So You Want High Performance

By: Peter Lin

Reviewers:
Tim Funk, Mike Curwen,
Remy Maucherat

Table of Contents

So You Want High Performance

Who doesn't? With clock work regularity, questions about performance appear on tomcat-user mailing list at least once a month. Sometimes more frequently, if a commercial server publishes new benchmarks claiming they out perform Tomcat. This article is my attempt to answer some of these questions and try to provide some useful tips, and tricks. Remy Maucherat and I co-wrote Tomcat Performance Handbook, but as many have heard Wrox went out of business. For those who don't know, Remy is a consultant with JBoss and the release manager for Tomcat 4 and 5. Originally, I wrote a couple articles to donate to the community, but before I completed them, Wrox made a book offer. Nine months later the book is still non-existent, but I still have an itch to scratch.

For this article, I ran a couple new benchmarks and use some data from the book. This article is a complete rewrite and does not plagiarize from the book. Plus, this way I can be more informal and add a bunch of jokes. Any and all errors are my mistake, so don't go asking Remy why certain things are wrong. If you find spelling, grammatical or factual errors, please email me at woolfel@yahoo.com and I'll fix them.

What to Expect

There's a thousand ways to talk about performance, but I've chosen to tackle it in the following order:

1. An address book webapp as an example
2. How it performs and tuning
3. Myths about performance and scalability
4. Development process is the problem not the server
5. Performance strategies

Many of the comments from the book reviewers wanted to see the results sooner than later. In my mind, any conversation about performance has to take into consideration the development process and functional requirements, but people find that boring. Ironically, that's where most projects fail. Many projects go through the development cycle without formal requirement documents, which leads to total chaos. I'm sure everyone is dying to learn how to write functional and performance requirements, but you'll have to wait. I know it's tough, since I've whet your appetite.

An Address book Webapp

I wrote this quick webapp to demonstrate the relationship between architecture and performance. It's not meant to be complete or usable, but address books are common features on many commercial sites. The webapp uses XML and JSTL so that readers don't have to install a database. Yeah, I know there are tons of free database on the Internet, but do you really want to spend 5 hours installing a database so you can run a webapp. Before you start screaming “XML has horrible performance! Why in the world did you choose it?” There's a good reason. XML and XML derived protocols are

becoming more popular and prevalent. For better or worse, applications that use XML will continue to increase. Binary protocols are generally faster than XML. For example, JDBC is faster than XMLSql drivers. In some cases, researchers have demonstrated that XML based protocols can match and beat RMI in terms of speed. Depending on which benchmarks you look at, binary protocols tend to perform 10-100 times better than XML. Just in case you want to see actual benchmarks, here's a list of URL's.

http://www.cs.fsu.edu/~engelen/soap.html
http://www2003.org/cdrom/papers/alternate/P872/p872-kohlhoff.html
http://www.extreme.indiana.edu/xgws/papers/soap-hpdc2002/soap-hpdc2002.pdf
http://xmlbench.sourceforge.net/index.php?page=results.php
http://easynews.dl.sourceforge.net/sourceforge/xmlbench/features.pdf
http://www.sosnoski.com/opensrc/xmlbench
http://www.ingorammer.com/Articles/REMOTINGVS.ASP.NETWEBSERV.html
http://riboe.homeip.net/articles/ws
http://www-staff.it.uts.edu.au/~rsteele/irl/wsrdg/kohlhoff.pdf

The address-book webapp uses a combination of servlets and JSP's with three different data models to illustrate the impact of functional requirements on the architecture. The architecture ultimately defines the performance limits of your webapp.

Installing the War file

Download the war file and put it in your tomcat/webapps/ directory. By default Tomcat will auto deploy the application on startup. You can manually install the webapp by extracting the files to the webapps directory with “jar -xvf addrbook.war”. The webapp is organized into three directories:

```
webapps
   |
   | --addrbook
            | -- jsp files
            | -- data
                  | -- xml data
            | -- random
                  | -- xml data used for random selection
            | -- web-inf
                  | -- JSTL, servlets and other stuff
```

If you look at the JSP files, you will notice the files are named simple_100.jsp, simple_500.jsp and so on. The basic idea was to simulate a flat data model versus a normalized data model.

Simple – equivalent to flat file or single database table
Medium – semi-normalized model
Complex – normalized model

Features

The site only has a few features:

1. Display all the addresses in the XML file
2. Search for an address
3. Randomly select a data file and display the contents
4. Use SAX with XSL/XSLT to transform the data to HTML
5. Use DOM with JSTL to transform the data to HTML
6. A modified version of the stock snoop.jsp page for testing GZip

Feel free to modify the webapp as you wish. If you decide to use it for a real site, I take no responsibility and provide no warranty. It's not appropriate for real use and is only for benchmarking purposes.

How it performs and tuning

I used a combination of Apache Bench and Jakarta Jmeter to load test the webapp on my servers. Being the geek that I am, I have a couple of development servers. Each test ran for a minimum of 1000 requests and each run was performed several times. The Jmeter test plans are included in the resource package. The systems used to perform the benchmarks are listed below.

- System 1
  Sun X1 400mhz UltraSparcIIe
  768Mb ECC Reg PC 133 SDRam
  Tomcat 4.1.19
  Sun Jdk1.4.1_01
  JMeter 1.7
  Apache 2.0 ab
- System 2
  Red Hat 8.0 server
  AMD XP 2ghz
  1Gb DDR Ram
  Tomcat 4.1.19
  Tomcat 5.0.9
  Sun Jdk1.4.1_01
  IBM Jdk1.4
  JMeter 1.7
  Apache 1.3 ab
- System 3
  Windows XP Pro
  Sony Vaio FX310
  900mhz Celeron
  256Mb Ram
  JMeter 1.8
  Sun Jdk1.4.1_01
- System 4
  Home built System

450mhz P3
512Mb PC100 SDRam
JMeter 1.8
Oracle8i release 2
Sun Jdk1.4.1_01

- System 5
  Home built System
  200mhz Pentium Pro
  256Mb SDRam
  Oracle8i

I chose to use this setup, because that' the hardware I have. If you want to know how the webapp would perform on a Sun E450 or E6800, try it and send me the results. If anyone wants to donate a Sun E4500, I'm always willing to adopt a server. The network setup consists of 2 linksys 10/100 switches. Both the X1 and Linux box have two 10/100mb Ethernet cards. All network cables are CAT5, except for a cross over cable between my router and one of the switches. The network traffic was strictly on the switches, so the router doesn't have any impact. All of the raw data generated by me are available, so feel free to scrutinize it. I didn't do an exhaustive analysis of the data, but feel free to do it yourself. In all cases, Apache Bench and Jmeter did not run on the server machine. When the X1 is the server, the linux system was running AB/Jmeter. When the linux box is the server, the X1 was running AB/Jmeter. I must acknowledge Sarvega corporation for their assistance. They were kind enough to run some benchmarks using their XML accelerator. You can find out more about the Sarvega XPE accelerator at http://www.sarvega.com/.

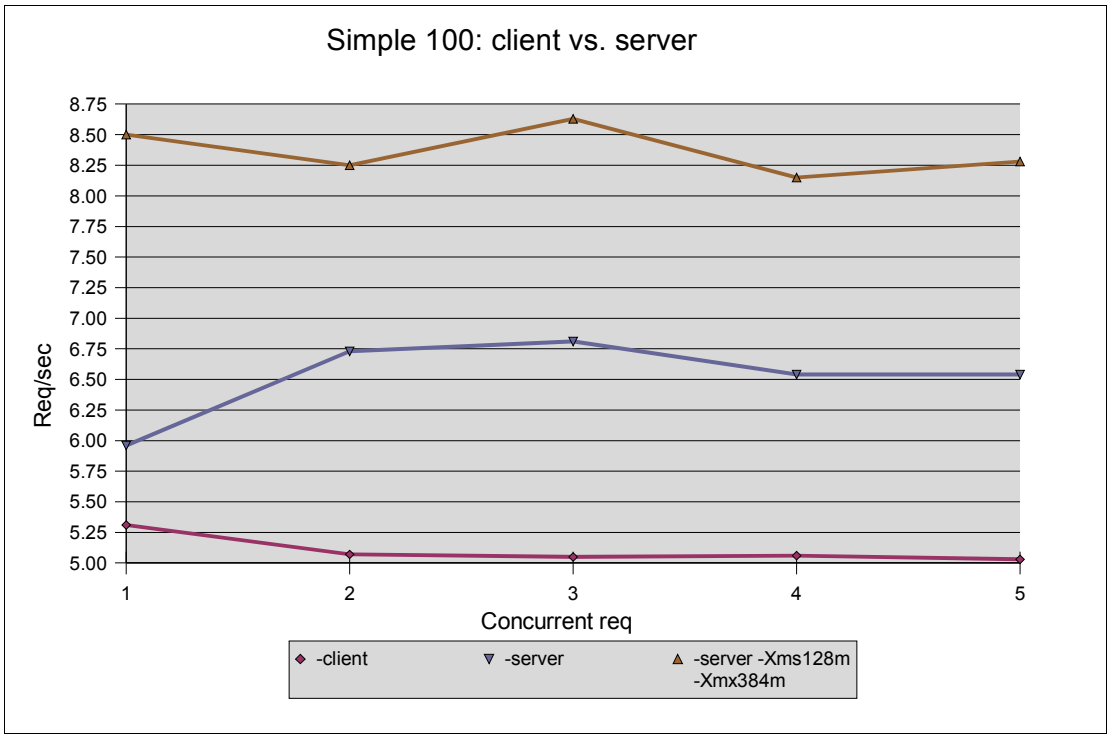
Enough Jibber Jabber Already, Show me the Data

This first graph shows the performance difference between client and server mode. For those not familiar with server and client mode, the default mode is client. To run tomcat with server mode, you have to modify catalina.bat/catalina.sh to include JAVA_OPTS.

```
JAVA_OPTS=”-server”
JAVA_OPTS=”-server -Xms128m -Xmx384m”
```

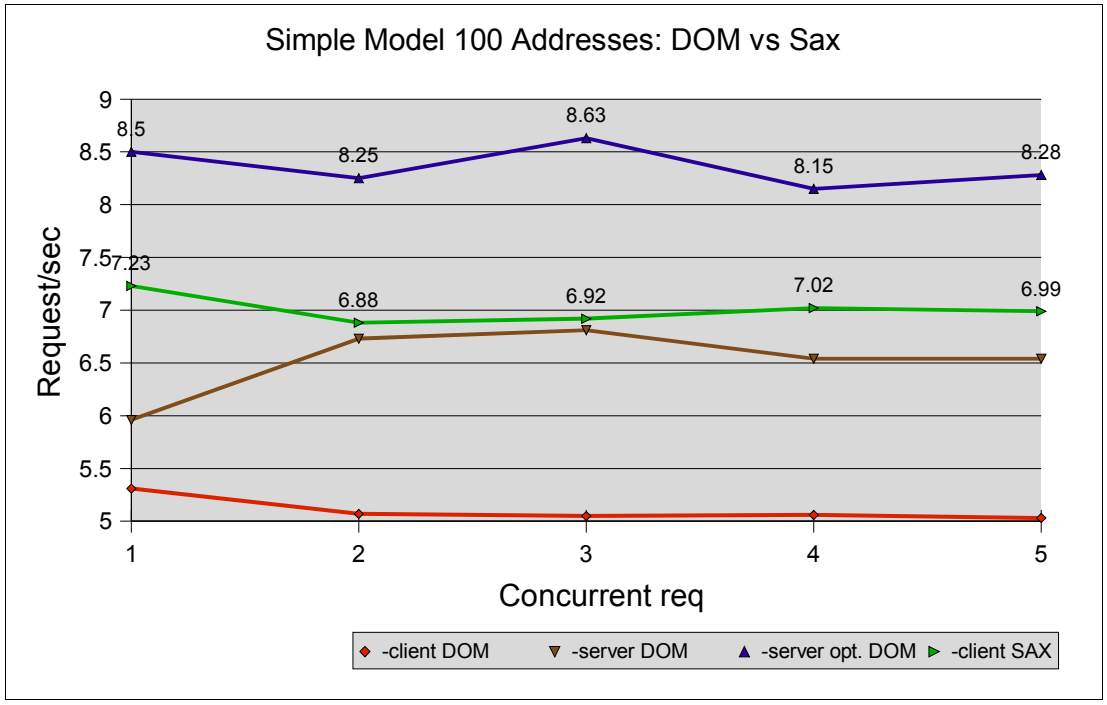
The servers were warmed up before the real test, so page compilation is not an issue.

Concurrent req	-client	-server	-server -Xms128m -Xmx384m
1	5.31	5.96	8.5
2	5.07	6.73	8.25
3	5.05	6.81	8.63
4	5.06	6.54	8.15
5	5.03	6.54	8.28



Graph 1: Simple 100 client vs. servers

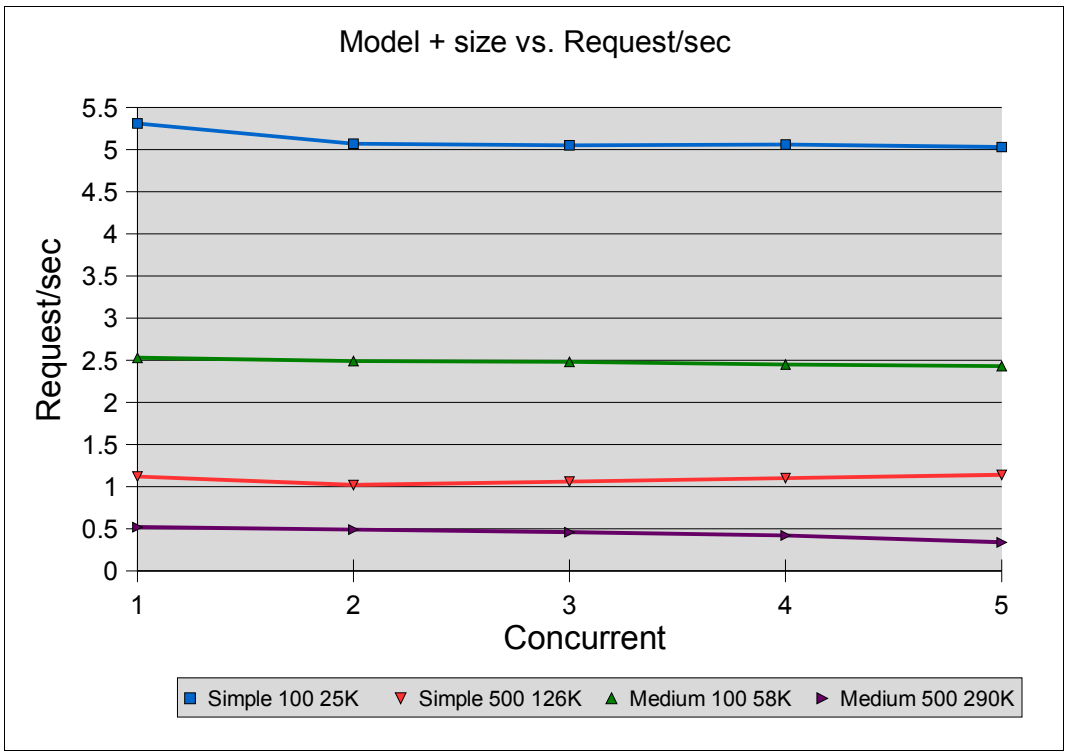
Wait a minute, are you serious? Unfortunately, the numbers in graph 1 are correct, since I ran the tests over a dozen times. If you happen to skip the first 3 pages, XML is heavy weight and doesn't come free. Parsing XML consumes copious amounts of CPU and memory, so it's kinda like a hungry hungry hippo for those who remember the game.



Graph 2: Simple 100 DOM vs. SAX

Concurrent req	-client DOM	-server DOM	-server opt. DOM	-client SAX
1	5.31	5.96	8.5	7.23
2	5.07	6.73	8.25	6.88
3	5.05	6.81	8.63	6.92
4	5.06	6.54	8.15	7.02
5	5.03	6.54	8.28	6.99

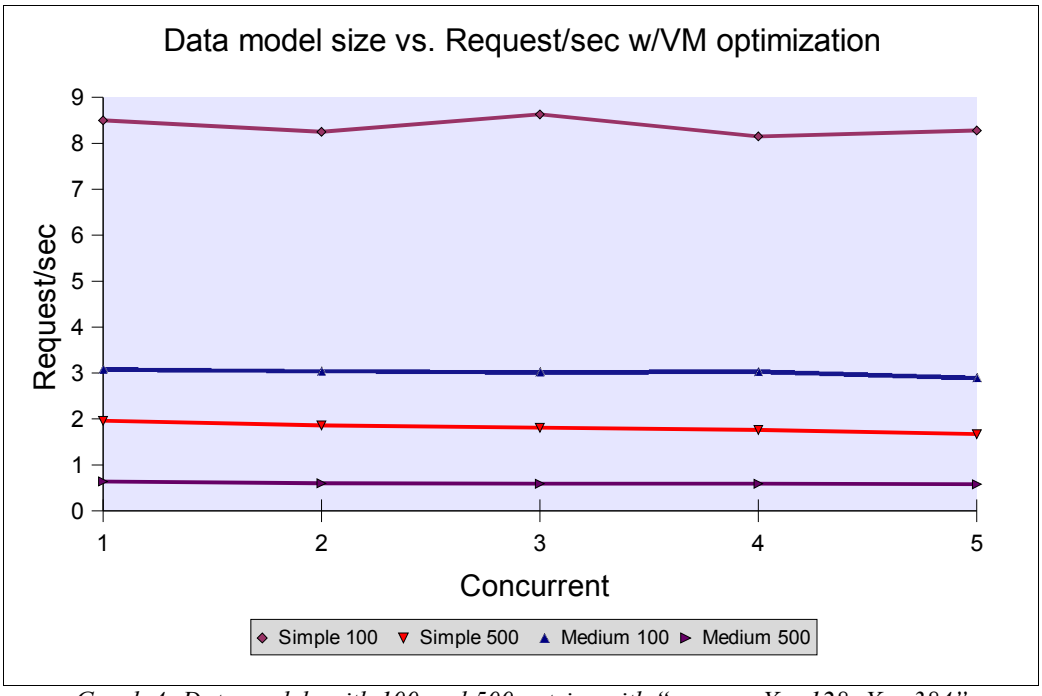
Graph 2 compares SAX to DOM. Using SAX helps client mode beat server mode, but optimized server mode still beats SAX. An important consideration between using DOM and SAX is memory and CPU utilization. In all tests comparing DOM to SAX, the CPU and memory usage was higher for DOM. Excessive CPU and memory utilization could impact other processes on the system, so use DOM carefully.



Graph 3: Simple and medium model with 100 and 500 entries

Concurrent	Simple 100 25K	Simple 500 126K	Medium 100 58K	Medium 500 290K
1	5.31	1.12	2.53	0.52
2	5.07	1.02	2.49	0.49
3	5.05	1.06	2.48	0.46
4	5.06	1.1	2.45	0.42
5	5.03	1.14	2.43	0.34

When the data is more than 100K, the throughput drops dramatically. The take home from this graph is limit the size of XML data. Performance intensive applications that load a lot of data should not use XML, unless you want to make sure the application is slow. In general, I would limit XML data to 2K or less for performance sensitive applications. Applications that have moderate load will want to limit XML data to 20K.

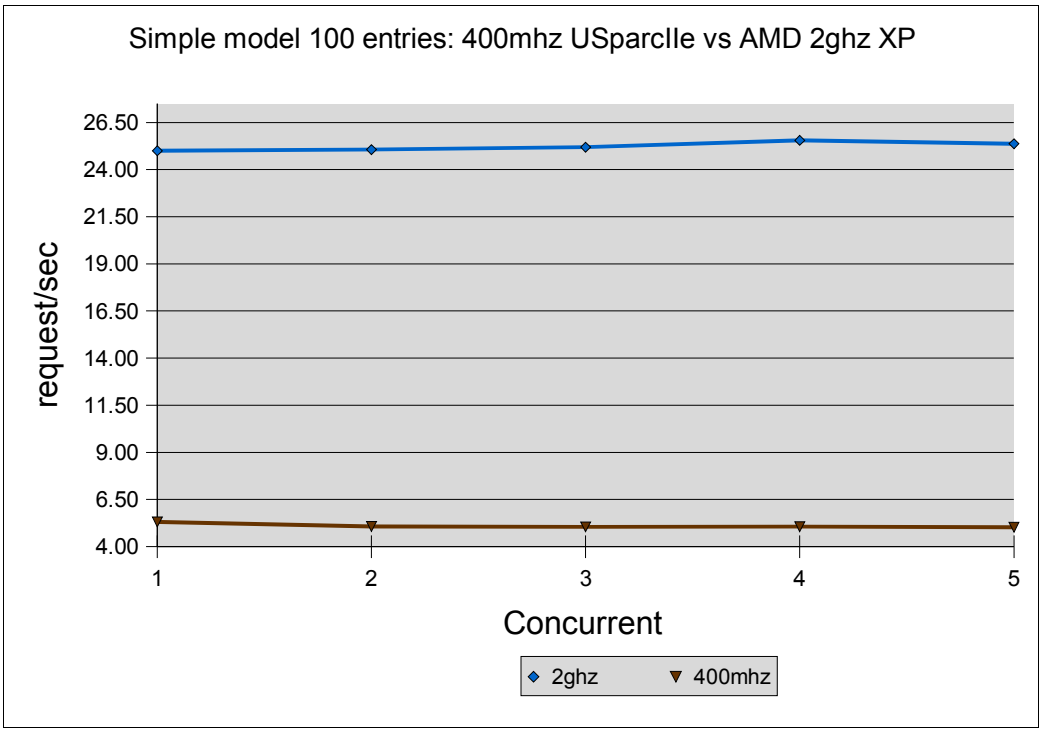


Graph 4: Data models with 100 and 500 entries with "-server -Xms128 -Xmx384"

Concurrent	Simple 100	Simple 500	Medium 100	Medium 500
1	8.5	1.96	3.08	0.64
2	8.25	1.86	3.04	0.6
3	8.63	1.81	3.02	0.59
4	8.15	1.76	3.03	0.59
5	8.28	1.67	2.9	0.58

Although running Tomcat with optimized heap increases the throughput from 5 to 8 requests per second, it's nothing to shout about. Since I have a Linux AMD box with 2ghz CPU and 1Gb of RAM, we can compare the throughput on a faster CPU.

Concurrent	2ghz	400mhz
1	25	5.31
2	25.06	5.07
3	25.19	5.05
4	25.55	5.06
5	25.37	5.03

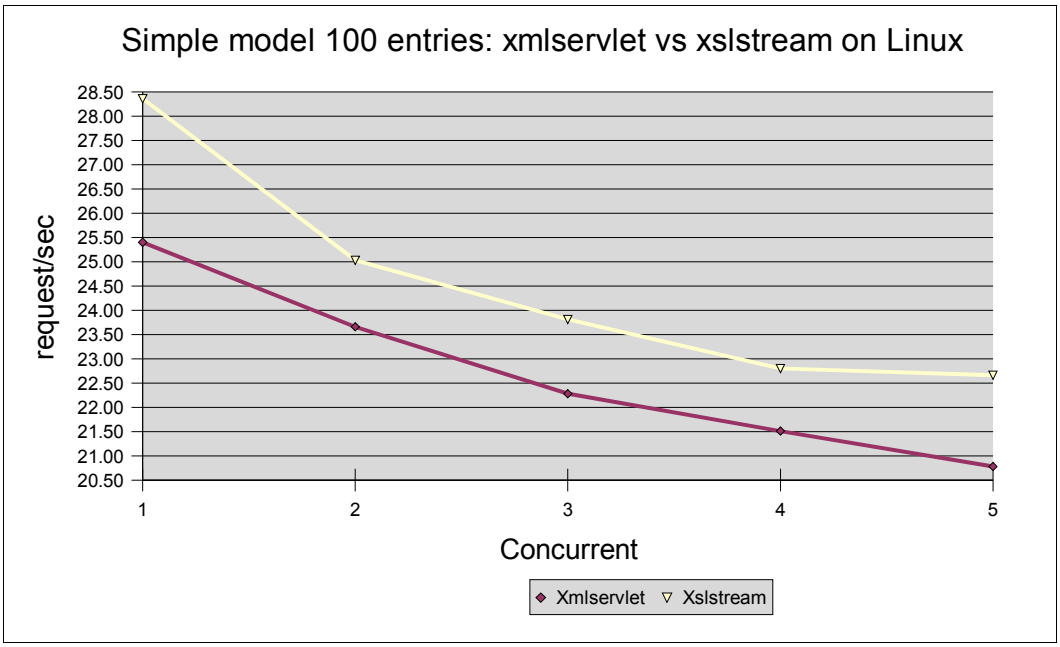


Graph 4: Simple model 100 entries 400mhz UltraSparc vs. AMD XP 2Ghz

Well the numbers speak for themselves. Let's put this in perspective. The AMD box cost me approximately 850.00 in 2002 and the X1 cost me 1150.00 in 2001. Both systems are 1U rackmounts, but the X1 runs much cooler. In fact, I had to drill some holes in the top of the AMD system, because the heat was causing it to lockup. Obviously, the heat isn't a great issue and can easily be solved with a 2U rackmount case and a couple more fans. In fact, a 2U configuration will probably be cheaper than a 1U depending on where you buy it from. Based on these numbers, increasing the CPU from 400Mhz to 2Ghz roughly quadruples the throughput per second. I did quick comparison between 400Mhz UltraSparc and 450Mhz Pentium III and the throughput was roughly equal. This is expected since XML parsing is CPU and memory intensive.

The next graph compares a servlet and JSP page that uses SAX with XSLT. What was surprising to me is the JSP was faster than the servlet. At first I thought this was a fluke and that something was wrong. To make sure this wasn't some bizzare anomaly, I rebooted both systems and ran the tests clean a half dozen times. When I made the X1 the server and the Linux box the client, I saw the same performance delta. I have no conclusive evidence for this behavior, but it might be the JSP mapper is more efficient than the servlet mapper. I had some discussions with Remy about this observation, but I haven't researched this any further.

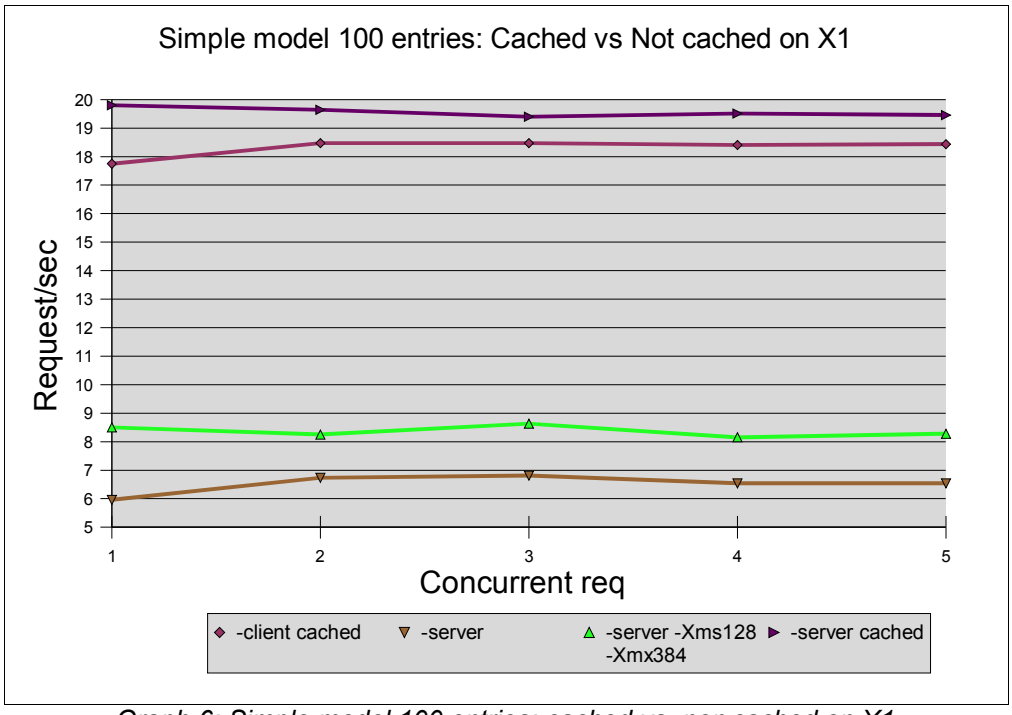
Concurrent	Xmlservlet	Xslstream
1	25.4	28.36
2	23.66	25.03
3	22.28	23.81
4	21.51	22.8
5	20.78	22.66



Graph 5: Simple Model 100 entries: xmlservlet vs. xslstream.jsp on Linux

So far the Linux box out performs the X1, so lets compare what happens if we cache the XML on the X1. Sadly, even with caching, server mode and increased heap, the Linux box is still faster (25 req/sec). In this particular case, scaling up with faster hardware is the better solution. Especially when you consider how cheap a 2Ghz system is today. Of course this isn't a solution for all cases. If you have 2000 servers, using AMD or Intel will generate so much heat, it will cost you more in the long run. After a couple of months, any saving you got from cheaper hardware will be lost in cooling and power. What's worse is that cost won't go away and stays with you for the life time of the setup. In fact, Google made a statement to that effect when they were asked “Does Google plan to upgrade to Pentium 4 systems?” The last time I checked, Google was using 4000 servers. Just think about how many kwH Google consumes each day. If you had to rent that much rack space, it would cost you 20-100 thousand dollars a month. The last time I researched co-location prices for 20 servers, it came out to 3 full racks at 5000.00 a month. A couple of the servers were 6U and the RAID boxes took up half a rack.

Concurrent req	-client cached	-server	-server -Xms128 -Xmx384	-server cached
1	17.75	5.96	8.5	19.8
2	18.47	6.73	8.25	19.64
3	18.47	6.81	8.63	19.4
4	18.41	6.54	8.15	19.51
5	18.44	6.54	8.28	19.46

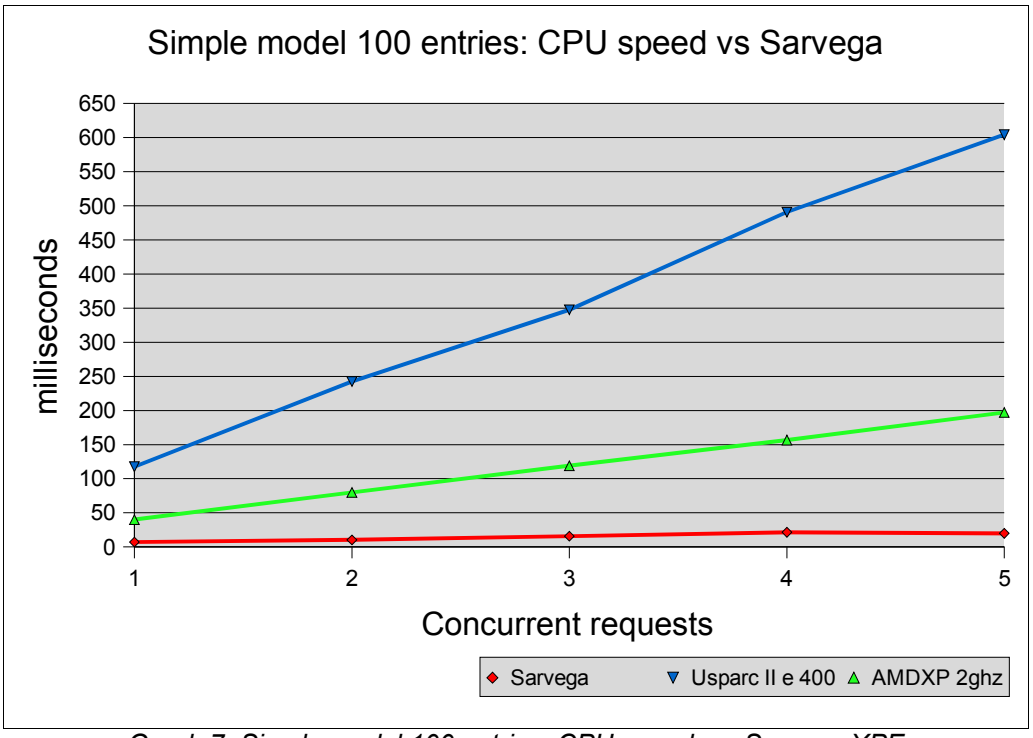


Graph 6: Simple model 100 entries: cached vs. non cached on X1

By now, you should be thoroughly convinced that XML performance sucks. But there's hope. A couple companies have begun selling XML accelerators and it does work. If we take a trip back to 1996, we can see a similar pattern. Back when I was in college, Amazon.com opened it's doors on the Internet. A year after that, SSL/TLS became an industry standard. As E-Commerce grew, many sites had a hard time stabilizing their servers, because encryption is CPU intensive. Companies like IBM, Rainbow Technologies and nCipher began selling SSL enabled Ethernet cards and routers. The largest sites rapidly deployed hardware accelerators to improve their performance and it worked. Not only did the response time improve, but so did reliability.

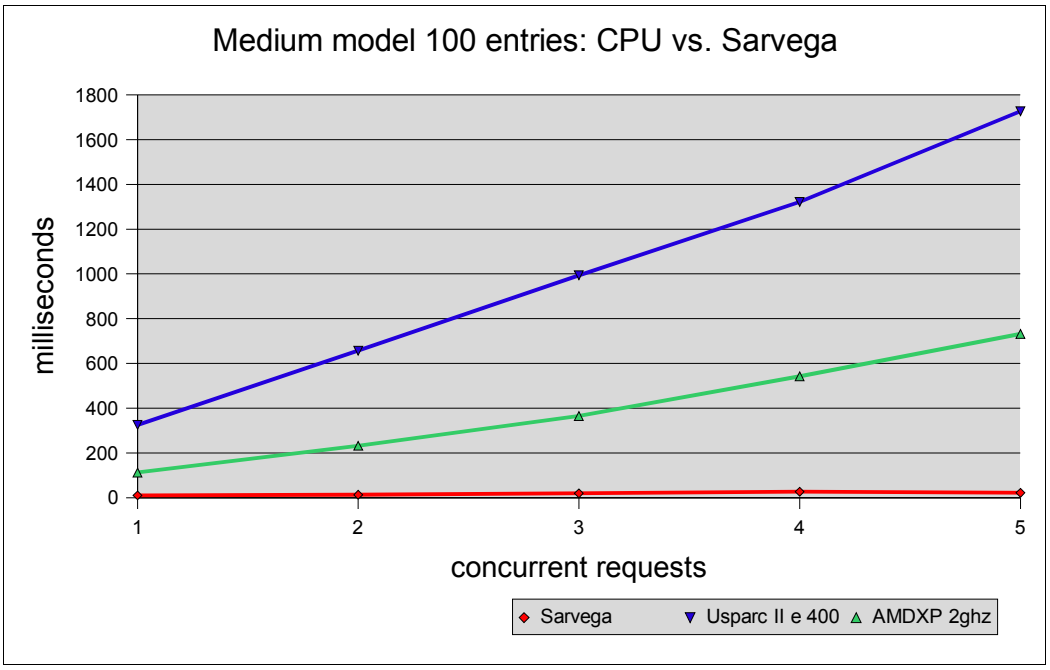
XML is going through the same growing pains. Even if some one writes a better stream parser, it's not going to beat a hardware solution. So how much faster are hardware accelerators?

Users	Sarvega	Usparc II e 400	AMDXP 2ghz
1	7.12	117.64	40
2	10.38	242.38	79.82
3	15.84	347.55	119.08
4	21.47	490.73	156.58
5	19.85	604.22	197.08



Graph 7: Simple model 100 entries: CPU speed vs. Sarvega XPE

For 25K of XML, Sarvega's XPE blows the software solution away. As the graph shows, performance scales linearly so, I didn't run the tests with 10 or 100 concurrent requests. As the number of concurrent requests increases, the performance delta will grow.

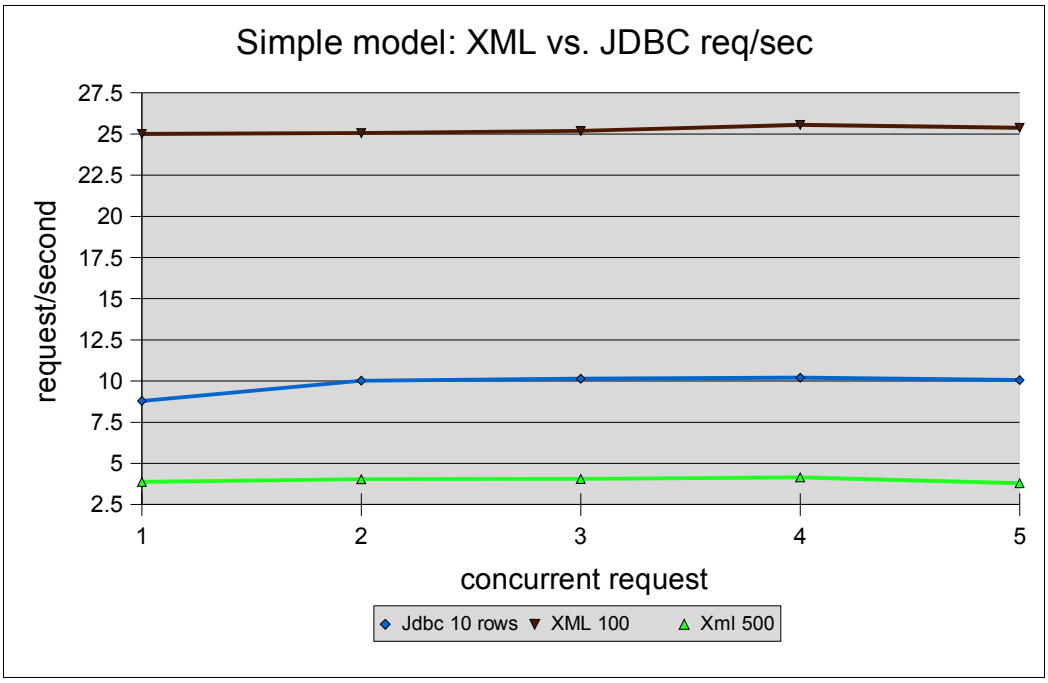


Graph 8: Medium model 100 entries: CPU vs. Sarvega

Users	Sarvega	Usparc II e 400	AMDXP 2ghz
1	10.21	325.13	112.73
2	13.16	656.88	232
3	19.54	993.53	365.21
4	26.61	1321.39	542.15
5	21.77	1726.48	731.45

With 58K of XML, the performance gain is greater as expected. We could go on and on increasing the size of the data to 1Mb, but it's obvious. Hardware accelerators provide concrete performance improvement. Some people will argue servers are cheap, so don't bother with hardware accelerators. I'll address this myth later in the article. For now, I'll just say that scaling horizontally is not as easy as it seems.

Ok, so XML sucks. Lets dump it for something more efficient, like JDBC and store the addresses in a database. At first, it might be surprising to see that JDBC is slower than XML. This is because of the high cost of making a database connection. Regardless of the database server, creating a connection and authentication takes 50-150 milliseconds.



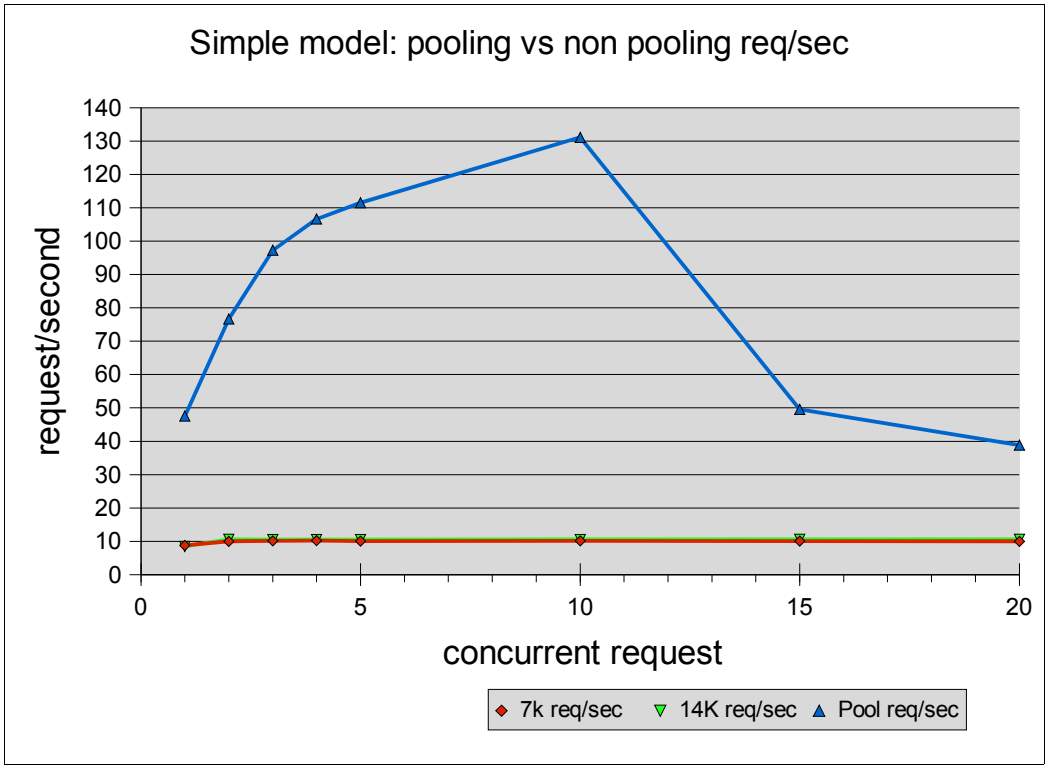
Graph 9: Simple model: XML vs. JDBC req/sec

	1	2	3	4	5
Jdbc 10 rows	8.79	10.02	10.14	10.2	10.06
XML 100	25	25.06	25.19	25.55	25.37
Xml 500	3.87	4.04	4.06	4.15	3.79

With connection pooling, the throughput easily beats XML and jumps to 100 requests per second. At 10 concurrent requests, we see a dramatic drop-off in throughput. This is primarily because Oracle is running on a 450mhz system with 512Mb of RAM and no

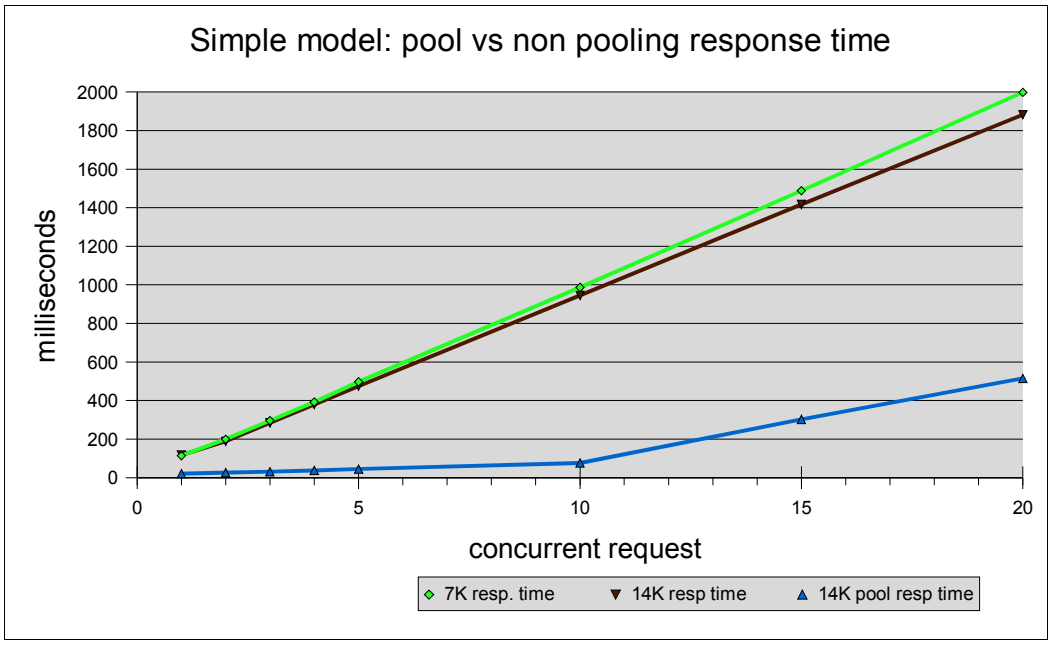
tuning. The database manager I wrote for the test application is simplistic and only tries to get a new connection once. Ideally, Oracle should be tuned to meet the needs of the peak traffic and the database manager should try to get a connection 3 times before throwing an exception. Another option is to make the database manager asynchronous, so that it can send the queries to Oracle once it's ready.

In most cases, tuning Oracle for high concurrency will be sufficient for small and medium sites. Oracle was chosen, because I that is the database I am most familiar with. Regardless of the database you use, it should be tuned to meet the needs of your website.



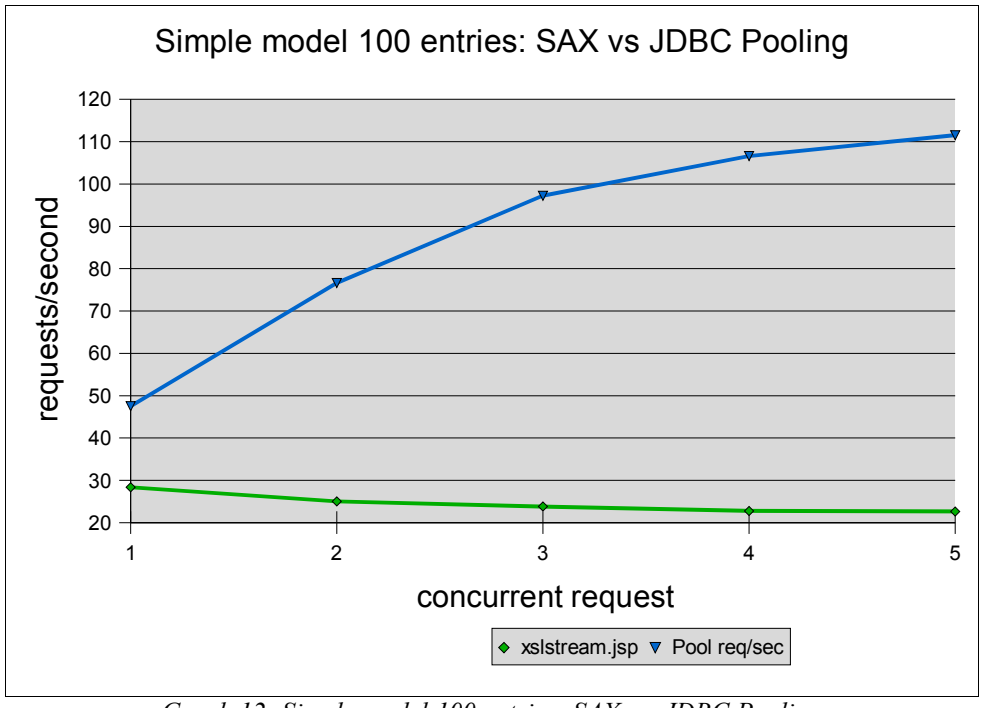
Graph 10: Simple model: pool vs. non pooling req/sec

	1	2	3	4	5	10	15	20
7K resp. time	113.81	199.62	295.85	392.17	497.14	986.96	1488.39	1998.4
14K resp time	117.33	188.44	284.56	379.18	474.11	945.2	1416.75	1882.68
14K pool resp time	21.03	26.1	30.85	37.52	44.83	76.26	302.58	514.76



Graph 11: Simple model: pool vs. non pooling response time

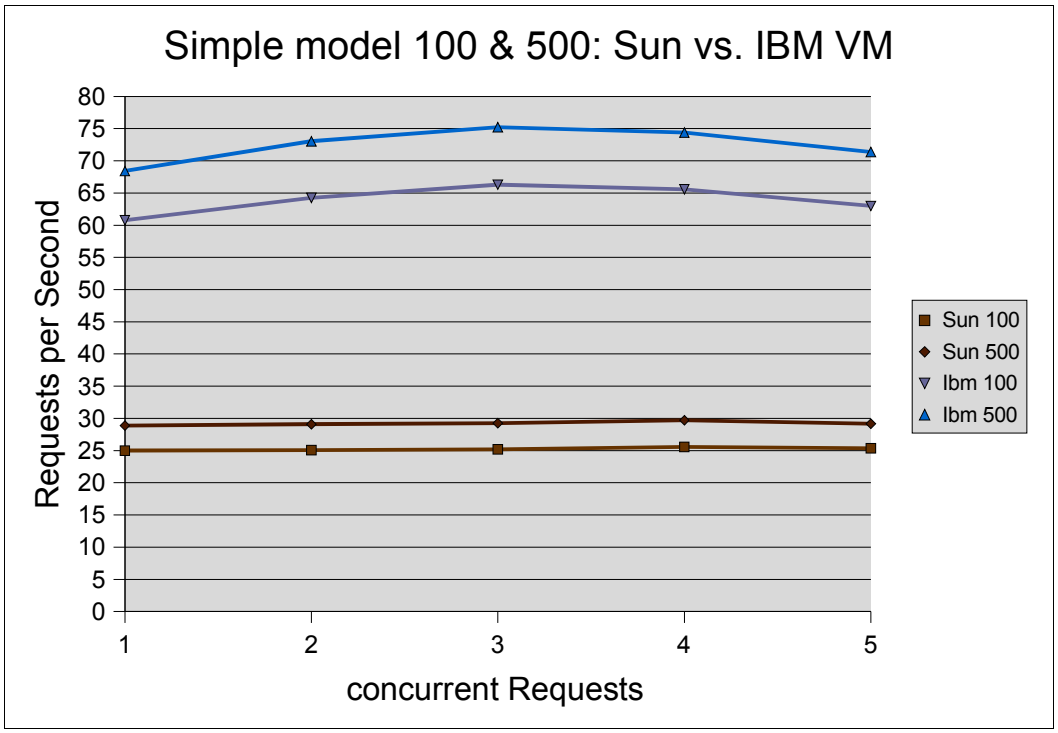
On a properly tuned database with connection pooling support, the response time should be linear until CPU utilization exceeds 85-90% or the system starts to hit the disk cache.



Graph 12: Simple model 100 entries: SAX vs. JDBC Pooling

	1	2	3	4	5
xslstream.jsp	28.36	25.03	23.81	22.8	22.66
Pool req/sec	47.54	76.62	97.24	106.61	111.53

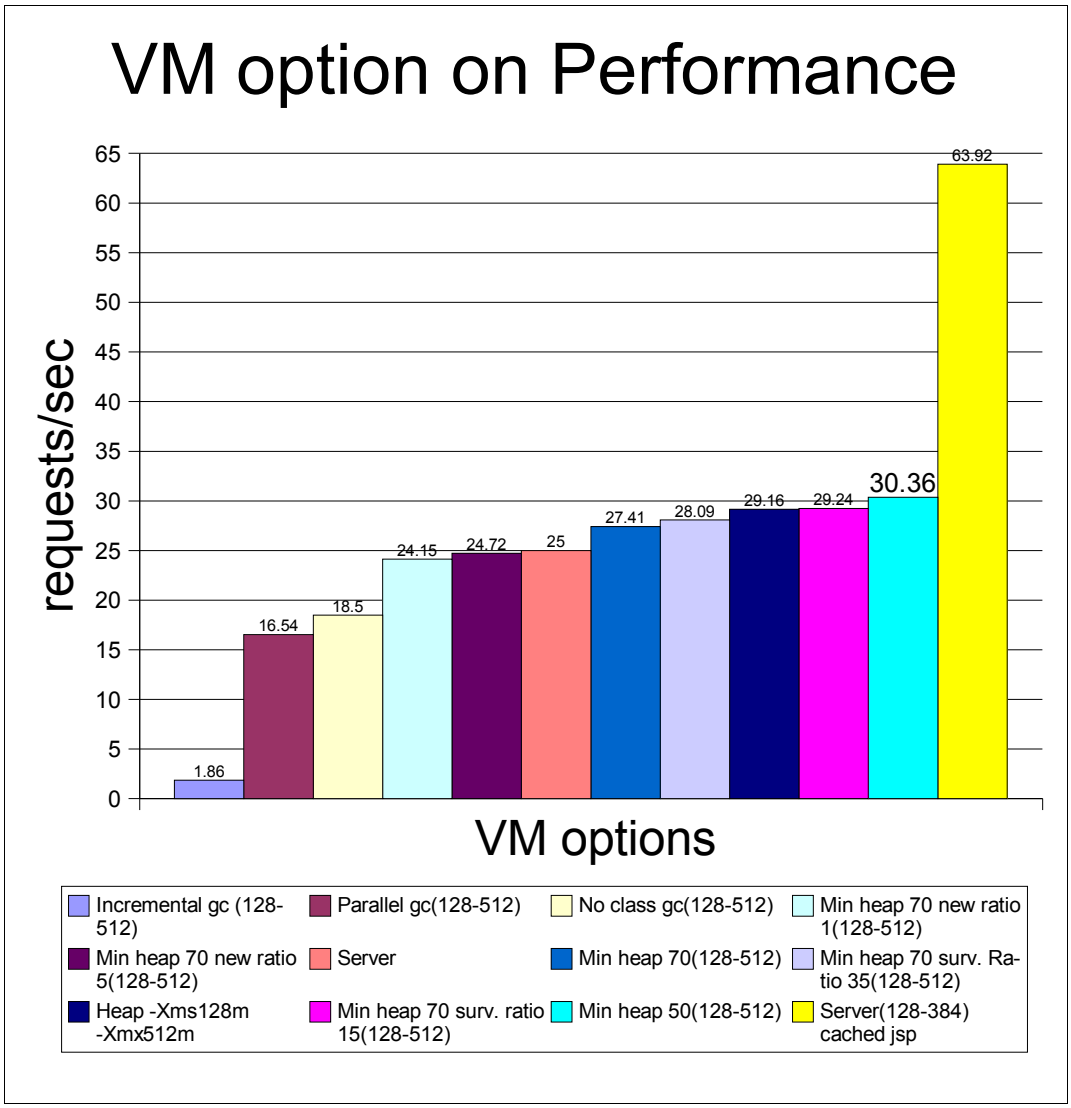
All the jdbc test used the second ethernet card and dedicated switch for database communication. With one ethernet card, the throughput dropped to 50 requests per second. Database intensive applications can increase throughput by adding a third ethernet card and dedicated switch. Just for kicks, I compared the results of SAX versus JDBC connection pooling. It's no surprise that connection pooling beats SAX by a wide margin.



Graph 13: Simple Model 100 & 500: Sun vs. IBM VM

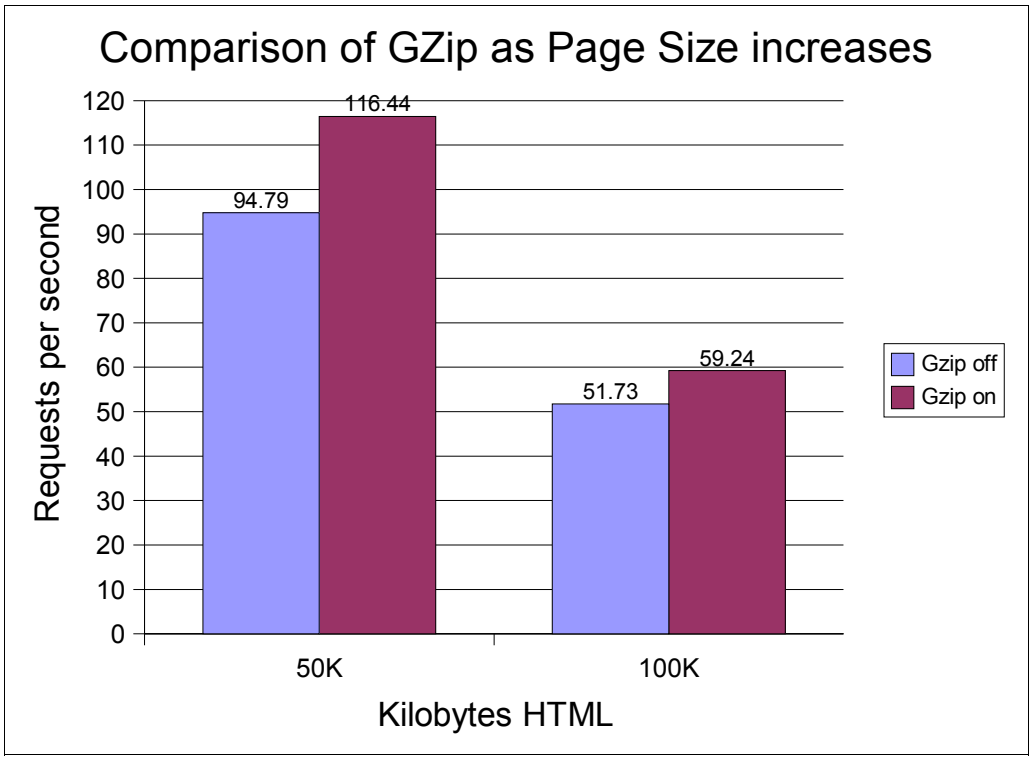
	Sun 100	Sun 500	Ibm 100	Ibm 500
1	25	3.87	31.89	7.66
2	25.06	4.04	35.14	8.8
3	25.19	4.06	37.05	8.92
4	25.55	4.15	35.87	8.82
5	25.37	3.79	33.83	8.38

Using the simple model with 100 and 500 entries, IBM VM is about 25-29% faster than Sun on Linux. Given that I'm using XML to benchmark the VM's, the difference in performance is probably the result of garbage collection. The results Remy produced for the book also show a similar deltas. I have no conclusive proof garbage collection is better in the IBM VM, but that's my guess. For those who use obscure extra VM options like tuning the eden, I ran a series of tests with different settings.



Graph 14: extended options for Sun VM

I used the cached jsp page as base of comparison for graph 14. The VM used for the test was Sun 1.4.1. In the case of XML, using parallel and incremental GC actually decreases the throughput. Although tuning the survivor ratio and new ratio can provide some improvement, the improvement versus time spent tuning may not be worth it. Depending on your application, tuning the extended options may show significant improvement. Application that are CPU and memory intensive shouldn't bother with new ratio and survivor ratio. Exploring ways to improve the application code or modify the architecture to cache data is more promising. Of course, not all applications have that option, so use your own judgement.



Graph 15: Gzip compression as Page Size Increases

Tim Funk thought it would be a good idea to compare the performance with and without Gzip compression. At a previous job, I worked on a Gzip request filter, but Coyote now supports Gzip thanks to Remy. Using Gzip compression on 50Kb pages produces a 20% improvement in throughput. That may not seem like much, but the observed behavior from the browser is tremendous. For example, Verizon Superpages.com now uses Gzip compression on most of their pages. The download speed over the internet is now 2-3 times faster and the amount of bandwidth used is reduced by half or more. Most of the page compress 6-10 times, depending on the page. The listings page compresses 10 times, whereas the details page is reduced 6-8 times. For the benchmark, I used Mozzila to compare the page size. With compression on, 50Kb went down to .77Kb and 100Kb went down to .96Kb.

Myths about Performance and Scalability

First thing about performance is this. “Persistence, persistence, persistence!” Achieving high performance is not a finish line. It takes a lot of hard work and constant monitoring to maintain high performance. Contrary to what some companies claim, high performance isn't easy and it's never as simple as point and click. As you might have guessed by now, architecture is the primary predictor of performance. As an example, if someone decides their entire architecture should use webservices, they must accept that it will never match the response time and scalability of systems that use binary protocols. There are several myths I want to address.

1. Bandwidth is unlimited
2. Any ISP will do
3. Scaling horizontally is preferable to vertical
4. You need the fastest webserver to get best the performance

5. A website doesn't need much maintenance

Bandwidth is Unlimited

I've heard this from people on mailing lists, in person and on IM. It simply isn't true and never will be true. What some people don't realize is that allocating more bandwidth moves at human speed, not machine speed. Lets review WAN technologies and dispel some myths about each one.

T1 – 128kbits to 1.5mbit

This definition used to be clear, but these days some sales people are calling DSL and broadband as T1. A true T1 line is a copper trunk containing 24 telephone lines. Most small and medium businesses with a T1 usually use the T1 for voice. Most businesses with a T1 use DSL for data, since that is what providers recommend. T1 typically takes a minimum of 3 months to install and takes a couple weeks to stabilize. Bottom line; don't expect installation to happen in a week or a month. No matter what the sales person tells you, it takes months to install.

T3 – 3mbit to 45mbit

T3 lines are also known as DS3. It is equivalent to 672 telephone lines or 28 T1 trunks. The main difference between T1 and T3 is cost. T3 lines can cost up to $15,000.00 per month. That's not a typo by the way. Normally, large corporations, or a high rise that has 80 thousand square feet of office space use T3. Just to give you an idea, a 20 story high rise typically will have a T3 or a half dozen T1. Your typical 3-4 story office building will probably have 2-3 T1 lines.

DSL – 128kbits to 10mbits

DSL stands for digital subscriber line. There are a lot of types of DSL: ADSL, SDSL, and VDSL. Unlike T1, DSL uses one telephone line, and has a lot of limitations. Most ISP todays offer Asynchronous DSL. It's called that because the download speed from the Internet is faster than the upload speed. Most business users fit this category. ADSL is not really meant for hosting websites. Symmetric DSL has the same download and upload speed. In many ways, this is very similar to T1, but it also suffers from the same limitations of ADSL. Both ADSL and SDSL have a limit of 3 cable miles. There has been a lot of talk about VDSL, a faster version of DSL, but I don't know anyone offering it. In theory, VDSL can reach speeds up to 10mbit. Unless you're within 1 cable mile, the typical bandwidth is 384-768kbits. Within the first cable mile, you can get as high as 2mbit bandwidth with ADSL/SDSL.

Frame Relay – 56kbits to 1.5mbits

Frame relays use shared lines and are similar to T1. Frame relay uses a different protocol than T1 and has the ability to provide bandwidth on demand. But there is a catch. Since the lines are shared by multiple customers, getting bandwidth on demand rarely works as advertised. Unless the line between you and the phone company is dedicated, don't expect your bandwidth to double. Most likely the ISP has sold more than half the physical bandwidth on the shared line, so at best you will only see a slight improvement. Some ISP will use frame relays if they have T1 lines going to their offices. This way they can allocate the bandwidth of their existing T1 appropriately.

OC1 – up to 51mbits
OC1 uses fiber optics instead of copper trunk lines. It is roughly equal to a T3 and the prices range from $10,000-20,000 per month. Unless you're an ISP, major corporation, or phone company, the cost is prohibitive.

OC3 – up to 155mbits
OC3 also uses fiber optics and is similar to OC1. Unless your pocket is deep, don't expect ISP's to give you a quote. Getting an OC3 line will require a team of hardhats and takes months to install. To my knowledge, only backbone providers have access to OC3 lines. Even if the office building is next to an existing OC3 line, don't expect the phone company to open the faucet for you.

For those unfamiliar with megabits and what it means, let's convert it to kilobytes per second. 1 byte is equal to 8 bits. If we take that equation and do a simple calculation, we get the following:

T1 – 187.5 kilobytes per second max
T3 – 5.25 megabytes per second max
VDSL – 1.25 megabytes per second max
SDSL – 187.5 kilobytes per second max
Frame Relay – 187.5 kilobytes per second max
OC1 – 6.375 megabytes per second max
OC3 – 19.375 megabytes per second max

In reality, you won't get the maximum. Because of Internet congestion, packet collision, bad phone lines, and distance from the provider, the ideal throughput on a cable modem, and DSL is probably closer to 80-90% of the maximum. Now, lets do some simple math to see how a T1 translates to page views per second. Let's use the buy.com homepage as an example. For now, I will assume the images are cached in the browser to calculate the maximum requests per second a T1 can support. Pageview is the industry standard for measuring site traffic, so I will use that, since requests per second is not very useful.

187.5 Kbsec / 56 Kb per page = 3.35 pageviews per second

What do you mean 3.35 pageviews per second? That's actually an optimistic estimate. In reality, a percentage of the requests will need new images. For example, if I search for a product on buy.com, the browser will have to download the thumbnails. Therefore, the total kilobytes sent for those types of requests could be 2-3 times (100-150Kb). It's easy to see how page size affects the number of requests per second the bandwidth can handle. Chances are the bandwidth will die long before Tomcat does.

Any ISP Will Do

Now that we've covered the types of connection available, lets see how it applies to ISPs. Service providers belong to one of three tiers.

Tier 1: Backbone Provider
Backbone providers are large telecommunications companies who own their own lines

and networks. The major players today in this area are AT&T, Verizon, Level3, MCI-Worldcom and SBC. Even backbone providers don't have unlimited bandwidth. You can expect multiple OC12 lines for backbone providers.

Tier 2: Regional

Regional providers include tier 1 providers. With the current economy, many tier 2 providers have gone out of business. Tier 2 providers typically buy bandwidth from backbone providers and build out a second line for redundancy. The quality and size of their network varies from region to region. Typical tier 2 provider will have 50-100mbits of total bandwidth include backup lines per hosting facility.

Tier 3: Local

Local providers typically rent one or more T1. They're the mom and pop providers. Normally, tier 3 can't afford their own line, so they will buy bandwidth from one or two tier 2 providers. Some do buy bandwidth from tier 1, but that is the exception. Total bandwidth for tier 3 ranges from 10-50mbits.

Why should I care about the provider? Let's say the functional and performance requirements state the website has to support 100 pageviews per second and the average size of a page is 45 kilobytes. That would mean the actual bandwidth has to be 4.5 megabytes per second. Depending on providers in your area, a tier 2 provider may only have a single T3 connection. Chances are the provider has other customers using the bandwidth, which means they won't be able to support the requirements. Let's put that in perspective, before we move on.

100 pageviews/sec X 86400 sec/day = 8,640,000 pageviews/day

How many sites get 8 million pageviews a day? According to an old report on internet.com, USAToday.com had 2.4 billion pageviews for 2000 with an average of 9 million per day and a peak of 15.8 million the day after nation elections. Based on Verizon's own published statistics, Superpages.com gets 63 million searches a month. If I convert that to pageviews, it comes to 2.1 million per day. That is assuming each search only views one page. Based on my own experience, I tend to view 3-5 pages per search, so Superpages.com probably gets closer to 10 million per day. On the extreme end, yahoo gets 1.5 billion pageviews a day. The links for the articles are below.

Http://dc.internet.com/news/article.php/560421
http://www.directorystore.com/aboutus/about_verizon.jsp
http://public.yahoo.com/~radwin/talks/yahoo-phpcon2002.htm
http://www.theopenenterprise.com/story/TOE20021031S0001

Since the yahoo presentation at PHPCon listed the hardware, I'll use that to estimate the hosting requirements. Yahoo uses 4,500 server to serve up 1.5 billion pageviews each day. If we divide that by the number of seconds in a day, we get 17,361 pageviews per second. Assuming the load is distributed evenly across the servers, each server handles 3-4 pageviews per second per system. The presentation says the servers are distributed across 16 facilities, which means each site should have approximately 280 servers. The question you have to ask now is, “how many providers can handle that many servers?” The answer to that is tier 1 providers. Let's take the best case scenario and say the systems are all 1U rackmount servers in a 48U rack.

That means 6 full racks in the best case. In reality, some of the systems will be multi-CPU systems running a database, which means a 4U or 6U rackmount. I don't know the actual number of racks each facility has, but the minimum is 6 racks. How much does that cost? The last time I got a quote for 4 racks with Level3 was 5,000.00 per month. This leads into the next myth.

Scaling Horizontally Is Preferable To Vertical

In an ideal world, this is true, but we don't live in wonderland. When a company builds a website there are typically 3 options they can choose from.

1. Host it locally
2. Co-locate the servers
3. Rent servers from a provider

Most business don't set aside huge rooms for the servers; therefore adding 20 new servers isn't something that happens magically. If anything, it will require a lot of planning to make sure the site can scale horizontally. The biggest challenge of hosting it locally is the bandwidth will max out. Adding more bandwidth typically takes 3-5 months. Other hidden problems with hosting locally are zoning laws. Depending on the zone, installing a T3 or additional T1 lines may require lengthy paperwork and ultimately may not be possible. This isn't to say scaling horizontally won't work, it just takes a lot more work and patience than most people expect. For example, say I build a website using XML and webservices. I start with a dozen servers to distribute the load. As the traffic increases, I can add servers as long as there's room for it. At some point, I need a bigger generator, bigger room, more racks, and more routers. If on the other hand I get a XML accelerator, it may scale better in the long run. Based on the benchmark results for the Linux system, I can calculate the number of servers:

Concurrent requests / 20 = number of systems

For this estimate, I used 20 per system assuming the messages average 10 kilobytes and CPU usage average 60% during normal load. Each system still has to get data from the database and handle other processes, so I don't want XML to hog all the resources. Let's say my website provides an Online accounting system for small businesses. If I want to support 400 concurrent active users, I would need 20-25 servers. On the other hand, if I use an hardware solution like Sarvega's XPE, I would be able to handle the same amount of traffic with half the number of servers. Why is this more scalable? Say each server has a 400 Watt power supply and you want to make sure there's backup power for 8 hours. If I have 12 servers, I can get by with off the shelf UPS. With 25 servers, I need a gas generator and place to put it. I can't just put the generator in the server room, since it generates noxious fumes. Reducing the number of servers also makes it easier for the system administrators to manage. A critical part of managing the servers is making sure each release is published successfully and no unexpected bugs appear.

Co-locating the servers is typically the preferred choice for most companies. The benefits of co-location is provider has redundant power, fire suppression system, extra rack space and multiple connections for redundancy. There's a huge catch with co-

location and scaling horizontally. The ISP may not have enough room. The goal of a provider is to fill up all the racks with paying customers. An empty rack isn't making money, so unless you've purchased extra rack space for growth, chances are you can't drive up and add a couple servers. From first hand experience, it typically requires several months to change providers, or move the systems to a new set of racks. In worst case scenarios, the local providers will be completely full. In those situations, it means shipping the server to another city, and sending the administrators to install and configure the servers. For 25 servers, it should take two experienced system administrators a couple of days if nothing goes wrong. This is one of the hidden costs, which people don't expect.

Renting servers from a provider is the least flexible option and is typically tied to a set of configurations they support. Trying to scale in this configuration is a lost cause, so I don't recommend it at all. If performance is important to you, skip this configuration all together.

You Need The Fastest Web Server To Get The Best Performance

Most people know to ignore hype when it comes to movies or music, but for some reason, the same common sense is missing from the IT world. That's not really true, but if you look at the debates about .NET vs. Java, it would indicate common sense is missing. Another possible explanation is idiots scream louder on mailing lists and developer forums than experienced developers. The sales people making grand statements don't believe it, so neither should you.

By now, it should be painfully obvious the architecture is the most important part of high performance. Without that, the webserver means very little. If the application you're building is heavy on the database side, you're better off exploring the database options first. The key is to focus on the functional requirements and figuring out which ones are unrealistic and which are reasonable.

A Website Doesn't Need Much Maintenance

If you expect the business to grow and the traffic to increase, it is important to measure the traffic and keep an eye on it. In a well managed business, new hardware and rack space should be ordered when the average load reaches 75% of the maximum. Given the time needed to get new hardware, configure each system, ship it to the provider, install and setup the network takes 3-4 months, waiting until a problem appears is asking for trouble. I'll go into greater detail about how to monitor the systems later in the article. A simple thing like archiving the server logs and regular house cleaning is important.

Development Process Is The Problem Not The Server

From first hand experience, performance problems are the result of bad development process and poor communication. Having worked at several startup companies, the

biggest challenge to a well organized development process is clear functional and performance requirements. One would think functional requirements would be the top priority, but often that is the hardest part to nail down. Regardless of the type of development process, none of them will work without clear functional requirements. In fact, I argue any development process will work when the requirements are well documented and understood by the developers.

How do I write functional requirements? The first step is listening to the business guys and figuring out exactly what they want. I've seen engineers ignore business guys and others who totally rollover like a puppy. The approach I take is assume all my assumptions are wrong and ask specific questions. Often the responses won't answer the question completely, so I will rephrase the question until I get the information I want. This process may feel like drawing blood, but it's a necessary part of the process if you really want to succeed. I try to follow a couple simple rules:

1. Don't assume everyone thinks the same
2. Everyone answers the same questions with different responses and each person interprets the question a different way
3. Forget everything you know and think you know about the requirements when you start writing the functional specifications
4. No detail is insignificant
5. Paraphrase other people's response before analyzing it
6. On purpose, state an incorrect assumption and hope someone catches it

Number six was Tim's idea. I thought it was good, and added it. I can't stress listening and asking enough. Most projects are under tight deadlines to produce a marketable product, so the best way to manage the process is to get everyone on the same page. Telling others what to do rarely works. Getting everyone to focus on solving the problem does work and works consistently. That's the role of documentation and requirements.

How do I write performance requirements? The functional requirements should to be done before you write performance requirements. Assuming the functional requirements are known in sufficient detail, the next step is to build a simple prototype to test the performance under ideal and worst case scenarios. Let's use the online accounting site as an example. Build two small prototypes with and without XML. Use Jmeter to load test each one with an increasing number of concurrent requests. Once you have the results, create graphs showing the relationship between concurrent requests and CPU/Memory usage. Also, generate a graph showing the response time over concurrent requests. The graph will look similar to the XML benchmarks.

Once you get some good data and charts, write up a short summary and presentation. It's not programming, but if you want management on your side, you have to bite the bullet. Most likely, you'll have to repeat this process dozens of times before everyone agrees on a common set of performance requirements. Sometimes certain requirements are driven by external forces, so getting real numbers will help the management make a case. Your boss can't just ay “XML sucks,” to his superior. Having a solid presentation with graphs of the throughput with and without XML will make a compelling argument.

Once you have the functional and performance requirements document, the next step

is to schedule spot benchmarks at keys points. Each milestone should have a thorough set of benchmarks. Depending on the importance of a component, more benchmarks may be needed. For example, a database manager should be benchmarked with extreme loads to insure it creates connections efficiently and closes connections correctly. A bug in the database manager will have a serious impact on performance. The key to high performance is persistent attention to detail. It's better to run a quick benchmark, than say “Oh, it should be within the requirements.” Unless you test the code thoroughly, you don't know for sure.

Performance Strategies

There are a couple of things you can do to maintain high performance once you've built the application.

1. Monitor the servers
2. Log application performance
3. Generate nightly statistics
4. Document the performance boundaries
5. Make sure everyone involved in the decision making process reads the document
6. Send out regular summaries of the server load and performance
7. Write a formal plan for disaster recovery
8. Run through the disaster recovery plan so everyone knows what their tasks are
9. Assign administrators and developers as point man for key responsibilities
10. Have a staging environment to test new releases

Monitor the Servers

What should I monitor? Monitoring can be categorized as active and passive. Active monitoring is usually done by getting a webpage, or sending a query to the database. Passive monitoring is done several ways. Some popular types of passive monitoring include:

1. Reading access log files for last request
2. Have the application send out a status

Active monitoring is the more accurate than passive, but it puts load on the servers. Generally, hitting the webserver shouldn't be shorter than 5 minute intervals. With complex applications, that won't be enough.

Log Application Performance

Let's say your application makes a lot of sql queries. It is a good idea to log the elapsed time to execute all the queries and elapsed time to generate the final HTML page. Logging the performance of each query is too much, so most people generally don't do

that. My perspective is that low level components shouldn't be logged in production. Processes that communicate with the external world, or make remote calls should be logged.

Generate Nightly Statistics

Now that you're logging and monitoring the systems, reports should be generated nightly or at minimum weekly. This will tell you how traffic affects your systems and a pattern will emerge over time.

Document Performance Boundaries

This isn't the performance requirements document. It is a short document describing how much traffic the setup can support and when new hardware is needed. It can use information from the performance requirements document, but the goal is different. The site performance is a combination of the hardware, software, hosting facility and staff. Depending on your environment, adding hardware may only take a couple weeks.

Make Sure Everyone Reads The Documents

The documents are not meant to kill trees. Many programmers hate to read documents and find it a waste of time. Everyone in this case isn't the secretary. It is everyone who is directly responsible for the site and responsible for approving hardware purchases. The CTO, CEO, project manager, programmers, system administrators and network administrators should all read the documents.

Write A Formal Plan For Disaster Recovery

Disasters will happen. It's Murphy's law. The most common unexpected failures are hardware related. Things like harddrives and memory will go bad, so have a plan of attack. Be very specific in the instructions. It's not desirable to put RAID on all the webservers, so when one goes bad, the person responsible should have the instructions at their fingertips. The last thing a system administrator wants is to search for some piece of documentation at 3 a.m.

Run Practice Drills

Fire fighters practice drills so they stay sharp. The same is true for the IT staff. If no one reads the disaster recovery document or tries the procedure, there's no assurance the plan will work. For example, say a new build has a serious bug that crashes the whole system. If the disaster recovery plan is to rollback to the previous release, the system administrators should run through the rollback process at least once.

Assign People

It's a good idea to assign individuals as experts in specific areas. There should be alteast one developer assigned as the point man in case the system administrators need help diagnosing a problem. In small shops with only 2-3 people, everyone should know enough to jump in in a case of an emergency. In large projects with lots of staff, it's a good idea to have a backup person. That way people can go on vacation without worrying about getting called back.

Staging Environment

In today's economy, building a staging environment may be a tough sell, but it is very important. If your website has to meet Service Level Agreements (SLA), it's good to create a scaled down version of the production environment and run the release through regression and stress testing. Try to get the staging environment as close to production as possible. In the case of Tomcat, this means the same VM, OS, configuration and monitoring. Eventually, a problem will appear and everyone will have to drop everything to fix it. If there's a well defined deployment and release plan, tracking down the problem will be easier.

Conclusion

High performance is often misunderstood and a touchy subject. Although this article is short and skips a lot of details, hopefully it provides some useful information. I'm hoping Apress will do something with the book, or let Remy and I shop the book to other publishers. If not, I would like to cover some of the more mundane sections like writing functional and performance requirements in greater detail when I have more time. Remy and I wrote three chapters each, so a lot of details are left out. High performance isn't magic. If you want it bad enough and stay focused, there is a road to travel. It might be full of mud, cliffs and wild animals, but the journey is worthwhile.