SONY
make.believe

Functional testing with load

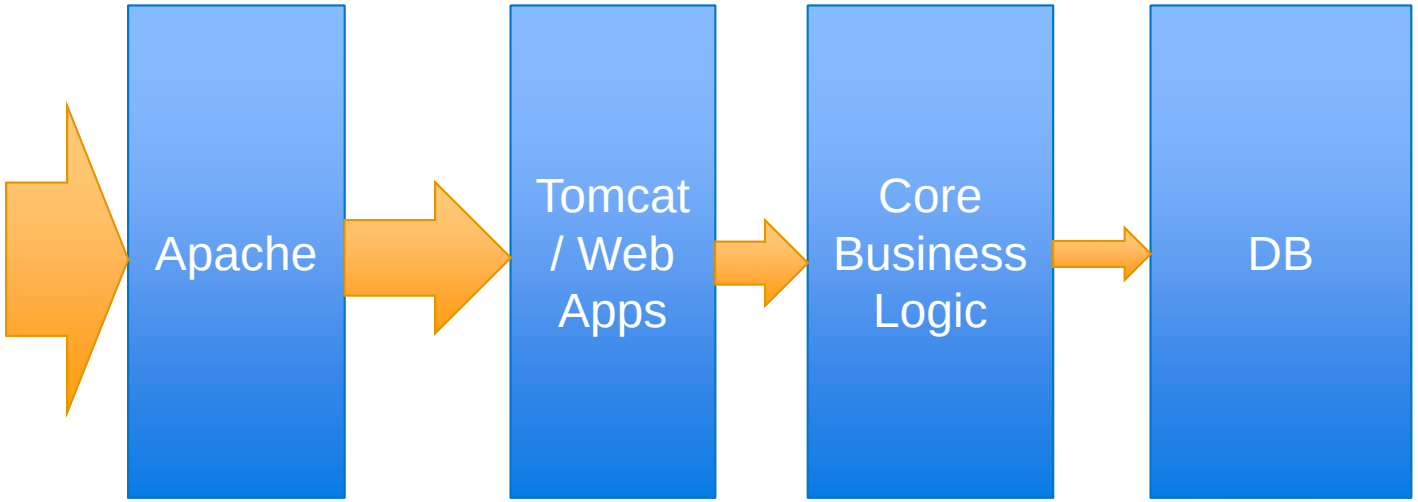
Liang Peng
Performance Engineer
April 2011

Outline

- Background
- Load testing approaches in functional testing
  - Story1: Finding performance defects in functional testing
  - Story2: Adjust load testing approaches in functional testing
  - Story3: Finding functional defects with load testing

Background – The Apache Threshold Module (ATM)

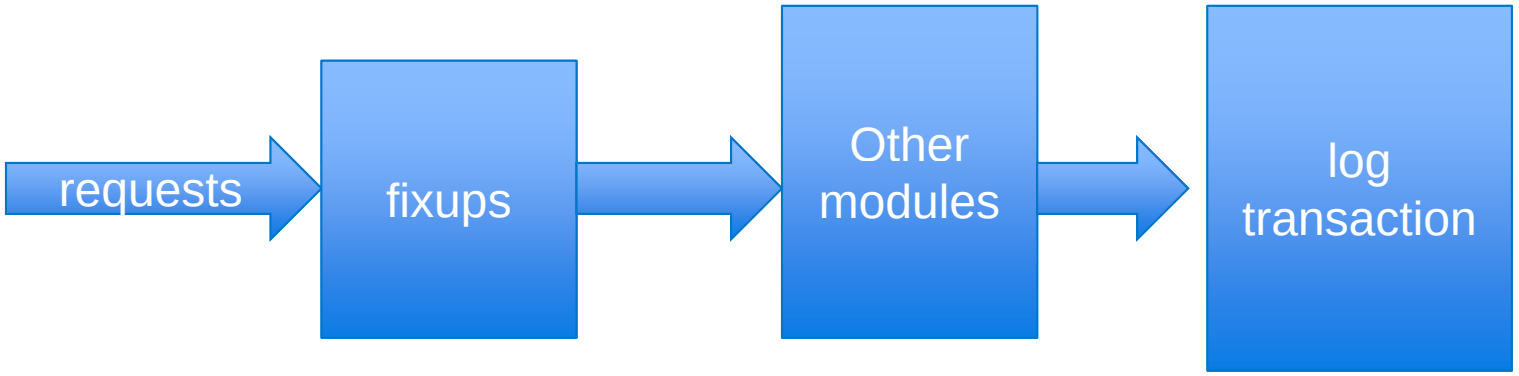
- What is ATM
- Why do we need this ATM



Background – The Apache Threshold Module (ATM)

- Why do we need this ATM
  - To control the traffic when it goes beyond the capacity that the system can handle
    - Based on throughput/threshold
    - Based on URLs
    - Based on session durations

How does ATM work



- It makes fixup and handler the first in dataflow
- It makes log_transaction the last in dataflow

How does functional testing ATM depend on load testing

- We need load testing approaches to simulate the loaded environment in which ATM will be working
- More about the load
  - HTTPS requests
  - No sessions are created yet
  - Load generated by HP using LoadRunner 9.5
  - The tested Apache/ATM in dedicated host and not connected to the architecture/platform yet

Story #1: Finding a performance defect in functional testing

SONY
make.believe

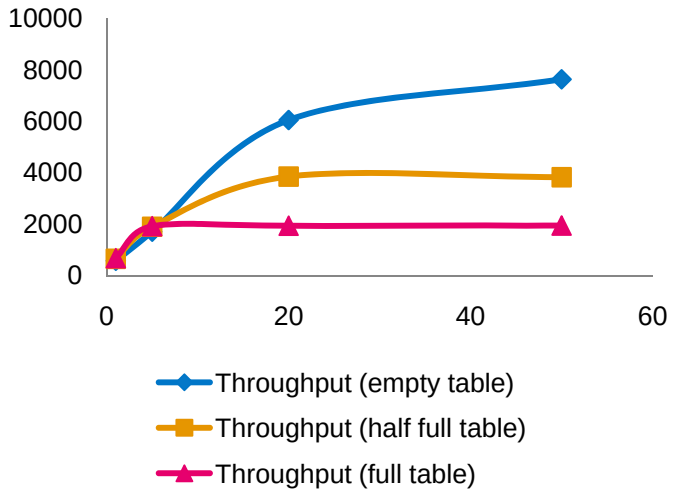
- Statistics Table in ATM
- From perspective of functional testing, we want to test if ATM works when the statistics table is full (empty and half full cases too, of course)

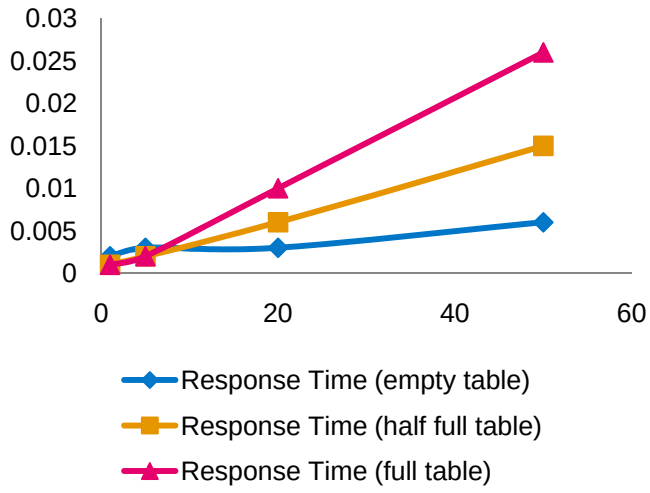
Story #1: Finding a performance defect in functional testing

SONY
make.believe

- Test cases (table size is 2048):
  - With only one URL requests (table very empty)
  - With 1024 URL requests (table half full)
  - With 2048 URL requests (table full)
- Conducting the tests:
  - Clearing the table by restarting apache (with ATM)
  - Run tests with certain number of URLs (1, 1024, and 2048)
- The ATM state changes works well (functionally)

Full table tests results:





Why the more full the table, the lower achievable throughput and higher latency?

A little bit more details of implementation

- The URLs in table are stored in an array
- For every single new request, ATM search the table and see if it's already in table
- If it's already in table, update statistics
- If it's not in table, add a new entry to the table for this URL

Here's the problem:

- There are a lot of search/comparisons in table
- String comparison/matching/search could be time consuming, especially when the string is not short
- When the string is an URL, it can easily go up to 50 characters, the worst case is to compare the entire URL with each existing element
- For a large number of URL requests to the same website, the chance for the URLs to be different in tail is very high, and this is the worst (or close to worst) case scenario

Suggestion:

- Change the data structure of the table, or
- Implement the table search in a more efficient way

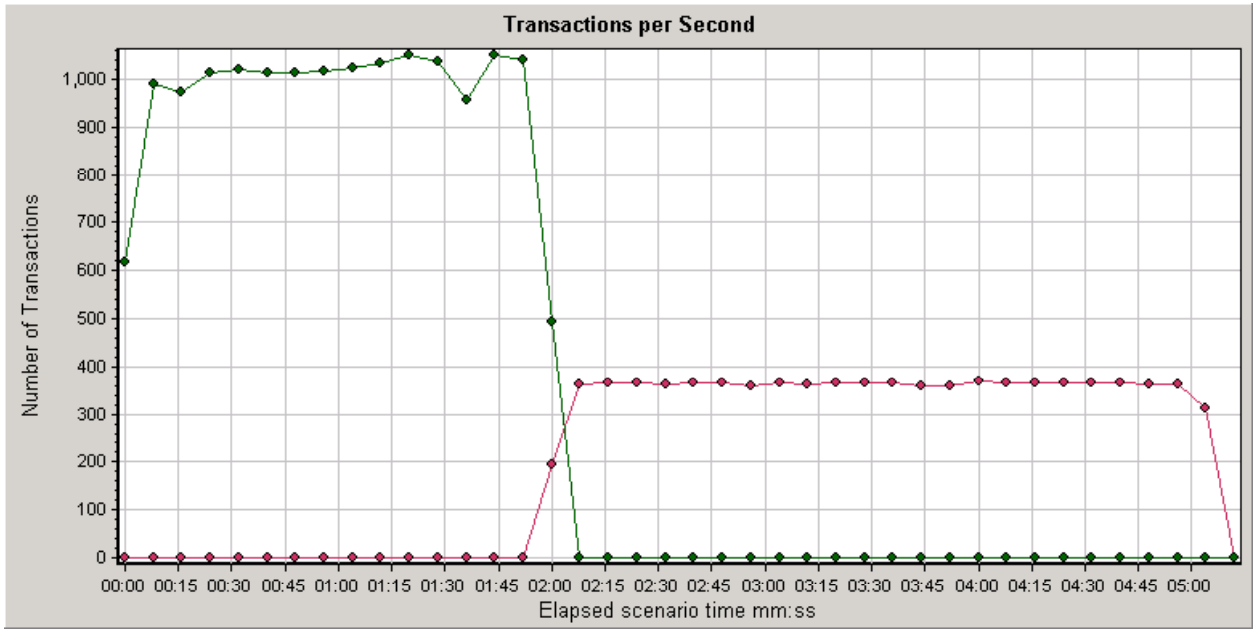
Story #2: Adjust load testing approaches in functional testing

- In functional testing the state change from *normal* to *iplock*, we need to verify when the system under load, changing ATM from *normal* to *iplock* will block all of the traffic

Executing the test:

- Set ATM in *normal* state
- Send out load to Apache/ATM
- When keep sending out load, change ATM state from *normal* to *iplock*
- Verify that all requests are blocked

Test results:



When all requests pass, the throughput is about 1,000/s. *iplock* works. However when all requests fail, the throughput much less. Why?

Making use of load testing methodologies in functional testing

- Tried to increase the throughput from 1,000/s to 10,000/s (by using more virtual users) but still seeing similar results: fail throughput much less, and the ratio of failure rate vs. pass rate is roughly same.

Making use of load testing methodologies

- Looking at the structure of load testing script:
- Main Action{

```
// preparation work
    ……
    For (i = 0; i < 10; i++){
      call_http_request_action();
    }
}
```

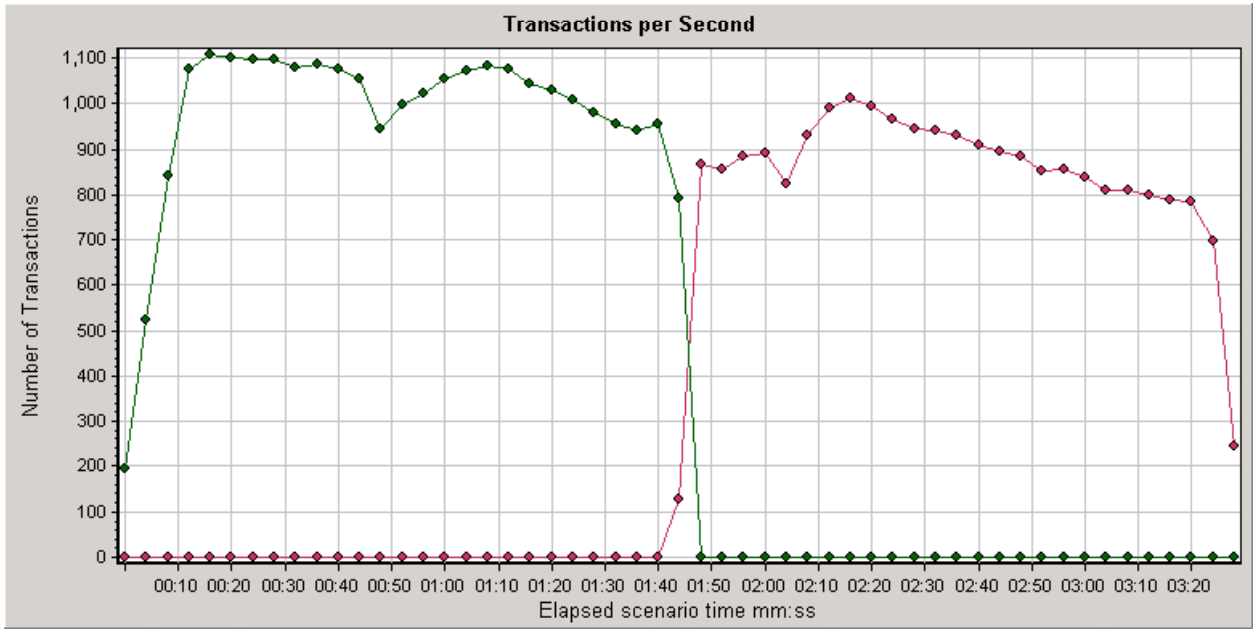
- In load testing, sometimes we iterate for more times to help increase throughput

Making use of load testing methodologies

- However when ATM in *iplock* state, all requests failed and the iteration will not continue upon failure
- Upon failure, the control logic goes to the next iteration of the entire action instead of next loop within the action
- That's why we see failure throughput is less than pass throughput

See what happen when I don't iterate

They are roughly at same level now:



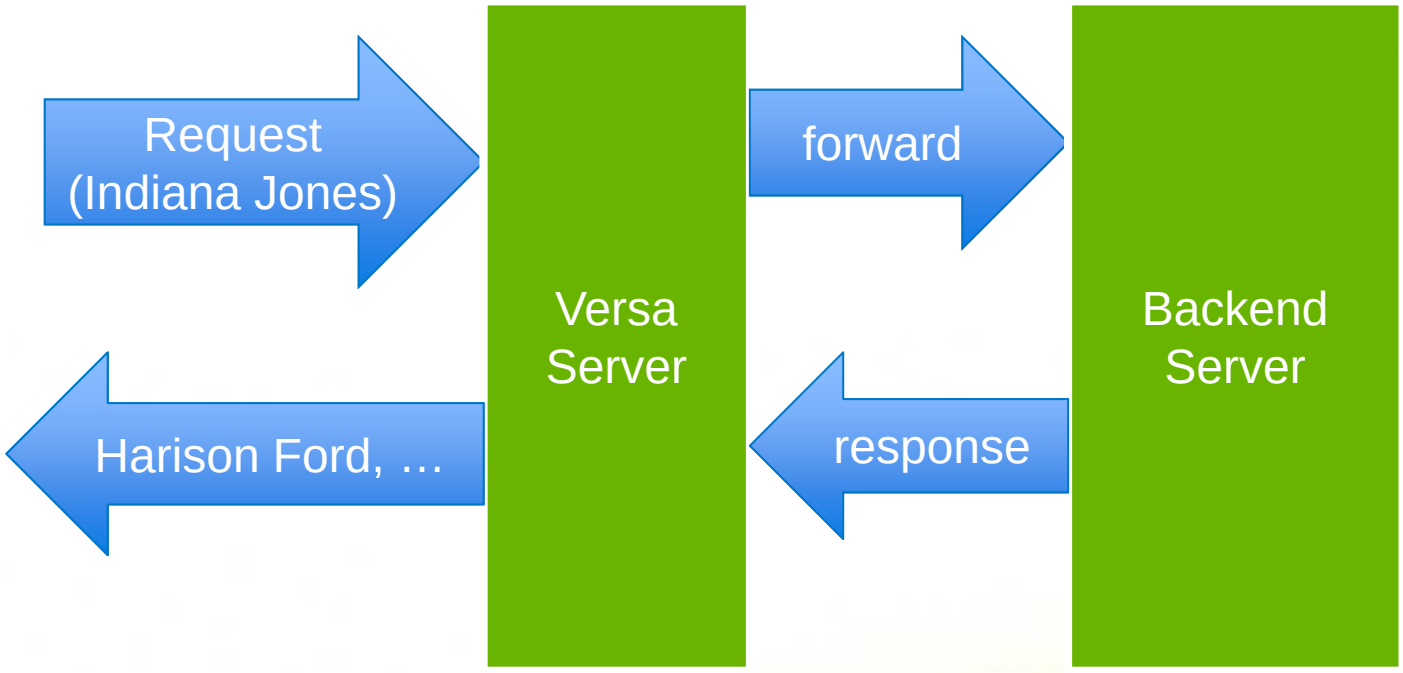
Failure throughput is still slightly less than pass throughput because failure response is bigger size

Issues in applying load testing methodologies in functional testing

- It’s not a one time thing
- Pass rate vs. failure rate

Store #3: Performance testing reveals functional defect

- Versa server resource usage problem
- A little bit background:



Story #3: Performance testing reveals functional defect

- What we want to do: performance characterization
- Executing the tests:
  - HTTPS requests with some parameters
  - Load is sent out by using LoadRunner (ramp up from light load to heavier load)
  - In the mean time monitoring client side throughput and server side resource usage

Story #3: Performance testing reveals functional defect

SONY
make.believe

- At light load (say, 5/s), everything looks fine
- When increase load to certain level (>5 to 10/s), system becomes not stable:
  - CPU 100% usage
  - Not responding to requests
- Only restarting the server can bring it back to normal status

Story #3: Performance testing reveals functional defect

SONY
make.believe

- The server saves a lot of contents in cache
- Upon request, it first see if it's already in cache
- The contents in cache are organized with a HashMap

Performance testing reveals functional defect

SONY
make.believe

- It turns out that it went into an infinite loop with Java HashMap, due to unsynchronized use of it
- Some people declare that it's a known issue and ConcurrentHashMap resolves this issue
- After the developer synchronized accesses to the cache, the issue was resolved

Takeaways

SONY
make.believe

- There are two situations where functional tests need performance test approaches (there could be other situations, of course)
  - The functionality is about controlling/adjusting network traffic
  - The functionality is about handling a number of requests within certain period (potential concurrency/parallelism)
- Functional testing can find performance defects, and performance testing can also possibly find functional defects.