CSE 403
Lecture 19

Reliability Testing

slides created by Marty Stepp
http://www.cs.washington.edu/403/

Load testing

- **load testing**: Measuring system's behavior and performance when subjected to particular heavy tasks.
  - often tested at ~1.5x the expected SWL (safe working load)
    - testing a word processor by loading large documents
    - testing a printer by sending it many large print jobs
    - testing a web app by having several users connect at once

- Differences between performance and load testing
  - performance testing looks for code bottlenecks (which are often the same at any level of load) and fixes them
  - in load testing, *level of load remains fixed*
  - load testing measures performance at a given level of activity so that you can make decisions about allocating resources

Goals of load tests

- Expose bugs that don't show up during normal usage
  - memory leaks / memory management problems
  - buffer overflows
  - concurrency problems / race conditions

- Ensure that system meets performance requirements
  - load tests can be done repeatedly as regression tests to make sure system retains required performance

Stress testing

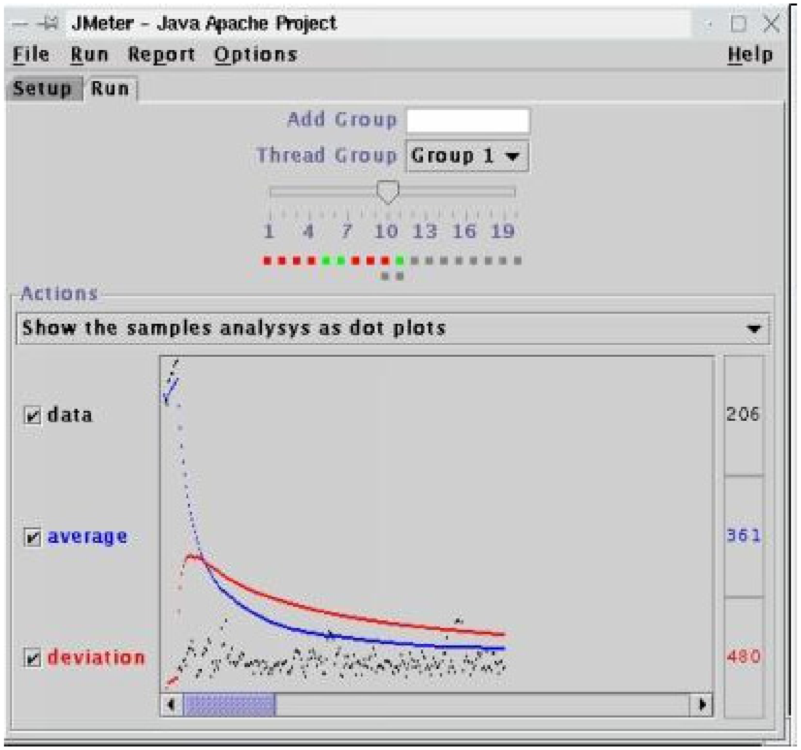
- **stress testing**: An extreme load test where the system is subjected to much higher than usual load.
  - often an error (timeout, etc.) is the expected result

- Goals of stress testing
  - see how gracefully the system handles unreasonable conditions
  - discover any data loss or corruption problems when under load
  - test recoverability and recovery time

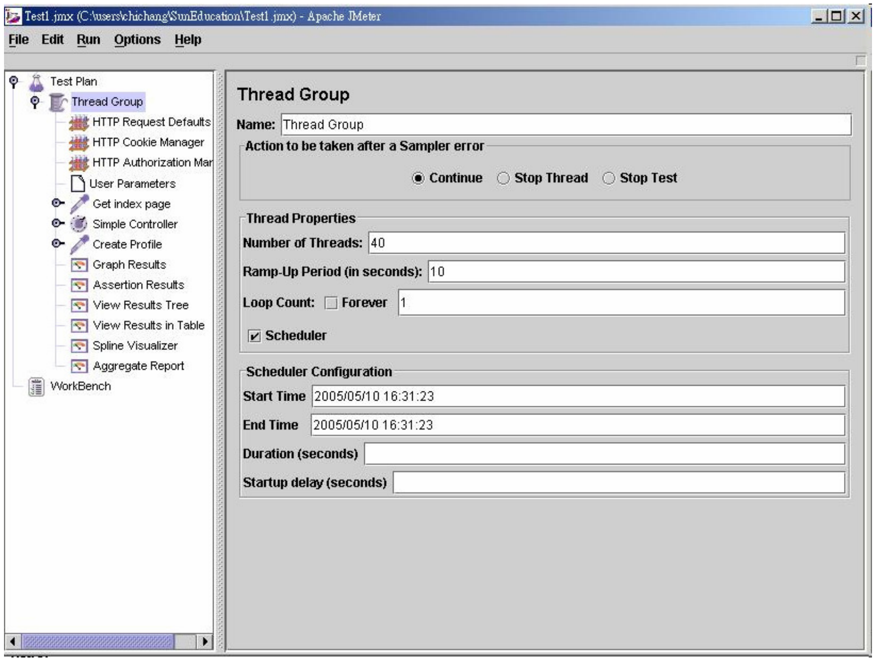
- Difference between load and stress testing
  - stress testing often runs at a level of load far beyond load tests
  - stress tests are meant to break things
  - there's no clear boundary between a load test and stress test

JMeter

- Apache's JMeter software performs load tests on a web server
  - Youtube Video: Getting Started with JMeter



Add Group
Thread Group: Group 1
Actions: Show the samples analysys as dot plots
✓ data 206
✓ average 361
✓ deviation 480


Test1.jmx (C:\users\chichang\SunEducation\Test1.jmx) - Apache JMeter
File Edit Run Options Help

Test Plan
- Thread Group
  - HTTP Request Defaults
  - HTTP Cookie Manager
  - HTTP Authorization Mar
  - User Parameters
  - Get index page
  - Simple Controller
  - Create Profile
  - Graph Results
  - Assertion Results
  - View Results Tree
  - View Results in Table
  - Spline Visualizer
  - Aggregate Report

WorkBench

Thread Group

Name: Thread Group

Action to be taken after a Sampler error: [✓ Continue] Stop Thread Stop Test

Thread Properties
Number of Threads: 40
Ramp-Up Period (in seconds): 10
Loop Count: ☐ Forever 1
✓ Scheduler

Scheduler Configuration
Start Time: 2005/05/10 16:31:23
End Time: 2005/05/10 16:31:23
Duration (seconds):
Startup delay (seconds):

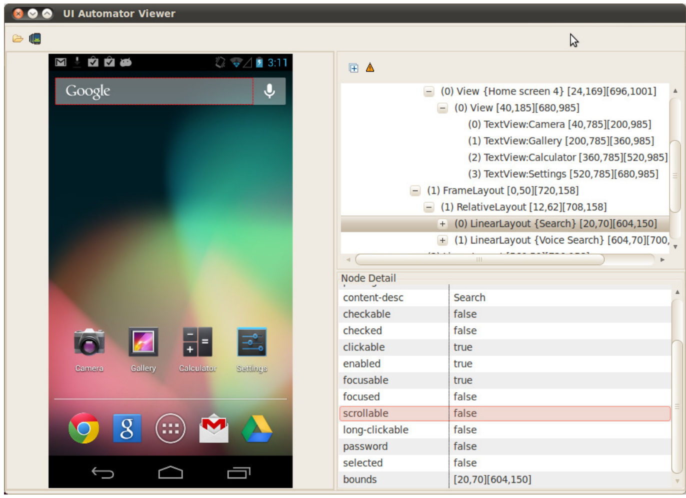
Android Monkey

- **Android UI/Application Exerciser Monkey**: Simulates lots of random clicks / UI interactions on your app.
  - `adb shell monkey` [options] <event-count>

    e.g. adb shell monkey -p your.package.name -s 42 -v 500
  - random, but can be seeded (-s 42) so results are repeatable

  - runs until:
    - given number of events are done
    - emulator exits your app
    - app crashes / throws exception
    - app-not-responding state

  - http://developer.android.com/tools/help/monkey.html



```
(0) View {Home screen 4} [24,169][696,1001]
  (0) View [40,185][680,985]
    (0) TextView:Camera [40,785][200,985]
    (1) TextView:Gallery [200,785][360,985]
    (2) TextView:Calculator [360,785][520,985]
    (3) TextView:Settings [520,785][680,985]
(1) FrameLayout [0,50][720,158]
  (1) RelativeLayout [12,62][708,158]
    (0) LinearLayout {Search} [20,70][604,150]
    (1) LinearLayout {Voice Search} [604,70][700,
```

content-desc	Search
checkable	false
checked	false
clickable	true
enabled	true
focusable	true
focused	false
scrollable	false
long-clickable	false
password	false
selected	false
bounds	[20,70][604,150]

System reliability

- **reliability testing**: Measuring how well the system can remain available and functional over time.

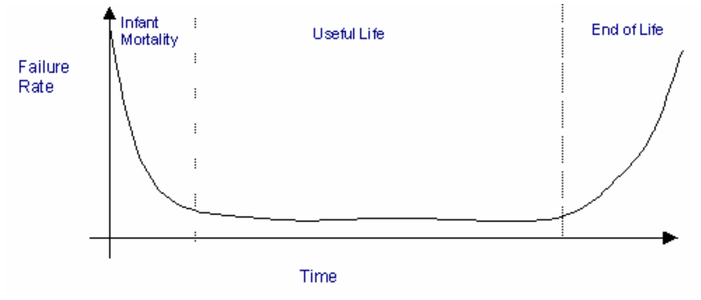
- Common goals:

"two nines":	99% uptime	(down 3.65 days/year)
"three nines":	99.9% uptime	(down 9 hours/year)
"four nines":	99.99% uptime	(down 52 minutes/year)
"five nines":	99.999% uptime	(down 5 minutes/year)

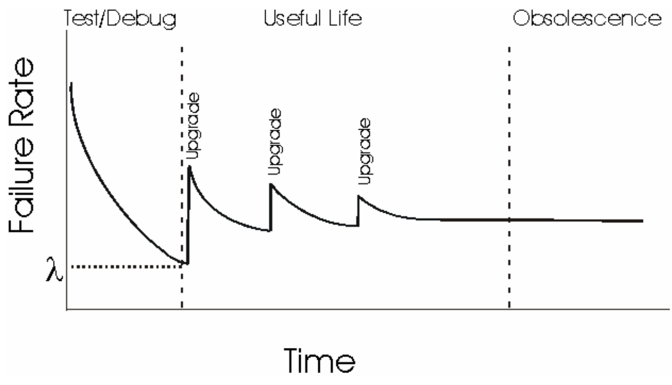
- What causes a system to "fail" or become unavailable?
  - software failures/bugs
  - hardware failures (broken HD, fan, etc.)
  - network failures
  - incorrect system configuration

Estimating failures

- Most hardware fails very early or late
  - mean time between failures (MTBF)
  - FITS: failures per billion hours



- Software failure depends on:
  - quality of design and process
  - complexity and size of system
  - experience of development team
  - percentage of reused code
  - depth of testing performed
  - **"bathtub curve"** (see right)
  - Software failures are often measured per 1000 lines of code.



Things to test

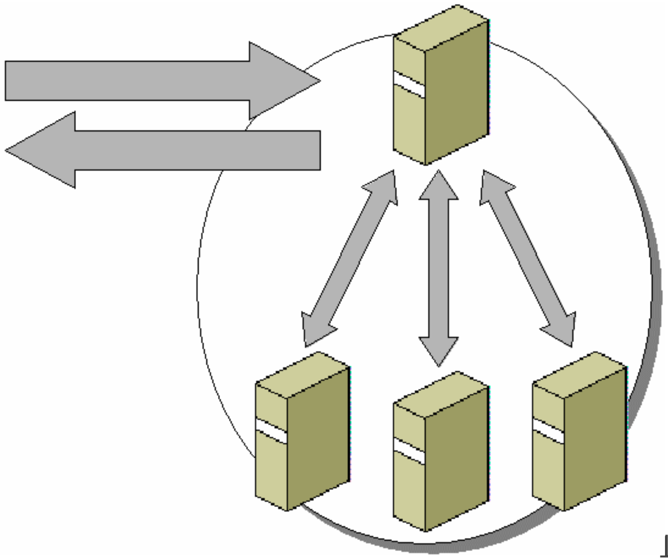
- What does the product do when subsystems go down?
  - database, file system, network connection, etc.
- What does the product do when subsystems degrade?
  - slow network/database, etc.
- What does the product do when it receives corrupted data?
  - garbage data through network, file system, etc.
- Does the site go down?
- Does the system display an error? If so, is it a good one?
- Does the site hang? How much does user experience suffer?

Improving reliability

- Replicate crucial system components
  - example: three web servers, two database servers, etc.

- Benefits of replication
  - system is only considered unavailable if BOTH are down
  - 2 replicas * 99% reliability => 99.99% reliability

- massive replication (e.g. Google)
  - If we have 20k machines and each has 99% reliability, how often will a machine fail?



Repairing failures

- Some failure recovery can be automated:
  - roll back transaction / fall back to another database
  - shift the load to another machine(s)
  - redirect to another site
  - reboot (Windows!)

- Some failure recovery requires a human component:
  - on-site support staff
  - on-call developers or QA engineers

- Estimating failure recovery time
  - MTTR (mean time to repair)
    - < 10 minutes if problem/fix are known
    - 1 hour to 2 weeks if not
    - availability = MTBF / (MTBF + MTTR)

Logging

- **logging**: Writing state information to files or databases as your application runs.

- Reasons to do logging:
  - logging is the println-debugging of web apps
  - notice patterns of repeated errors
  - a minimal guarantee on use cases: the error will be logged

- Why not just output error messages to the page?
  - the wrong person sees the message (user, not developer)
  - user shouldn't be told detailed information about the system or about the causes of errors
  - message is not permanent (shows on page, but then is lost)

Logging in Android

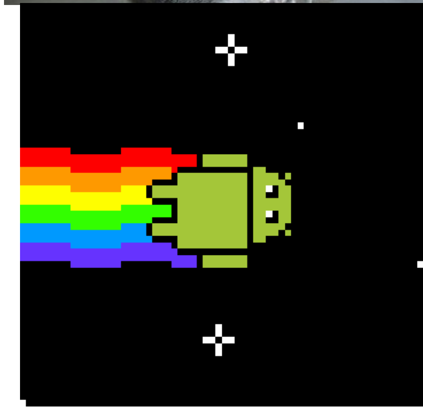
- Class `android.util.Log` has these *static* methods:

`d(`**tag**`, `**message[**`, `**ex]**`)`	writes a "debug" message to the log file
`e(`**tag**`, `**message[**`, `**ex]**`)`	writes an "error" message to the log file
`i(`**tag**`, `**message[**`, `**ex]**`)`	writes an "info" message to the log file
`v(`**tag**`, `**message[**`, `**ex]**`)`	writes a "verbose" message to the log file
`w(`**tag**`, `**message[**`, `**ex]**`)`	writes a "warning" message to the log file
`wtf(`**tag**`, `**message[**`, `**ex]**`)`	writes a "what the F" message to log file

- Example:

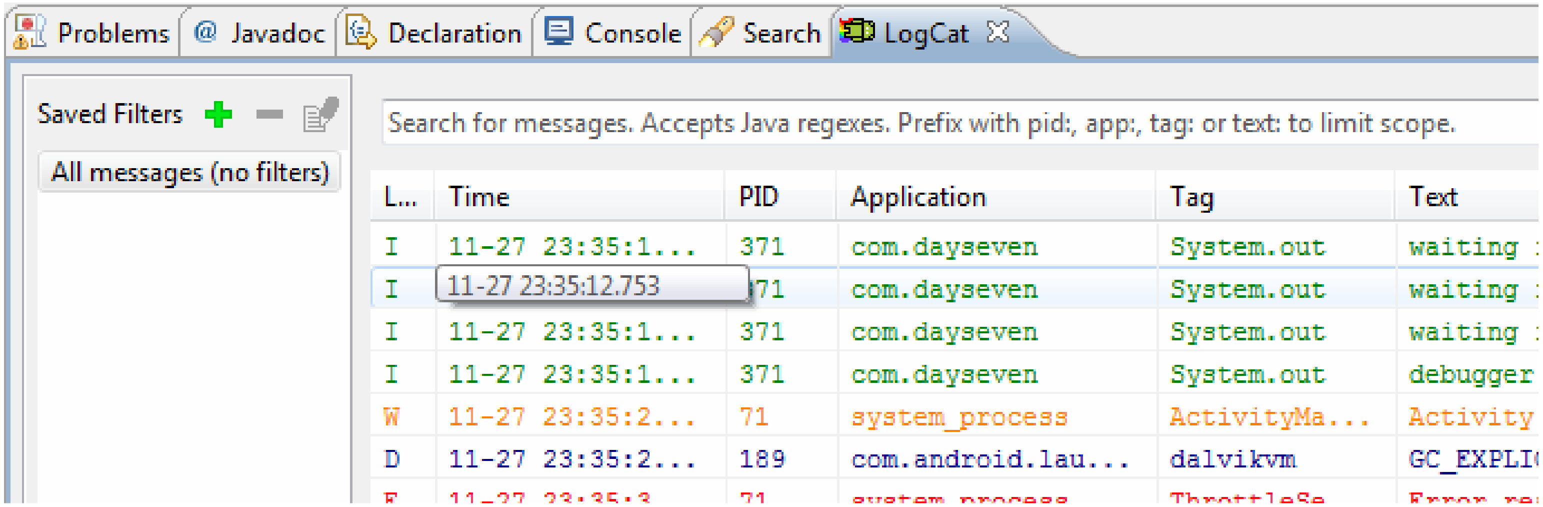
```
Log.i("MyActivity", "getView item #" + i);
```

LogCat



- **logcat**: A tool for viewing Android logs.
  - viewable in Eclipse
  - runnable as a stand-alone tool
    - adb logcat -v *format*

      e.g. adb logcat -v long



Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.

L...	Time	PID	Application	Tag	Text
I	11-27 23:35:1...	371	com.dayseven	System.out	waiting
I	11-27 23:35:12.753	371	com.dayseven	System.out	waiting
I	11-27 23:35:1...	371	com.dayseven	System.out	waiting
I	11-27 23:35:1...	371	com.dayseven	System.out	debugger
W	11-27 23:35:2...	71	system_process	ActivityMa...	Activity
D	11-27 23:35:2...	189	com.android.lau...	dalvikvm	GC_EXPLI
E	11-27 23:35:3	71	system_process	ThrottleSe	Error re

What to log

- log mostly errors, not success cases (don't flood the log file)

- log enough detail that you can understand it later
  - What page was the error on?
  - What method or command was called?
  - What parameters were passed?
  - What was the current date/time?
  - Who was the current user or IP (if known)?
  - What was the complete stack trace of any exception/error?

- Suggestion: switch log files daily
  - (many web frameworks do this for you automatically)

Properties of Java logging

- It is easy to suppress all log records or just those below a certain level, and just as easy to turn them back on.
- Minimal penalty for leaving logging code in your application.
- Log records can be directed to different handlers, for display in the console, for storage in a file, and so on.
- Logs can be formatted in different ways, e.g. text or XML.
- By default, logging configuration is controlled by a config file.

Logging in J2EE

- JSP pages can call a global `log` method to log errors:

`log(`**`message`**`)`	writes the given message string to the log file
`log(`**`message, ex`**`)`	writes the given message string to the log file, along with a stack trace of the given exception

- Example:

```
try {
    double avg = currentUser.getAverageScore();
    out.println(avg);
} catch (ArithmeticException e) {
    log("failed to get average score", e);
}
```

  – There is also a more advanced Apache `log4j` package (not shown)

`java.util.logging`

- **log levels**: Partition different kinds/severities of log messages.
  - Java's: SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST

- Using the global logger:
  - `Logger.global.info("File->Open menu selected");`
  - `Logger.global.setLevel(Level.FINE);`

  - `logger.warning(`**message**`);`
  - `logger.fine(`**message**`);`
  - `logger.log(Level.FINEST,` **message**`);`

- A specific logger for one part of your application:
  - `Logger myLog = Logger.getLogger("com.uw.myapp");`

Logging in Ruby on Rails

- Rails built-in `logger` lets you log errors, warnings, etc.:

```
require 'logger'

def add_to_cart
    product = Product.find(params[:id])
    @cart = find_cart
    @cart.add_product(product)
rescue ActiveRecord::RecordNotFound
    logger.error("invalid product #{params[:id]}" )
    flash[:notice] = "Invalid product"
    redirect_to :action => 'index'
end
```

- `logger` methods:
  - `logger.error`, `logger.fatal`, `logger.warn`, `logger.info`

Log file contents

- Web server software has its own request/error logs
  - A typical Apache log file `/var/log/apache2/error.log` :

```
[Tue Nov 20 12:07:37 2007] [notice] Apache/2.2.6 (Win32) mod_ssl/2.2.6 OpenSSL/0.9.8e
   PHP/5.2.5 configured -- resuming normal operations
[Tue Nov 20 12:07:37 2007] [notice] Server built: Sep  5 2007 08:58:56
[Tue Nov 20 12:07:37 2007] [notice] Parent: Created child process 8872
PHP Notice:  Constant XML_ELEMENT_NODE already defined in Unknown on line 0
[Sun Nov 04 18:29:19 2007] [error] [client 24.17.240.98] File does not exist:
   D:/www/gradeit/courses/142/07au/a5/files/BL/BEIROUTY_0726779/output_3_2.png, referer:
   http://pascal.cs.washington.edu:8080/gradeit/testcases_view.php?course=142&quarter=07au&
   assignment=a5&section=BL&student=BEIROUTY_0726779&small=1
[Sun Nov 04 18:29:19 2007] [error] [client 24.17.240.98] File does not exist:
   D:/www/gradeit/courses/142/07au/a5/files/BL/BEIROUTY_0726779/output_3_3.png, referer:
   http://pascal.cs.washington.edu:8080/gradeit/testcases_view.php?course=142&quarter=07au&
   assignment=a5&section=BL&student=BEIROUTY_0726779&small=1
[Tue Oct 30 17:22:06 2007] [error] [client 128.95.141.49] PHP Warning:  Invalid argument
   supplied for foreach() in D:\\www\\gradeit\\Section.php on line 43, referer:
   http://pascal.cs.washington.edu:8080/gradeit/assignment_view.php?course=142&quarter=07au
   &assignment=a5
[Tue Oct 30 17:22:06 2007] [error] [client 128.95.141.49] PHP Warning:  Invalid argument
   supplied for foreach() in D:\\www\\gradeit\\Section.php on line 43, referer:
   http://pascal.cs.washington.edu:8080/gradeit/assignment_view.php?course=142&quarter=07au
   &assignment=a5
```

Viewing Rails logs

```
$ gem install production_log_analyzer \
                 --include-dependencies
```

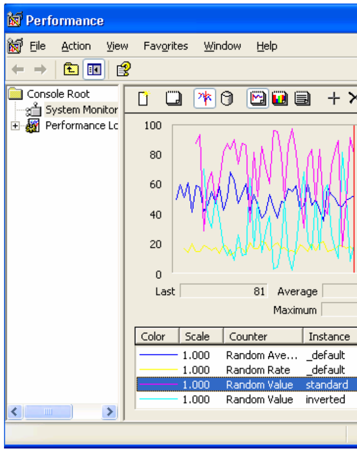
- Then, to identify potential performance bottlenecks in our application, use `pl_analyze` command to generate a report:

```
$ pl_analyze log/production.log
Request Times Summary: Count Avg Std Dev Min Max
ALL REQUESTS:              8 0.011 0.014 0.002 0.045
PeopleController#index:    2 0.013 0.010 0.003 0.023
PeopleController#show:     2 0.002 0.000 0.002 0.003
PeopleController#create:   1 0.005 0.000 0.005 0.005
PeopleController#new:      1 0.045 0.000 0.045 0.045

Slowest Request Times:
PeopleController#new     took 0.045s
PeopleController#index   took 0.023s
PeopleController#create  took 0.005s
PeopleController#index   took 0.003s
PeopleController#show    took 0.003s
```

Performance counters

- **performance counter**: A piece of hardware or software inserted to aid in measuring the efficiency of a system.
  - **hardware** perf counters are often special registers in a chip
  - **software** perf counters are timing code inserted and logged
- software (e.g. `perfmon`) can be installed to aid perf. counters
- programmers can add their own monitoring code
  - grab system time before and after a task; log it
  - ask for system info (RAM, HD, CPU %, etc.); log it
  - **counters** increase continuously
    - e.g. hit counter, DB query counter, error counter
  - **gauges** report an absolute value at a given time
    - e.g. RAM free; CPU usage

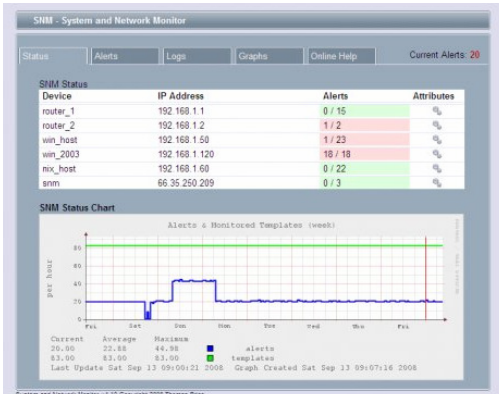


Color	Scale	Counter	Instance
	1.000	Random Ave…	_default
	1.000	Random Rate	_default
	1.000	Random Value	standard
	1.000	Random Value	inverted

Web site monitoring

- **web site monitoring**: A program or third-party machine "pings" your server periodically to make sure it is live.
  - downloadable scripts/apps (ZABBIX, Spong, CheckWebSite)
    - many based on SNMP (simple network monitoring protocol)
    - JMX (Java Management Extensions) for J2EE web apps
  - paid services (ExternalTest, SiteUpTime)

  - *internal:* installed on your own server, as root
  - *external:* runs on another server, connects remotely

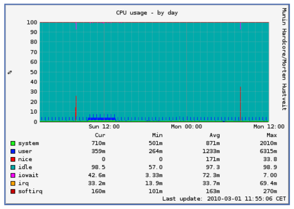


SNM Status

Device	IP Address	Alerts	Attributes
router_1	192.168.1.1	0 / 15	
router_2	192.168.1.2	1 / 2	
win_host	192.168.1.50	1 / 23	
win_2003	192.168.1.120	18 / 18	
nix_host	192.168.1.60	0 / 22	
snm	66.35.250.209	0 / 3	

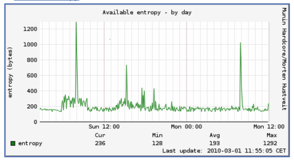
SNM Status Chart

```
Current    Average    Maximum
20.00      22.88      44.98     alerts
83.00      83.00      83.00     templates
Last Update Sat Sep 13 09:00:21 2008   Graph Created Sat Sep 13 09:07:16 2008
```



```
                Cur      Min      Avg      Max
■ system       710m     501m     871m    2010m
■ user         359m     264m    1233m    6315m
■ nice            0        0     171m     33.8
■ idle         98.5     57.0     97.3     98.9
■ iowait      42.6m    3.33m    72.3m     7.00
■ irq         33.2m    13.9m    33.7m    69.4m
■ softirq      160m     101m     163m     270m
                      Last update: 2010-03-01 11:55:06 CET
```

:: Available entropy



```
                Cur      Min      Avg      Max
■ entropy       236      128      193     1292
                      Last update: 2010-03-01 11:55:05 CET
```

Reliability Engineer (SRE)

- **software reliability engineer (SRE)**: Responsible for making sure network services stay running.
  - a job description recently popularized by Google
  - a mix of coding, sw. engineering, testing, system administration
- some SRE responsibilities:
  - review specs/design to point out vulnerabilities and problems
  - make sure that sites and products can tolerate growth/scaling
  - help deploy hardware and servers for new products/sites
  - write scripts to automate reliability tests, monitoring, sys. config
  - carry pager that goes off if site goes down or subsystems fail
  - analyze logs and data to discover potential issues
- *goal:* Heavily automate diagnosis <u>and repair</u> of major issues.

Quotes from an SRE

"Much of (SRE) is just encouraging people to add proper instrumentation to software, especially performance counters, and then graphing the output. Code coverage doesn't hurt, but that's more of a safety measure for engineers. They're less likely to have problems and be woken up at 3am to resolve an issue if the code is tested first, especially test cases where their dependencies are slow or broken."

"Performance counters are a big help. Logging is an even bigger help if you can replay logs to do load/performance tests in the future. Degrading gracefully under heavy load or subcomponent failure is a good approach that gets insufficient attention. In some cases, latency may be more important than correctness."

"Monitoring is a little simpler. There are black box tests that can be run on the outside to see if the service is responding correctly or not. With the right performance counters you can start to do white box monitoring. e.g. if the ms of CPU needed per request step outside some bound of "normal" that may need some investigation. Is a limited resource being starved? Is it one particularly complicated query that is coming in repeatedly? Is that query malicious?"

-- Andy Carrel, SRE, Google

SRE interview questions

- What is the difference between a hard and soft link?
- What is an inode, and what is inside it?
- What is RAID, and how does it work?

- What is a "zombie process"?
- What happens when a process is forked?
- Describe how a ping and a traceroute work.

- When user types "google.com" in the browser, what happens?
- Describe the TCP/IP 3-way handshake: Syn, Syn/Ack, Ack.
- What is the difference between a network switch and a hub?

- Write a shell script to rename *.html to *.htm and vice versa.
- Write a program to remove all userids matching a pattern.