IS2545, LECTURE 17: PERFORMANCE TESTING 2

Bill Laboon

CATEGORIES OF PERFORMANCE TESTING

- Service-Oriented
  - Availability
  - Response Time
- Efficiency-Oriented
  - Throughput
  - Resource utilization

EFFICIENCY-ORIENTED PERFORMANCE TESTING

Measures how well an application takes advantage of the computational resources available.

WHY DO EFFICIENCY-ORIENTED PERFORMANCE TESTING?

1. More granular than service-oriented testing
2. Easier to pin down bottlenecks
3. Possible to determine if problem can be solved by hardware modification / scaling / upgrading / etc.
4. Talk in a language developers can understand
5. Easier to get large amounts of data

EXAMPLE

Rent-A-Cat has added a RESTful API showing which cats are available to rent. However, service-oriented testing has shown that it takes five seconds (minimum) to respond to /cats/list (which shows a sorted list of all available cats).

After some testing, you see that after being accessed, network usage is 1%, disk usage is 3%, memory usage is steady, but the CPU is pegged at 99% for five seconds.

Where would you look for solutions to this issue?

POSSIBLE ISSUES / AMELIORATIONS

1. Just need faster hardware – time to migrate away from the Commodore 64
2. Bad / lengthy HTML – generate pages better
3. Cats sorted with BogoSort – use better sorting algorithm
4. Lots of malloc/free calls – tune garbage collector, modify code to create fewer objects
5. Everything running on single machine - Spread work to other cores/ processors
6. Just too popular - Cache listings

BENEFITS OF EFFICIENCY-ORIENTED TESTS

- The service-oriented test can tell us the general idea of what is wrong, but to fix it, we often need to undertake efficiency-oriented testing, to determine the actual source of the problem.
- If response time was 40 milliseconds, nobody would have cared, and there would have been little need for the efficiency-oriented testing.

“PREMATURE OPTIMIZATION IS THE ROOT OF ALL EVIL” -KNUTH

Oftentimes, it makes more sense to do service-oriented testing first, then drill down with efficiency-oriented tests to find out where problems lie later.

THROUGHPUT TESTING

- What is throughput testing?
- Measuring the maximum number of events possible in a given timeframe.

EXAMPLES

- You have a router, and you would like to know how many packets it can handle in one second.
- You have a web server, you'd like to know how many static pages of a given size it can serve in one minute.
- You are running a video game server, you’d like to know how many users can play simultaneously.

HOW IS THAT DIFFERENT FROM SERVICE-ORIENTED TESTING?

1. A given user doesn't care about the number of users who can access a system, just about what it means for them
2. Often more granular (users don't care about, e.g., packets)

LOAD TESTING

- Variations on load testing (see slides from last lecture) can be used to test throughput
  - Increase number of events until system crashes
  - Increase number of events until response time falls below threshold
  - Etc.
- Perspective is of system, not the user

LOAD TESTING

- Load testing can also be used for subsystems
    - How many entries can be made in database before events start being queued?
    - How many calculations can occur in one minute?
    - How many threads can be started in a given timeframe?
    - How many images can be written to disk?
    - How many videos can be compressed?
    - Etc.

MEASURING RESOURCE UTILIZATION

- *You need tools for this*
  - Unless you can tell by the sound of your fan exactly how many operations your program is running on the CPU

TOOLS

- General purpose
  - Windows Systems – Task Manager, perfmon
  - OS X - Activity Monitor or Instruments, top
  - Unix systems - top, iostat, sar
- Program-Specific Profilers
  - JProfiler
  - VisualVM
  - gprof
  - Many, many more

A VERY SIMPLE EFFICIENCY TEST

Watch CPU usage while you do something

KEY RESOURCES TO WATCH

- CPU Usage
- Threads
- Memory
- Virtual Memory
- Disk I/O
- Network I/O

YOU CAN GET MORE SPECIFIC

- Disk cache misses
- File flushes
- # Destination Unreachable message
- IPv6 Fragments Received/Sec
- Outbound Network Packets discarded
- # Print Queue "Out of Paper" messages
- ACK msgs received by Distributed Routing Table

MEASURING MEMORY USAGE

- Understand difference between private bytes, virtual bytes, working set, etc.
  - Private bytes = What app has asked for
  - Working set = In physical memory
  - Virtual bytes = Total virtual space allocated
- Caching can mess you up
- Really only good for trends (e.g., whether or not you have a memory leak, not where it is)

RESOURCE UTILIZATION MONITORING OF THIS KIND IS VERY BROAD

- Lots of memory being taken up…
  - …but by what objects / classes / data?
- Lots of CPU being taken up…
  - …but by what methods / functions?
- Lots of packets sent…
  - but why? And what's in them?

MORE SPECIALIZED TOOLS

- Protocol analyzers
  - e.g., Wireshark or tcpdump
  - See exactly what packets are being sent/received
- Profilers
  - See exactly what is in memory
  - What methods are being called and how often
  - What objects/classes have been loaded

- There's a big jump between "is our app slow?" and "we are leaking memory by never removing ConnectionCounter objects, causing more swaps and GC as a percentage of CPU time, thus causing response time to increase monotonically and exponentially in relationship to uptime."

FIXING PERFORMANCE ISSUES

1. Determine if it is a performance problem.
   - If it’s not, let sleeping dogs lie!
2. Track down from top-level to low-level
3. Keep track of performance throughout versions