# Scenario Care Load Testing in TTCN-3

#### Ji Wu BeiHang University, China

# Agenda

- Load testing in TTCN-3
- Load profile model
- Load control
- Test System Framework
- Virtual user Implementation
- Reuse existing test case
- Experiments
- Conclusion

## Structure of Load testing

 Load testing is an important way to explore SUT load and the bottleneck.



# Load Testing in TTCN-3

- Load testing in TTCN-3
  - Test cases/functions for implementing load in the name of virtual users
  - Controller is part of load strategy
- Concerns on load testing
  - More virtual users
  - More varieties of user behaviors
  - More realistic load profile
- Need piles of TTCN-3 code to implement the concerns!

# Load Testing in TTCN-3

- Load testing usually goes after successful functionality testing
- The existing test cases can guarantee
  - The varieties of behaviors
  - The realistic of load profile
- Two issues come
  - How to reuse the existing test cases?
  - How to coordinate the VUs with the test cases to get good load profile?

# Load Testing in TTCN-3

- How to reuse the existing test cases?
   a question of code conversion and generation
- How to coordinate the VUs ...?
  - A question of adaptive control
    - assign test cases for VUs
    - observe load profile
    - evaluate against given parameters
    - modify the assignment

#### Load Profile Model

Several relative metrics of load => clusters



## **Cluster Domination Equations**

session duration	= request interval within session	*	session length
request interval overall	= request interval within session	/	number of concurrent users
number of concurrent users	= session duration	/	session interval
session interval	= overall request interval	*	session length
request interval within session	= thinking time	+	response time

## Load Profile Control Points

- Five control points in two stages of control
  - Static control (pre-testing): proportions of various VUs → controlling session length, requests distri., byte distri.
  - Dynamic control (within-testing): #concurrent users, session interval, overall request interval, thinking time



#### Static Load Control

- Each kind of VU has pre-determined sequence of requests
- Given the proportions of various requests(PVR), to setup the number of different kinds of VUs



# **Dynamic Load Control**

• We design seven ways to control the load dynamically.

Mode	Points to Control				
1	Thinking time (distribution)				
	#concurrent users (value)				
2	Thinking time (distribution)				
	Overall request interval (value)				
3	Thinking time (distribution)				
	Session interval (distribution)				
4	Overall request interval (distribution)				
5	Thinking time (distribution)				
6	#concurrent users (value)				
7	Session interval (distribution)				

# Performance Testing Scenario



- Provides phase-based load profile and control chance
- Provides control on the types of virtual users
- It can construct complex test scenario with required load profile, and simulate the real usages.

# Test System Framework

- SA Delegate: Manage the communication connections to the SUT
- DMP-PA: Manages the Dynamic Module Parameters (DMP)
- TL: Manages virtual users, Monitors the test running, measures performance



# Virtual User Implementation

- Two levels of concurrency
  - Concurrency among virtual users
  - Concurrency within a virtual user
- Three levels of test components
  - MTC: manages the concurrency of virtual users
  - V-PTC: manages the concurrency within a virtual user, maintains a unique sessionID
  - PTC: manages the communication with other test component, maintains a unique componentID



#### Virtual User Implementation



# Load Control Implementation

- Load control is part of test system behavior.
  - A top level test case, keeps on running until test finished
- It monitors the relative ports to receive load control instruction
  - Read load parameters from DMP-PA
  - Create/Stop V-PTC
  - Start ready V-PTC with associated session
  - Control the V-PTC start interval
  - Control the number of live V-PTC

# Monitoring of Virtual User Behavior

- Record the time to send and receive messages within a session
- Save as log file
- Calculate the load metrics from the log
- As feedback for adaptive load control



# Performance Test Case

- Similarities with functionality test case
  - Similar test behavior (send/receive)
  - Use the same codec and adapter
- Differences from functionality test case
  - More powerful test configuration (three levels of test components)
  - Add the session hierarchy (runs on V-PTC)
  - Need dynamic module parameters management
- Automatically rewrite the test behavior in functionality test case

#### **Conversion Rules**

#### Downgrade functionality test case

*testcase* <*tc\_name*>() *runs on compType\_name1* {[*statements*]} →

function subst\_<tc\_name>(charstring sessionID) runs on compType\_name1{[statements]}

- Change references of *mtc* to *v-ptc*
- Get module parameter value from DMP-PA

statementWith(ModuleParameter); >

var ModuleParaType mp\_subst; mp\_subst:=getMpSubst(sessionID); statementWith(ModuleParameter);

• getMpSubst is an external function implemented in DMP-PA

#### **Conversion Implementation**



\*Parsing with TRex.

Load testing for a simple chat system



Functionality test case

• Load test case after automatic convert





- SUT is a web-based bookstore

   JSP for user interface and business logic
   MySOL for data management and access
  - MySQL for data management and access



- 11 functionality test cases cover the flows
- Use our own distribution platform to support the test



stcases								
▲ tc	(11)		_					
		= name	= len	= mixTeight				
	1	tc001	15	1				
	2	tc002	25	1				
	3	tc003	11	1				
	4	tc004	15	1				
	5	tc005	19	1				
	6	tc006	12	1				
	7	tc007	13	1				
	8	tc_nologin_003	7	1				
	9	tc_nologin_004	11	1				
	10	tc_nologin_005	15	1				
	11	tc_nologin_006	8	1				
npai gn								
🔺 pha	<b>se</b> (1	1)						
		= dur	() ses	sionRate		() concurrent¥V	() globalReqRate	() thinkingTime
	1	60	▲ ses	sionRate		3	🔺 globalReqRate	thinkingTime
				= name	null		📃 \Xi name null	<b>= name</b> aver
								📕 () para 0.1
	2	60	▲ ses	sionRate		12	🔺 globalReqRate	thinkingTime
				= name	null		📃 🚍 name null	= name aver
								📕 🌔 para 0.1
	3	60	▲ ses	sionRate		21	globalReqRate	thinkingTime
				= name	null		📃 🚍 name null	= name aver
								📕 🌔 para 0.1
	4	60	▲ ses	sionRate		30	🔺 globalReqRate	🔺 thinkingTime
				= name	null		<b>= name</b> null	<b>= name</b> aver
								🌔 🌔 para 0.1
	5	60	▲ ses	sionRate		39	🔺 globalReqRate	🔺 thinkingTime
				= name	null		<b>= name</b> null	<b>= name</b> aver
								() para 0.1
	6	60	ses 🔺	sionRate		48	🔺 globalReqRate	🔺 thinkingTime
				= name	null		<b>= name</b> null	<b>= name</b> aver
								() para 0.1
	7	60	▲ ses	sionRate		51	🔺 globalReqRate	🔺 thinkingTime
				🚍 name	null		<b>= name</b> null	<b>= name</b> aver
								() para 0.1
	8	60	ses:	sionRate		54	🔺 globalReqRate	thinkingTime
				= name	null		<b>name</b> null	= name aver
			_					() para 0.1
	9	60	ses:	sionRate		63	<b>∡ globalRegRate</b>	ThinkingTime
	_			= name	null		= name null	= name over
			_					
	10	60	ASCE	sionRate		66	🗖 globalRegRate	ThinkingTime
	10		- 363		<b>nu</b> ]]	~~	- nema null	- nama away
			_	— паше	narr			A name aver
	11	60		rionRota		69	alahal ResPote	A this bis a time.
	11	00	a ses		17	05	E RODALNEQUATE	Ininkingline
			-	= name	null		<b>name</b> null	= name aver
			1					🔲 🌔 para U.1

#### Experiment B performance\_measurement **= testID** 1229173581578 **phase** (11) sessionID ()load () output 1 ExeSes06093573796 🔺 load 🔺 output () session\_interval 683 () response time 0.039 () concurrent vu 2.98322857927647 () throughout 19.340453377712432 19.388925691942287 () request\_rate () pass\_rate 1.0 () thinking\_time 112 ExeSes06093573796 🔺 output () response\_time 0.19 () throughout 37.04242424242424 060606 1.2 6363635 () pass\_rate 1.0 文件夹 🔺 output 🗄 🛅 results ()response time 0.375 074586 () throughout 40. 25509460837474 1 in 🔁 🛅 122917358157 175653 () pass\_rate 1.0 🗄 🛅 ExeSes060 🔺 output () response time 0.568 🖻 📄 ExeSe: 0.8 () throughout 33.71984109040406 0467687 🦳 loc 794774 () pass\_rate 1.0 🗄 📄 ExeSes160 🔺 output 🗄 🛅 ExeSes160 () response time 0.881 0.6 () throughout 36. 5045852337744 072268 🗄 🦳 ExeSes160 4048064 () pass\_rate 1.0 🗄 🦳 ExeSes160 a output () response time 2.157 🗄 🛅 ExeSes160 0.4 158998 () throughout 19.801980198019802 🗄 🛅 ExeSes160 0.886363636363636364 1461854 () pass\_rate 🗄 🛅 ExeSes160 antnut 0.2 () response\_time 2.272 🗄 🛅 ExeSes160 () throughout 18. 2303030303030303 272727 🗄 🛅 ExeSes160 848485 0.8604651162790697 () pass\_rate 🗄 🦳 ExeSes160 🔺 output 0 () response\_time 1.902 () throughout 5. 509198105609072 694127 20 30 40 50 10 60 70 80 413608 () pass\_rate 0.0 🔺 output -0.2 () response\_time 2.024 6171314 () throughout 2. 277058234553148 52365 () pass\_rate 0.0 \_\_\_\_resp. time (2.5 second) throughput (45/second) succ rate 1-11-11 10 Exeses00033313130 🔺 1080 output () session\_interval 643 () response\_time 0.984 () throughout 0.04826487764853516 () concurrent\_vu 65. 97499879337806 () request\_rate 4.536898498962305 () pass\_rate 0.0 () thinking\_time 172 11 ExeSes06093573796 🔺 load output () session\_interval 553 () response\_time null () concurrent\_vu 68. 96623957043873 () throughout 0.0 () request rate 4.195939102615629 () pass\_rate 0.0 () thinking time null

# Conclusion

- This work is supported by 2 National High-Tech program projects.
- We develop the distributed load testing in TTCN-3
  - Profile based load generation
  - Adaptive load control
  - Reuse existing functionality test cases
- TTCN-3 language provides great flexibility and good structure for load testing.

# Questions?

• Thanks for your attention!