How to Do Website Performance Analysis and Tuning with Flood and Apache 2.0

Justin Erenkrantz *University of California, Irvine*

Aaron Bannert *Covalent Technologies*

No, not this type of flood

- http://www.eng.uci.edu/~bfs/flood.jpg

Flood

- http://httpd.apache.org/test/flood/
- Subproject of the Apache HTTP Server Project
- Licensed under the **Apache Software License** [a]
- ***Mailing List:*** **test-dev-subscribe@httpd.apache.org**

Updated slides at: *http://www.clove.org/flood-presentation/*

[a] http://www.apache.org/LICENSE.txt

Why should you care about performance?

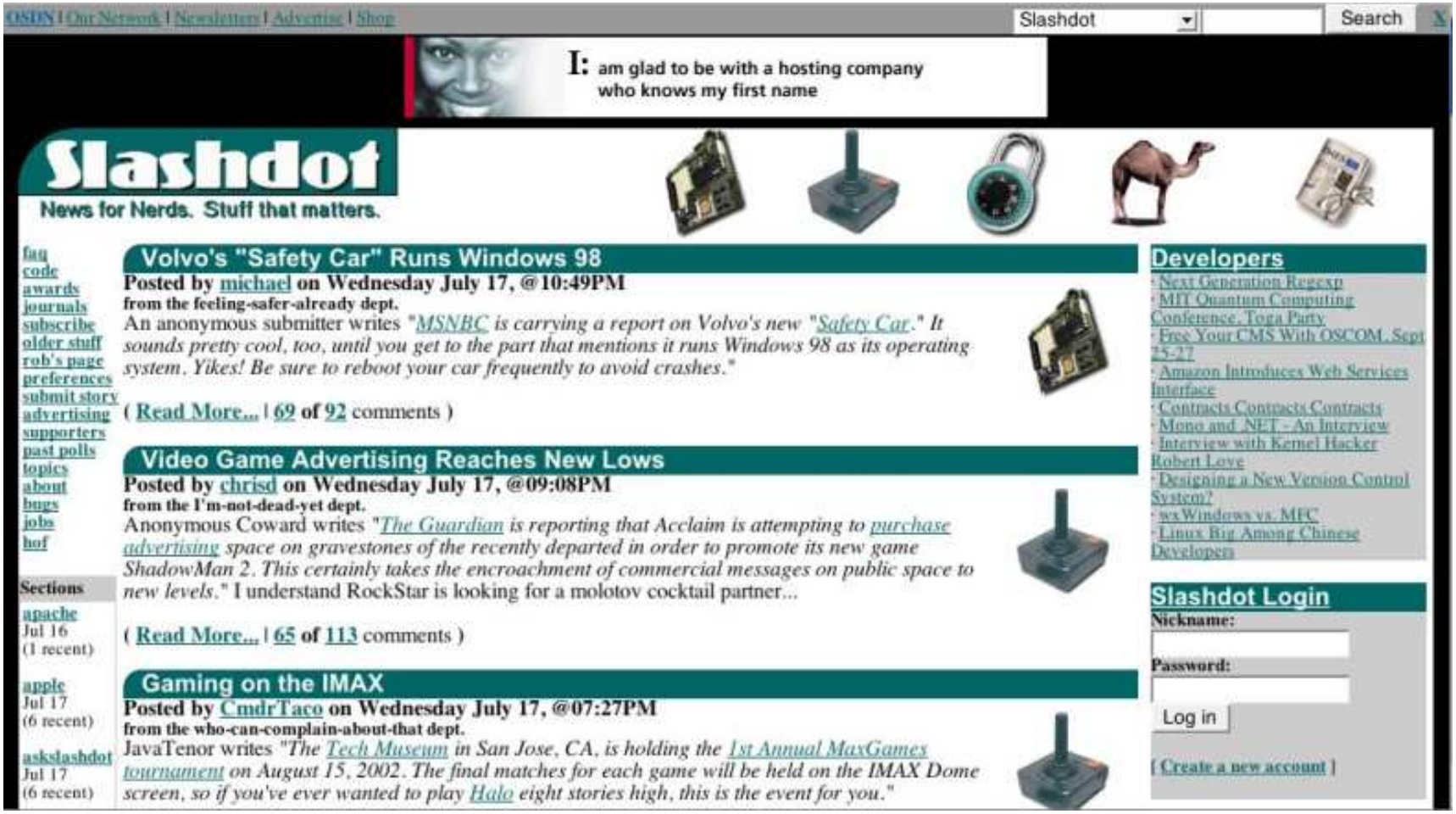
Performance equals $$

- Meet expectations
- Economies of scale
- Bang for buck

Capacity Planning

- How far can you go?
- Average load
- Peak load

Learn to love the Slashdot effect



Learn to love the Slashdot effect (cont.)

- Publicity is good
- May miss opportunity
- Need to capitalize

/.'ed, I will repost image later

Denial of Service

- Worms
- Malicious attacks
- Email harvesters
- Robots

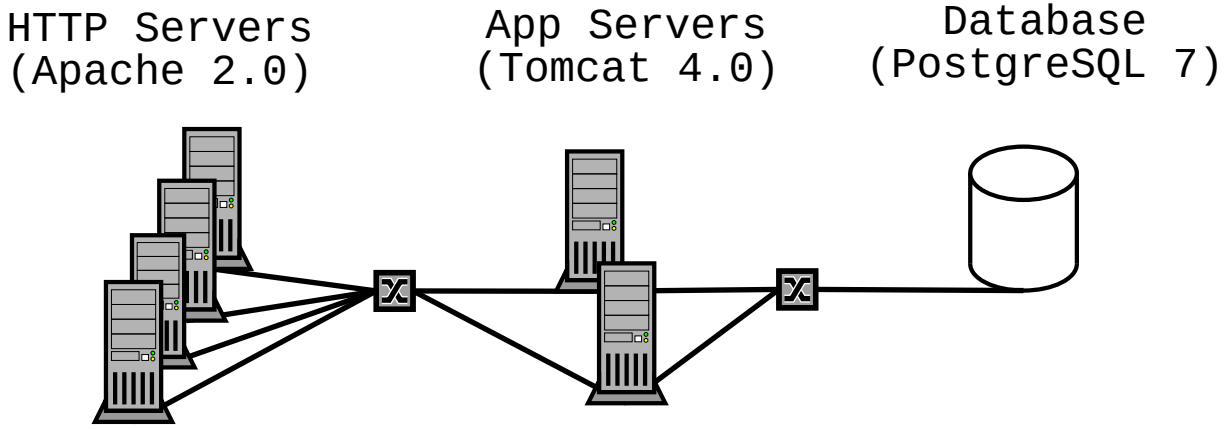
Unforseen factors

- Catastrophic events
  - A spokeswoman for Keynote Europe, a firm that monitors Internet performance, said “the [9/11] slowdown was worse than ... Code Red.” [a]
- May affect your ISP
- May affect potential client’s ISP

[a] http://news.cnet.com/investor/news/newsitem/0-9900-1028-7130948-0.html

Your site

- “How well does your site size up?”



Performance as design criteria

- Architecture can dictate performance
- Consider performance from the beginning
- Make it a part of your process

Performance as continuing practice

- Tuning
- Measurement
- Analysis

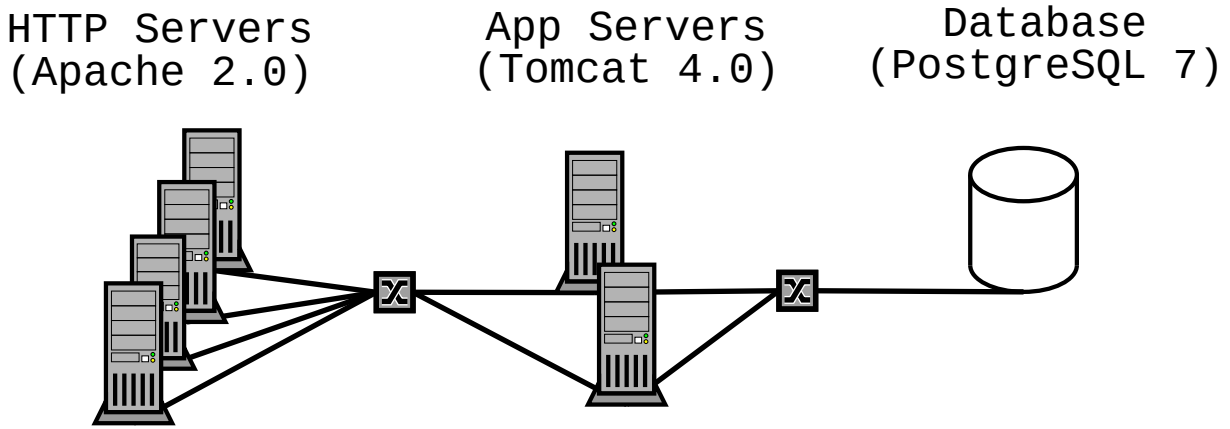
Approaching Performance Analysis

"A Recipe for Success"

1. Identify components

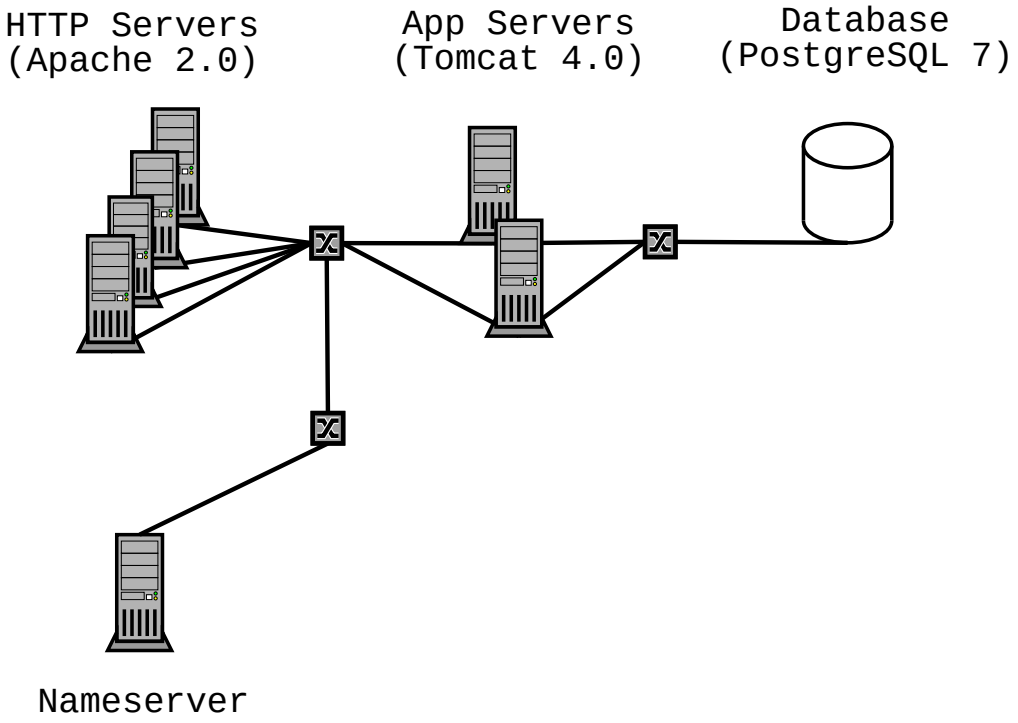
Does this represent your system?

- 1-tier
- 2-tier
- 3-tier



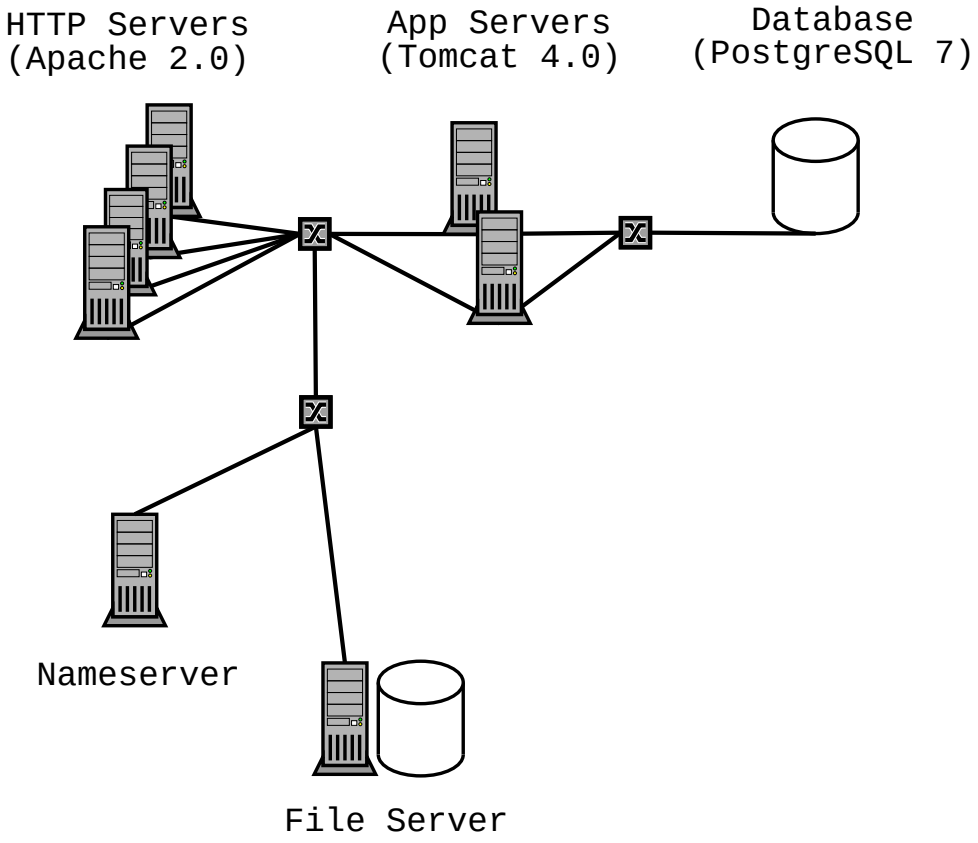
Identify components (cont.)

What about the Nameserver?



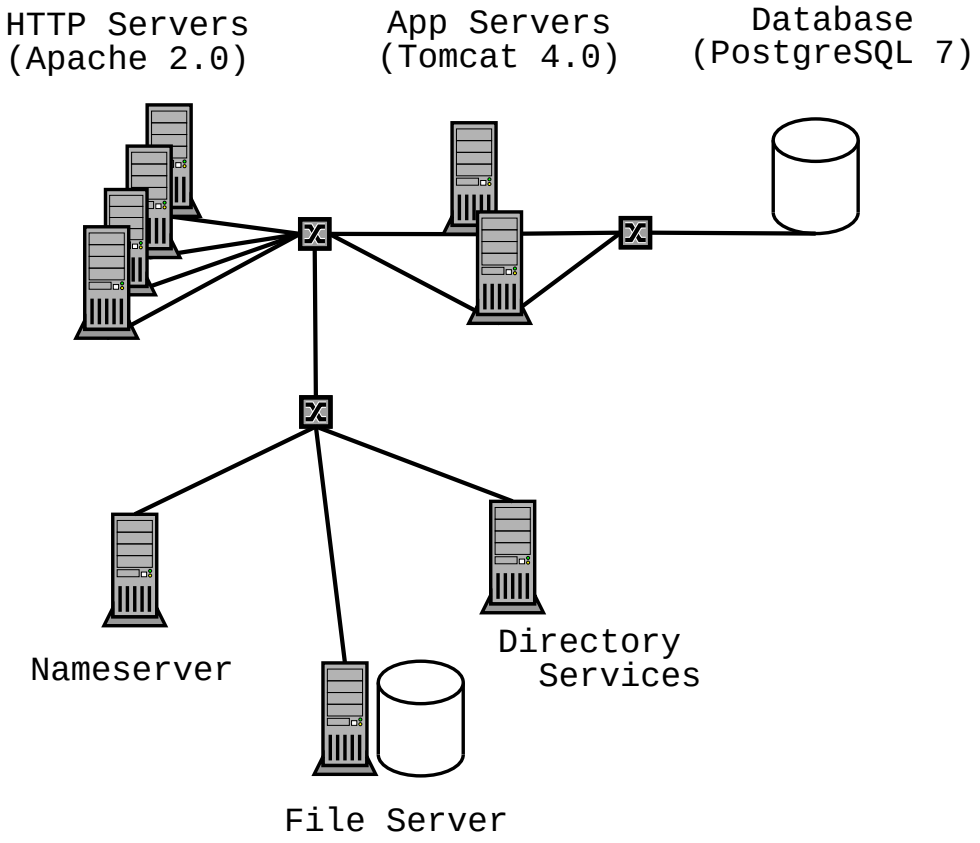
Identify components (cont.)

Do you have a file server?



Identify components (cont.)

How about “Single Signon”?



What's your point?

- Small components can affect the performance.
- Therefore, we need to look all the components in detail.

What do we have?

- Web servers
- Application servers
- Custom modules
- Custom applications
- Databases
- ...

What else do we have?

- File servers
- Nameservers
- Directory services (LDAP)
- Authentiation services
- SSL accelerators
- ...

What about the network?

- Hubs/Switches/Cables
- Routers
- Firewalls
- Transparent caches (HTTP caches)
- Load Balancers
- ...

2. How do components interact?

- Inter-component dependencies
- e.g. Setting up an account requires 10 table inserts in the DB.

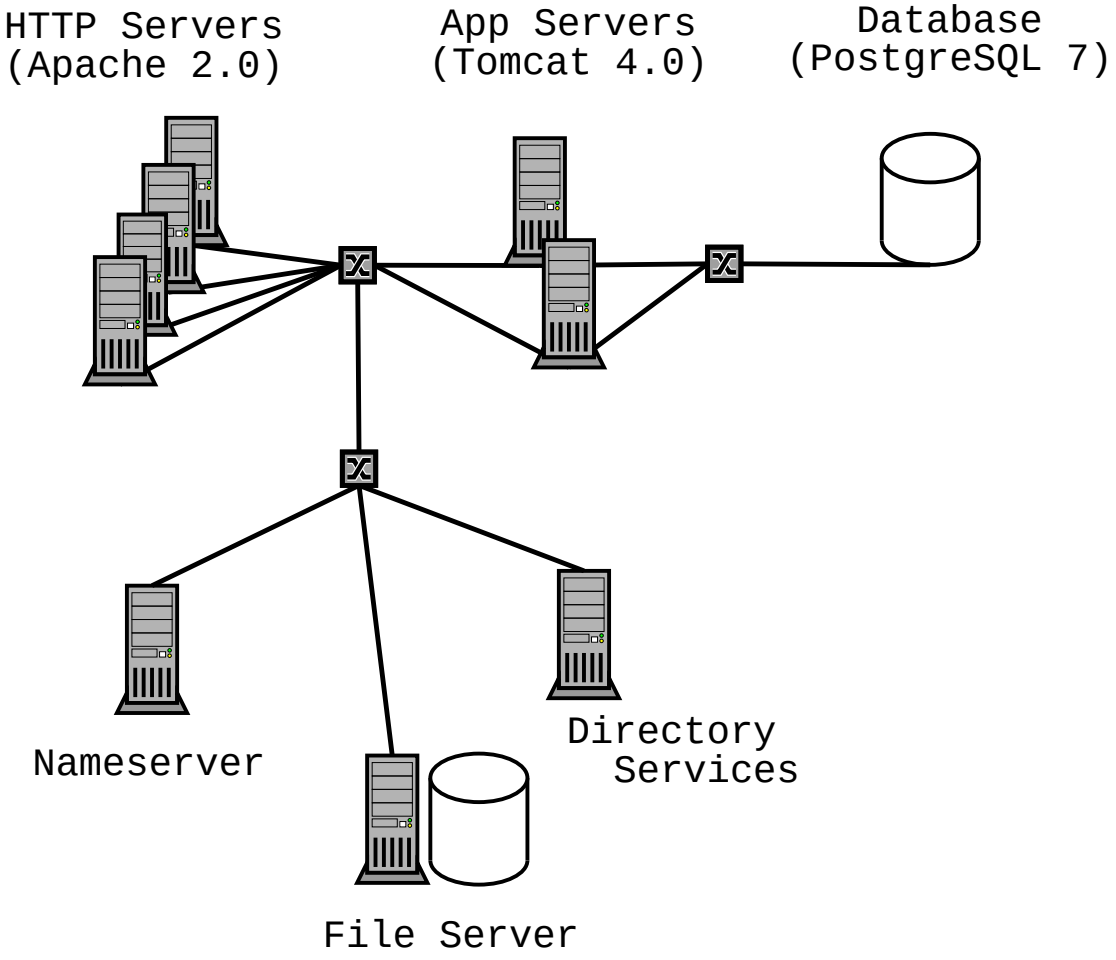
3. Identify points of contention

- Single point(s) of failure
- Critical resources
- Failure conditions
- Recovery plans

4. What information do we already have?

- OS statistics
- Network statistics
- Cache performance
- DB statistics
- Homebrew monitors

What is wrong with this picture?



5. We need to think about the users

- The web browser

How do we emulate such a complex set of users?

Why we developed Flood

"Let's generate some load!"

Scratch an itch

- Java-based web applications
- Performance requirements in the contract

Why we developed Flood? (cont.)

- Previous ad-hoc tools:
  - Shell scripts (`netcat`, `curl`, etc...)
  - Multiple clients via `ssh`/`rsh`
  - Coarse measurements
  - Difficult to control

Why we really open-sourced flood

- So others may benefit
  - Building a community
- So others may contribute
  - Further our work

Who wants flood?

Target Users:

- QA Engineer
- Performance Testers

Target Applications:

- Websites
- HTTP Apps
- “Web Services”

Features

- HTTP/1.1
- SSL
- Emulate users
- Emulate multiple users
- Emulate multiple complex users
- Emulate several different types of users

What can't it do?

- Automated scripting
- Find and fix your performance problems

This means that flood is merely a load generating tool that can aid in the collection of data. Analysis and interpretation of that data must be done by a human familiar with the system. In short, we can not fix your problems, but we can help you to identify them.

Flood Design

Design Goals

- Modular
- Simple but flexible configuration
- Scalable
- Accurate
- Portable

Modularity

- Easily add new features
- Design a framework
  - Add the kitchen sink later
- Framework defines actions
- Modules define behavior

What Flood modules exist today?

- normal BSD sockets
- SSL
- Round-robin URL lists
- Simple reports
- Relative-timing reports

Why XML?

- Simple
- Flexible
- Machine verifiable [a]
- Easily human generated
- Disadvantage:
  - some input must be XML-encoded, (which is ugly)

[a] so we can write fancy frontends later

Scalability

- Huge webserver farms
- Take advantage of multiple [a]
  - threads
  - processes
  - machines

[a] in order to eliminate hardware constraints

Accuracy

- Minimal overhead
- Reproduceable results
- Accuracy vs. Precision

Portability

- APR [a]
- Platforms (the short list):
  - Linux
  - FreeBSD, NetBSD, OpenBSD, Darwin (Mac OS X)
  - Solaris
  - Windows [b]
  - ...

[a] Apache Portable Runtime (http://apr.apache.org/)

[b] Thanks to William Rowe

Running Flood

“Farm World”

What the heck is this wacko naming scheme?

- Problem: Terms like
  - *thread*
  - *worker*
  - *process*

  are overused and ambiguous.

Wacko naming scheme (cont.)

- Solution: Come up with our own naming scheme. [a]

[a] Side-effect: Allows us to think outside of concepts like *sequences* and *threads* and *remote processes*.

url

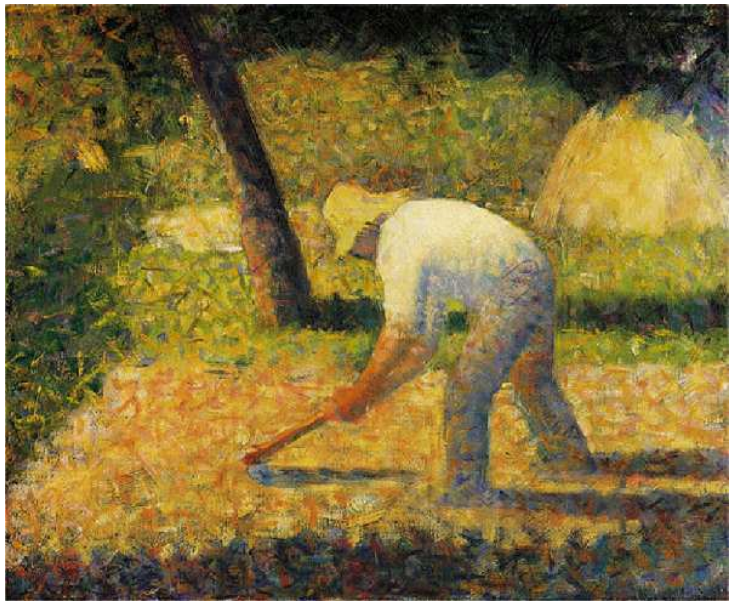
- You know what this is...

urllist

- Group urls together
- Predelay and postdelay

`farmer`

- Single thread
- Single site visitor
- Uses `urllists` in specific ways:
  - Random order
  - Round-robin (run in a loop)
  - Keepalive



farm

- Group of `farmer`s in parallel
- Controls ramp-up
- Creates/Controls `farmer`s:
  - looping
  - reordering [a]

[a] fine-grain control is not implemented, we only have simple looping at the moment

`collective`[a]

- A set of `farms` running on a single host in parallel

[a] The `collective` class hasn't been implemented, so this syntax is preliminary.

megaconglomerate[a]

- A set of `collectives` running on multiple host machines
- Invokes remote instances (using `rsh`/`ssh`/etc..)
- Central coordination
- Central reporting

[a]The `megaconglomerate` class hasn't been implemented, so this syntax is preliminary.

profile

- Extensions to core flood functionality
- Modules may override specific methods
- The runtime configuration is defined in the “profile”
- Examples of overridable methods:
  - `socket` *(*`generic`, `ssl`*)*
  - `verify_resp` *(200/OK)*
  - `report` *(*`easy`, `simple`, `relative_times`*)*

“Practical Analysis”

Let’s focus on Apache’s handling of server-side includes (SSI)

- Compare Apache 1.3 and Apache 2.0
- Same application
- Same client characteristics

What does the site look like?

- Main page:
  - SSI
  - contains 2 static images
  - links to secondary page and some big file
- Secondary page:
  - static HTML
  - contains 2 static images

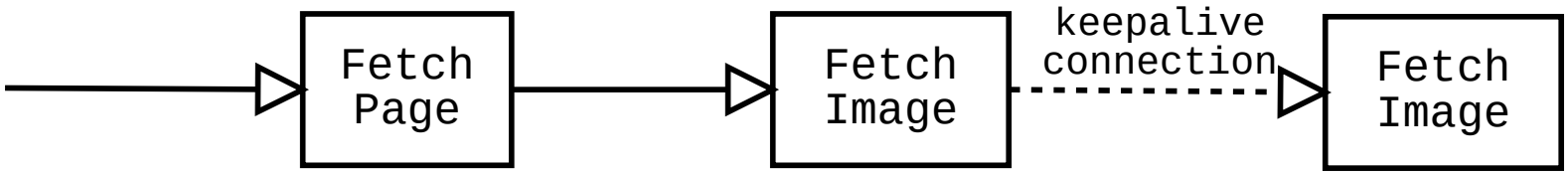
What are the use cases?

3 typical uses of this site:

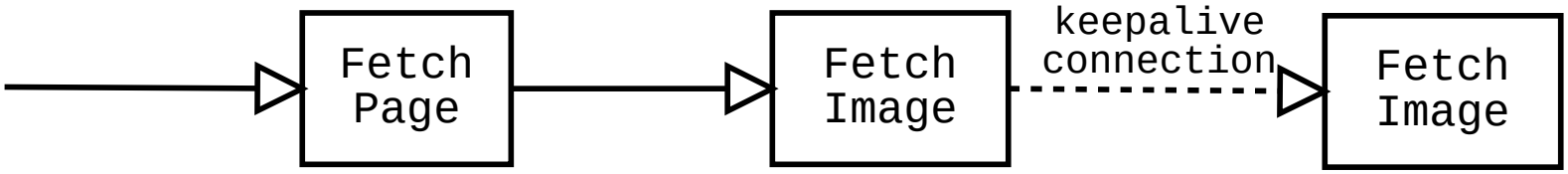
1. user just hits the main page
2. user hits main page then hits secondary page
3. user hits main page then downloads a big file

Gratuitous Flowcharts

- User just hits the main page:



Gratuitous Flowcharts

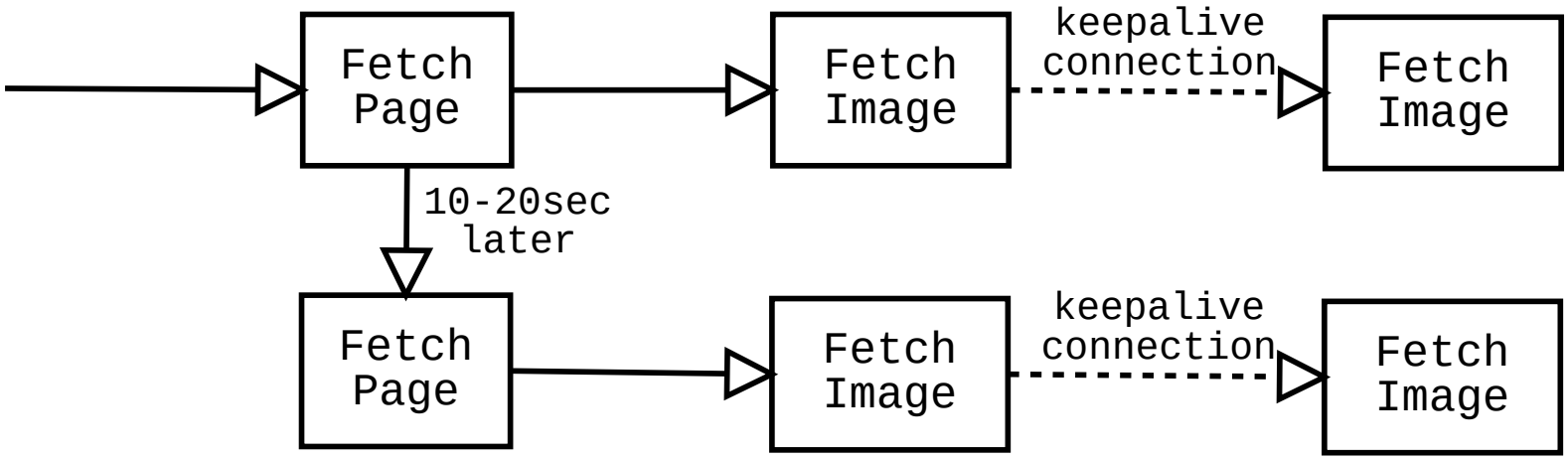


Sample XML Configuration

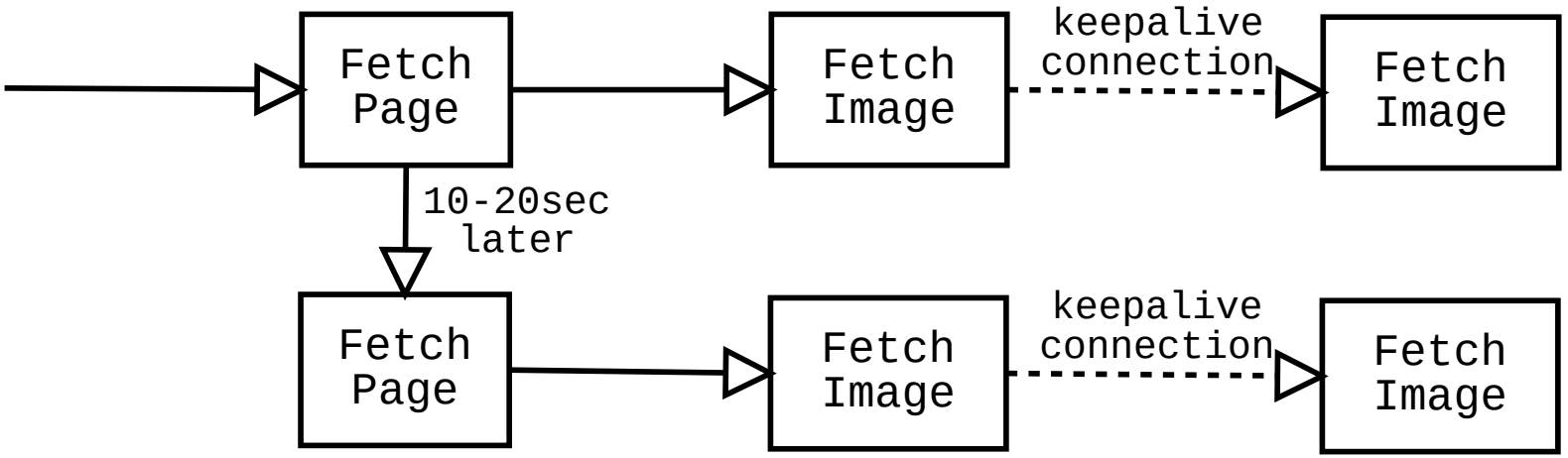
```
<urllist>
  <name>First</name>
  <description>Use Case 1</description>
  <url>http://localhost:8080/site/</url>
  <url>http://localhost:8080/site/httpd_logo_wide.gif</url>
  <url>http://localhost:8080/icons/apache_pb.png</url>
</urllist>
```

More Gratuitous Flowcharts

- User hits main page then hits one secondary page:



More Gratuitous Flowcharts

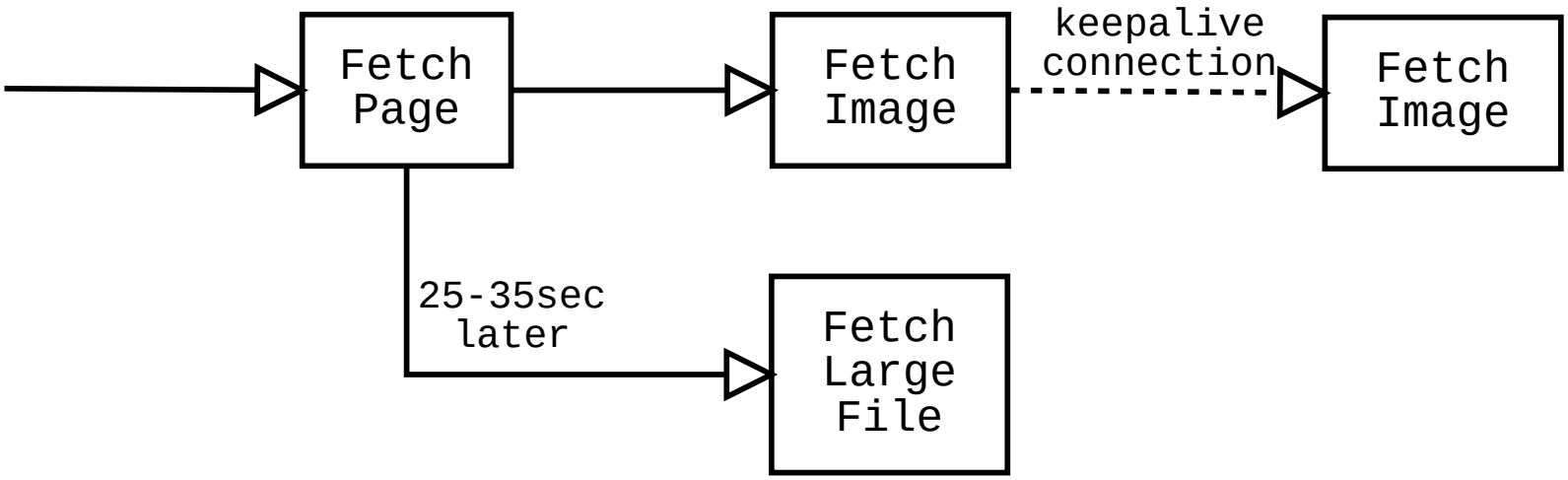


Sample XML Configuration

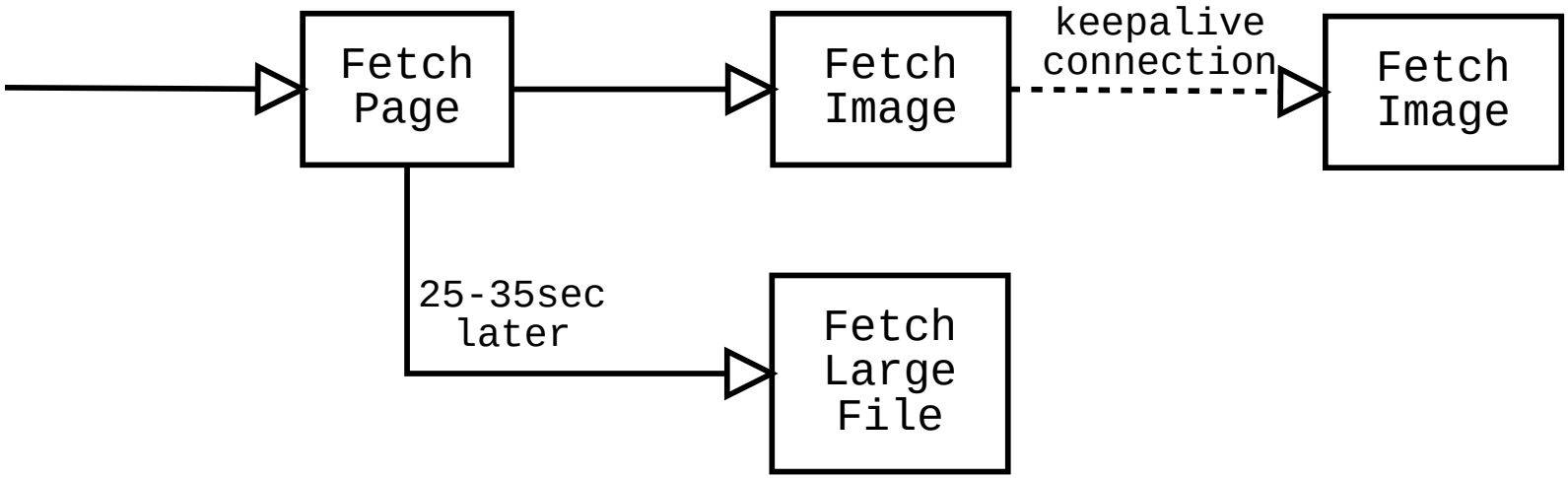
```
 <urllist>
  <name>Second</name>
  <description>Use Case 2</description>
  <url>http://localhost:8080/site/</url>
  <url>http://localhost:8080/site/httpd_logo_wide.gif</url>
  <url>http://localhost:8080/icons/apache_pb.png</url>
  <url predelay="30">http://localhost:8080/site/test2.html</url>
  <url>http://localhost:8080/site/oscon2002_lg_ad.gif</url>
  <url>http://localhost:8080/site/hotel_pool.jpg</url>
</urllist>
```

More Gratuitous Flowcharts

- User hits main page then downloads a big file:



More Gratuitous Flowcharts



Sample XML Configuration

```
 <urllist>
  <name>Third</name>
  <description>Use Case 3</description>
  <url>http://localhost:8080/site/</url>
  <url>http://localhost:8080/site/httpd_logo_wide.gif</url>
  <url>http://localhost:8080/icons/apache_pb.png</url>
  <url predelay="15">http://localhost:8080/site/flood.pdf</url>
</urllist>
```

Set up a `profile`

- enable keepalive
- use the `relative_times` output format
- check for `200 OK`

Sample XML Configuration

```
 <profile>
  <name>SiteProfile</name>
  <description>Round Robin Configuration</description>
  <useurllist>Second</useurllist>
  <profiletype>round_robin</profiletype>
  <socket>keepalive</socket>
  <report>relative_times</report>
  <verify_resp>verify_200</verify_resp>
</profile>
```

A `farmer` to use the `profile`

- run for 300 seconds (5 minutes)

Sample XML Configuration

```
<farmer>
  <name>John</name>
  <time>300</time>
  <useprofile>SiteProfile</useprofile>
</farmer>
```

The one and only `farm`

- 10 Farmer Johns

Sample XML Configuration

```
<farm>
  <name>Bingo</name>
  <usefarmer count="10">John</usefarmer>
</farm>
```

Intermission

Demonstration

“The showdown: httpd-2.0 vs 1.3”

What are we testing?

- SSI performance
- static file performance (small images, html files)
- big file downloads

Building Apache httpd-2.0

Building Flood

Running Flood

Gathering realtime stats

Flood Output

What output does Flood produce?

- Raw format
- No automatic analysis
- Define custom output module
  - Flood is very extensible

What kinds of metrics does Flood capture?

- Just some basic metrics:
  - Requests
  - Time frame
  - Requests per second
- Whatever metrics you want
  - Flood is very extensible

What are the provided output formats?

- Controlled by the report tag in the profile section
- `relative_times`
- `easy`
- `simple`

Relative Times Report Format - Summary

Start	Connect	Write	Read	Close	Valid	ID	URL
123123123	123	123	123	123	OK	456	http://www.example.com/
123123706	123	123	123	123	OK	457	http://www.example.com/
123124289	123	123	123	123	OK	458	http://www.example.com/
123124872	123	123	123	123	FAIL	459	http://www.example.com/
123125455	123	123	123	123	OK	460	http://www.example.com/
123126038	123	123	123	123	OK	461	http://www.example.com/
123126621	123	123	123	123	FAIL	462	http://www.example.com/
123127204	123	123	123	123	OK	463	http://www.example.com/
123127787	123	123	123	123	OK	464	http://www.example.com/

- Prints out an entry for each URL that is hit
- Times given as microseconds from epoch
- All times relative to the first time listed
- If using HTTP keep-alive, connect and close times may be 0

Relative Times Report Format - Detail

Start	Connect	Write	Read	Close	Valid	ID	URL
123123123	123	123	123	123	OK	456	http://www.example.com/
123123706	123	123	123	123	OK	457	http://www.example.com/
123124289	123	123	123	123	OK	458	http://www.example.com/
123124872	123	123	123	123	FAIL	459	http://www.example.com/
123125455	123	123	123	123	OK	460	http://www.example.com/
123126038	123	123	123	123	OK	461	http://www.example.com/
123126621	123	123	123	123	FAIL	462	http://www.example.com/
123127204	123	123	123	123	OK	463	http://www.example.com/
123127787	123	123	123	123	OK	464	http://www.example.com/

- Absolute time
- Microseconds since the epoch (Jan 1, 1970)

Relative Times Report Format - Detail (cont.)

Start	Connect	Write	Read	Close	Valid	ID	URL
123123123	123	123	123	123	OK	456	http://www.example.com/
123123706	123	123	123	123	OK	457	http://www.example.com/
123124289	123	123	123	123	OK	458	http://www.example.com/
123124872	123	123	123	123	FAIL	459	http://www.example.com/
123125455	123	123	123	123	OK	460	http://www.example.com/
123126038	123	123	123	123	OK	461	http://www.example.com/
123126621	123	123	123	123	FAIL	462	http://www.example.com/
123127204	123	123	123	123	OK	463	http://www.example.com/
123127787	123	123	123	123	OK	464	http://www.example.com/

- Relative to `start` time in microseconds
- Time it took to connect
- Time to write the request
- Time to read the response
- Time to close the response

Relative Times Report Format - Detail (cont.)

Start	Connect	Write	Read	Close	Valid	ID	URL
123123123	123	123	123	123	OK	456	http://www.example.com/
123123706	123	123	123	123	OK	457	http://www.example.com/
123124289	123	123	123	123	OK	458	http://www.example.com/
123124872	123	123	123	123	FAIL	459	http://www.example.com/
123125455	123	123	123	123	OK	460	http://www.example.com/
123126038	123	123	123	123	OK	461	http://www.example.com/
123126621	123	123	123	123	FAIL	462	http://www.example.com/
123127204	123	123	123	123	OK	463	http://www.example.com/
123127787	123	123	123	123	OK	464	http://www.example.com/

- Response valid
  - `OK` or `FAIL`
  - Indicates whether verification was successful

Relative Times Report Format - Detail (cont.)

Start	Connect	Write	Read	Close	Valid	ID	URL
123123123	123	123	123	123	OK	456	http://www.example.com/
123123706	123	123	123	123	OK	457	http://www.example.com/
123124289	123	123	123	123	OK	458	http://www.example.com/
123124872	123	123	123	123	FAIL	459	http://www.example.com/
123125455	123	123	123	123	OK	460	http://www.example.com/
123126038	123	123	123	123	OK	461	http://www.example.com/
123126621	123	123	123	123	FAIL	462	http://www.example.com/
123127204	123	123	123	123	OK	463	http://www.example.com/
123127787	123	123	123	123	OK	464	http://www.example.com/

- Unique client ID
  - usually just Thread ID or Process ID
  - Allows aggregate results for each virtual user

Relative Times Report Format - Detail (cont.)

Start	Connect	Write	Read	Close	Valid	ID	URL
123123123	123	123	123	123	OK	456	http://www.example.com/
123123706	123	123	123	123	OK	457	http://www.example.com/
123124289	123	123	123	123	OK	458	http://www.example.com/
123124872	123	123	123	123	FAIL	459	http://www.example.com/
123125455	123	123	123	123	OK	460	http://www.example.com/
123126038	123	123	123	123	OK	461	http://www.example.com/
123126621	123	123	123	123	FAIL	462	http://www.example.com/
123127204	123	123	123	123	OK	463	http://www.example.com/
123127787	123	123	123	123	OK	464	http://www.example.com/

- Request-URI
  - w/o query string
  - Allows aggregate results for each page

Easy report format

- Identical to the relative times format
- Except all times relative to the epoch

Simple report format

- For each URL hit
  - Verification result: OK/FAIL
  - Request-URI
- As a summary
  - Tally of all response status codes

Measurement Tools

Performance analysis

- Highly OS-dependent
- Varies greatly by OS
- Linux `sysstat` tools: `http://perso.wanadoo.fr/sebastien.godard/`

truss/strace

- Solaris has `truss`, Linux and others have `strace` [a]
- Features:
  - Traces system calls
  - Attach to running processes
  - No kernel-level info

[a] FreeBSD and Darwin currently have no good way to trace system calls on a per-process basis except for at the kernel level with `kdump` and `ktrace`. This makes performance analysis quite challenging on these platforms.

The `stat` tools

- Huge range of system statistics
- `vmstat`
  - Memory and paging metrics
- `iostat`
  - Disk performance metrics
- `nfsstat`
  - network filesystem metrics

More `stat` tools

- `netstat`
  - Network subsystem information
- `mpstat`
  - Multi-Processor metrics
  - (locks, threads, semaphores)
- `systat`
  - Overall system metrics (FreeBSD)

sar

- long-running stats-gathering program
- collects database of information
- wide-range of system metrics
- disk I/O, network I/O, memory, etc.

snoop/tcpdump

- captures raw network traces
- `ethereal`[a] can provide further high-level analysis of the raw trace

- SSL poses a problem since the payload is encrypted

[a] http://www.ethereal.com/

tcptrace

- http://www.tcptrace.org
- Gives statistical information from network traces
- Tracks and graphs network metrics [a]
- Can reassemble TCP sessions
- (useful for feeding back into `flood`

[a] See: `xplot`

pstack

- Takes a process ID
- Dumps current stack from each thread
- Great for snapshots
- Solaris, FreeBSD...

JVM stack trace

- Profiles all JVM threads
- Useful with `pstack`
  - Use thread ids to correlate
- Send your JVM a `SIGQUIT` signal, it dumps to stdout/stderr

dummynet

- Use to emulate real-world networks
- Can add:
  - Random packet loss
  - Random delays
  - Bandwidth limiting
  - Fine-grain statistics
- FreeBSD

Result Analysis

Hints for dealing with all the data

- perl/awk/sed/grep/etc... are your friends
- Look for trends
- Rely on the statistics

Processing the data

- We’ve provided a couple scripts:
  - analyze-relative

    various averages for the output of a `relative_times` *report*
  - analyze-relative-ramp

    same as above, only deals with a ramp-up period and helps isolate slow pages

Visualizing the data

- gnuplot/xplot
- tcptrace[a]

[a] http://www.tcptrace.org/

Things to look for

- Bottlenecks/Capacity limits

  "No matter how many webservers we add we can't seem to handle any more SSL traffic."

- Failure points

  "As soon as we hit 100 concurrent users, out database fails."

- Over/under-utilized resources

Trends

- Unbounded resource consumption
- Periodic failures

Iterative Tuning

1. Identify problems
2. Propose solutions
3. Test them
4. Rinse, repeat

Future

What happens now?

- Graphical and non-graphical frontends to generate XML configs
- Raw data processing and analysis
- Take advantage of multiple client machines
- Multiple verification routines in parallel
- Automated profile generation from things like
  - `tcptrace`
  - `snoop`/`tcpdump` raw traces
  - common log format

Thank You

Appendix A: Sample XML Configuration Snippets

url

Sample XML Configuration

```
<url>http://httpd.apache.org/test/flood/</url>
```

urllist

Sample XML Configuration

```
<urllist>
  <name>SSL Hosts</name>
  <description>A bunch of SSL hosts we want to hit</description>
  <url>https://www.modssl.org/example/test.phtml</url>
  <url>https://mozilla-crypto.ssleay.org/cryptocheck.php</url>
</urllist>
```

farmer

Sample XML Configuration

```
<farmer>
  <name>Joe</name>
  <count>5</count>
  <useprofile>RoundRobinProfile</useprofile>
</farmer>
```

farm

Sample XML Configuration

```
<farm>
  <name>Bingo</name>
  <usefarmer count="25">Joe</usefarmer>
</farm>
```

collective[a]

Sample XML Configuration

```
<collective>
  <name>Borg</name>
  <usefarm count=4>Bingo</usefarm>
  <usefarm count=2>Pepperidge</usefarm>
</collective>
```

[a]The `collective` class hasn't been implemented, so this syntax is preliminary.

megaconglomerate[a]

Sample XML Configuration

```
<remotehost>
  <name>dualcpu</host>
  <host>dual.example.com</host>
  <username>tester</username>
  <proto>ssh</proto>
</remotehost>

<megaconglomerate>
  <name>TestLab</name>
  <usecollective host="dualcpu">Borg</usecollective>
  <usecollective host="quadcpu">Bubba</usecollective>
</megaconglomerate>
```

[a] The `megaconglomerate` class hasn't been implemented, so this syntax is preliminary.

profile

Sample XML Configuration

```
<profile>
  <name>RoundRobinProfile</name>
  <description>Round Robin Configuration</description>
  <useurllist>Test Hosts</useurllist>
  <profiletype>round_robin</profiletype>
  <socket>keepalive</socket>
  <verify_resp>verify_200</verify_resp>
  <report>easy</report>
</profile>
```

Appendix B: Other Performance Measurement Tools

What other tools are out there?

- Commercial
  - Microsoft Web Application Stress Tool
  - Empirix e-TEST
  - Mercury Interactive LoadRunner
  - SPECweb99
- Open-Source
  - ApacheBench (`ab`)
  - `httperf`
  - S-Client Architecture
  - `JMeter`

Microsoft Web Application Stress Tool

- http://webtool.rte.microsoft.com/
- Pros
  - Uses Internet Explorer as request engine
  - Automated recording
  - Centralized administration
  - Free
- Cons
  - Only available on Microsoft platforms
  - Poor SSL support
  - Not able to handle dynamic sites

Empirix e-TEST

- http://www.empirix.com/
- Pros
  - Automated recording
  - Centralized administration
  - Server-side collection via SNMP and plugins
  - Multi-platform support
- Cons
  - Expensive

Mercury Interactive LoadRunner

- http://www.mercuryinteractive.com/products/loadrunner/
- Pros
  - Automated recording
  - Centralized administration
  - Server-side collection via SNMP and plugins
  - Multi-platform support
- Cons
  - Expensive

SPECweb99

- http://www.specbench.org/osg/web99/
- Pros
  - Widely accepted by industry
- Cons
  - Not a test tool, just a benchmark

ApacheBench (ab)

- Included with Apache HTTP Server releases:

  http://httpd.apache.org/

- Pros
  - Potentially higher concurrency
    - Uses select()/poll() model
  - Free
- Cons
  - Only able to handle one URL

httperf

- http://www.hpl.hp.com/personal/David_Mosberger/httperf.html
- Pros
  - SSL support
  - Provides a framework
  - Free
- Cons
  - Primative multiple URL support

S-Client architecture

- http://www.cs.rice.edu/CS/Systems/Web-measurement/
- Pros
  - Produces a reliable steady stream of requests
  - Free
- Cons
  - Only able to handle one URL

JMeter

- http://jakarta.apache.org/jmeter/
- Pros
  - GUI front-end
  - Extensible
  - Free
- Cons
  - Accuracy sacrificed as scales up
  - Can not handle dynamic requests