documentation is automatically built on ReadTheDocs at every commit

2.11.4 Docker

Docker containers are built automatically from Dockerfiles located in the benchsuite-docker repository.

To create a new tag of Docker images, create a tag in the Git repository that starts with "v" (e.g. "v2.0", "v1.2.3", "v1.2.3-beta1", ...)

2.12 FAQs

2.12.1 How to clear all stored benchmarking sessions?

Sessions are stored in a file in `~/.local/share/benchmarking-suite/sessions.dat`. Deleting that file, all sessions will be removed. This should be an extreme solution, the more correct way to delete a session is to use the destroy-session command.

2.12.2 How to clear all stored executions?

Executions are stored along with the sessions. See previous question: *How to clear all stored benchmarking sessions?*.

CHAPTER 3

Contacts

Main contact person for the Benchmarking Suite is:

Person Gabriele Giammatteo

Company Research and Development Laboratory Engineering Ingegneria Informatica S.p.A.

Address via Riccardo Morandi, 32 00148 Rome, Italy

e-mail gabriele.giammatteo@eng.it

For bugs, features and other code-related requests the issue tracker can be used at: https://github.com/benchmarking-suite/benchsuite-core/issues

CHAPTER 4

References

Python Module Index

b

Index

B