SIC tests Protocol

Legacy Reference protocol

Service Introduction Center

Version 1.0, 09-Jun-2016

Table of Content

Copy to	N/A
Contact	Service Introduction Center
Project Name & Version	N/A
Writer	Arnauld Van Muysewinkel <Arnauld.VanMuysewinkel@smals.be> (AVM)

Version	Status	Date	Author	Nature of modifications	File location
0.1	WiP	02,03-Jun-2016	AVM	Initial version	https://git01.smals.be/sic/tools.protocols
0.2	WiP	06-Jun-2016	AVM	After review by NIN	
1.0	Final	09-Jun-2016	AVM	Finalize after team meeting	

1. Scope

This document describes the default SIC tests protocol applicable up to June 2016.

Most standard tests mission (typical web applications) conducted during this period follow more or less this protocol, even though some variations are possible, depending on the tester and on the specificities of the mission.

The document can be considered:

- either as restrospective documentation of the protocol for past tests (without detailed protocol description)
- as default protocol for new *simple* tests (i.e. matching completely this protocol)
- as a template for creating new protocols (all other cases)

In the latter case, the protocol will be maintained in the code repository of the SIC project (location *protocol/src/main/adoc/protocol.adoc*) and published on SIC intranext, together with the Mission Sheet and Report.

2. Environment

Components	Environment	Tenant
test unit	SIC (OVM)	N/A

Components	Environment	Tenant
database	SIC (SCAN)	N/A
ESB	SIC (legacy)	N/A
base WS	SIC (OVM)	N/A
specific WS	ACCP or INTEG	N/A

3. Scenarios

S1:main

- *Objective:* Verify that the performance of the web application is compatible with interactive usage, and that it is stable
- *Configurations:* [C1:WebApp]
- *Load profiles:* [P1:reference], [P2:loadx1], [P3:loadx2], [P4:loadx4], [P5:stress], [P6:peak] (optional)
- *Sequence:* for each virtual user
  - user login
  - in a loop: page 1, page 2 … page *n*
  - user logout / session destroy (when possible)
- *Expected observations*
  - [P2:loadx1]: performances consistent with [SLA]
  - [P2:loadx1], [P3:loadx2], [P4:loadx4], [P6:peak]: moderate threads and CPU usage
  - [P5:stress], [P6:peak]: no memory leak
  - [P5:stress], [P6:peak]: constant response times

Sn:other

Other ad-hoc scenario if required

4. Load profiles

(See also: https://fr.wikipedia.org/wiki/Test_de_performance#Types_de_Tests)

P1:reference

- *duration:* 15'
- *threads:* constant, 1
- *virtual users:* constant, 1 (1 per thread)
- *sessions:* constant, 1 (1 per virtual user)

- *throughput:* free (no delay between requests)

P2:loadx1

- *duration:* 30'
- *threads:* 5' rampup from 0 to [REF_USERS], then constant
- *virtual users:* 5' rampup from 0 to [REF_USERS], then constant (1 per thread)
- *sessions:* 5' rampup from 0 to [REF_USERS], then constant (1 per virtual user)
- *throughput:* 5' rampup from 0 to [REF_THROUGHPUT], then constant, through auto adjusting delay between requests

P3:loadx2

- *duration:* 30'
- *threads:* 5' rampup from 0 to 2*[REF_USERS], then constant
- *virtual users:* 5' rampup from 0 to 2*[REF_USERS], then constant (1 per thread)
- *sessions:* 5' rampup from 0 to 2*[REF_USERS], then constant (1 per virtual user)
- *throughput:* 5' rampup from 0 to 2*[REF_THROUGHPUT], then constant, through auto adjusting delay between requests

P4:loadx4

- *duration:* 30'
- *threads:* 5' rampup from 0 to 4*[REF_USERS], then constant
- *virtual users:* 5' rampup from 0 to 4*[REF_USERS], then constant (1 per thread)
- *sessions:* 5' rampup from 0 to 4*[REF_USERS], then constant (1 per virtual user)
- *throughput:* 5' rampup from 0 to 4*[REF_THROUGHPUT], then constant, through auto adjusting delay between requests

P5:stress

- *duration:* 60'
- *threads:* 5' rampup from 0 to [STRESS_USERS], then constant
- *virtual users:* 5' rampup from 0 to [STRESS_USERS], then constant (1 per thread)
- *sessions:* 5' rampup from 0 to [STRESS_USERS], then constant (1 per virtual user)
- *throughput:* free (no delay between requests)

P6:peak (optional)

- *duration:* 15'
- *threads:* 1' rampup from 0 to [PEAK_USERS], then constant
- *virtual users:* 1' rampup from 0 to [PEAK_USERS], then constant (1 per thread)
- *sessions:* 1' rampup from 0 to [PEAK_USERS], then constant (1 per virtual user)
- *throughput:* 1' rampup from 0 to [PEAK_THROUGHPUT], then constant, through auto adjusting delay between requests

5. Configurations

C1:WebApp

typical *bubble* reference architecture for the test unit

- 3 physical OVM machine running 1 or 2 VM each
- on otextvms001b: ltextsic100: 1 reverse proxy running httpd
- on otextvms001b: ltextsic*XXX*: 1 "admin" VM running 1 WebLo admin server for 1 domain "*bubble*domain"
- on otextvms001a: ltextsic*YYY*: 1 "node1" VM running 2 WebLo servers: "*bubble*Sync1" and "*bubble*Async1"
- on otextvms001c: ltextsic*ZZZ*: 1 "node2" VM running 2 WebLo servers: "*bubble*Sync2" and "*bubble*Async2"
- 2 clusters defined in domain "*bubble*domain":
  - "*bubble*domainSync": containing nodes "*bubble*Sync1" and "*bubble*Sync2"
  - "*bubble*domainAsync": containing nodes "*bubble*Async1" and "*bubble*Async2"
- schema "*BUBBLE_SCHEMA*" in SIC database, using user "*BUBBLE_ADM*"

  Dependencies:

- 3 VMWare VM running SIC ESB domain "sicesbsocsecdomain": ltextweb002a (admin), laextapp016a (node1), laextapp016b (node2)
- base services running on 3 physical OVM machines (...)

Injection infrastructure:

+

- 1 physical distinct OVM machine (otextvms001d) running 1 "Jenkins" VM: ltextsic101
- 1 or 2 physical server(s) (laextapp002a and/or laextapp003a) running 1 or more instance(s) of jmeter (*2.13*) each (the total number of injectors depending on the load required)

Cn:other

Other ad-hoc configuration if required

6. Parameters

Some tests parameters should be given as part of the test requirements (typically in a mission sheet). Others need to be fine tuned as part of the tests preparation.

6.1. Given

According to mission sheet:

REF_THROUGHPUT

Nominal throughput (# of requests per unit of time).
Defined as: the average throughput during one hour, measured during the hour with the highest load (consider a meaningful period for the statistics : day / week / month / trimestre, depending on the periodicity of your business)

PEAK_THROUGHPUT

Peak throughput.
Same as above, but for a period of 5'

In some circumstances, the project team is not able to produce figures for [REF_THROUGHPUT] or [PEAK_THROUGHPUT] (for example a new application for which there are no statistics from production).

In this case, these figures must be approximated based on asumptions from the projects team, like: max concurrent users, most loaded business hours, volume per month, etc.

SLA

performance objective should be given in the Mission Sheet, typical values are given in the below table

Client	Type	Percentage	Duration
ONSS Portal	WebApp	95%	4s
	WS	95%	4s
	SOA	95%	2s
OrliPro - Intramuros	WebApp	95%	4s
	WS	95%	4s
	SOA	95%	2s
eHealth Core	WebApp	95%	4s
	WS	98%	1s
VAS	WebApp	95%	4s
	WS	95%	1-4s (?)

— Luc Vandam, eMail sent on 2016-06-08 12:50

6.2. Calibration

Before final measures, we must determine:

REF_USERS
: number of virtual users required to reach [REF_THROUGHPUT]

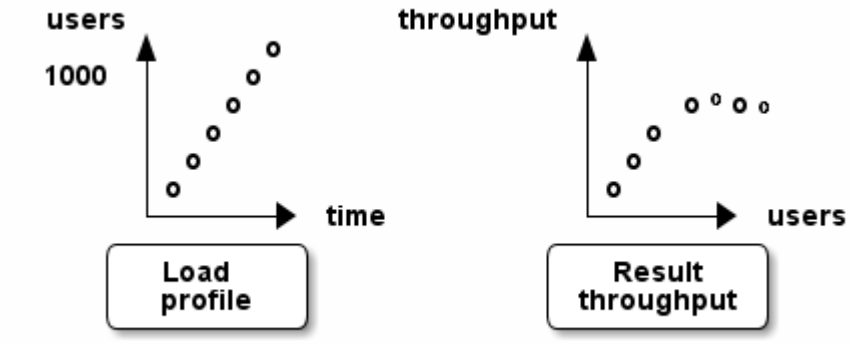
PEAK_USERS
: number of virtual users required to reach [PEAK_THROUGHPUT]

STRESS_USERS
: number of virtual users required to saturate the server (i.e. such that the throughput cannot be higher)

Actual values of these parameters must be documented in the tests report.

These values may be determined by convention: use typical values and make sure the throughput setpoint is reached during the actual test. However, this gives no guarantee that the saturation point is reached for the stress test.

Another method is to build a load profile consisting of a long rampup for the number of users, up to a very big count. This profile is injected in the system, and the resulting throughput is measured. It is then possible to determine the minimum number of virtual users required to reach a given throughput. Also there will be an inflexion point in the throughput graph, when the saturation point is reached.



7. Metrics

The following data will be recorded during each test run:

- on each node ("node1", "node2"): **vmstat** every 5 seconds, with timestamp
- for each weblogic server ("*bubble*Sync1", "*bubble*Sync2", "*bubble*Async1", "*bubble*Async2"): **GC log**
- on the injector(s): **http requests** (label, timestamp (ms), duration (ms), status code)

Based on these data, the following metrics will be calculated (for each test run):

vmstat

- run queue size over time
- CPU usage kind over time, by kind (user, kernel, io wait, stolen…)
- ratio kernel time / user time over time
- io activity over time
- context switches rate over time
- interrupts rate over time

GC log

- liveset over time (i.e. used heap size after each full (or old) GC)
- linear regression of the live set → live set slope
- ratio GC time / total time, over time
- memory allocation rate over time
- GC frequency over time, by kind (young, full/old)

http requests

for each label (and only for succesful requests)

- overall response times statistics (min, mean, median, pct90, pct95, pct98, max) during test window (i.e. rampup period excluded)
- linear regression of the response time → response time slope
- overall throughput during test window, by status (success, error)
- overall throughput statistics (mean, pct90, max)
- overall ratio error/success
- response times distribution
- dispersion ("cloud") graph, over time
- response time statistics, over time (pct90, mean, median, minimum)
- effective throughput over time