

Evidence based performance tuning of enterprise Java applications

By Jeroen Borgers

jborgers@xebia.com



Bad performance can lead to complete failure: the end of AT&T Wireless

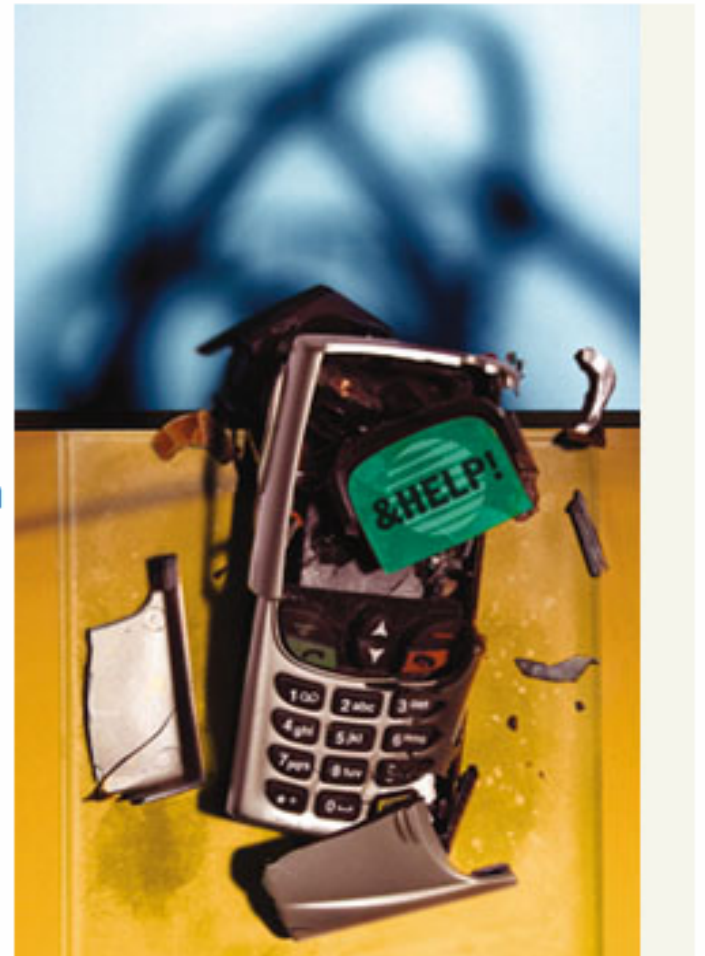
Apr. 15, 2004 Issue of **CIO Magazine**

PROJECT MANAGEMENT | AT&T WIRELESS

AT&T Wireless Self-Destructs

The story of a botched CRM upgrade that cost the telco thousands of new customers and an estimated \$100 million in lost revenue. Hard lessons learned.

BY CHRISTOPHER KOCH



Bad performance: hard to fix

- It took an AT&T call center agent 20 minutes to work through five or six screens with customer
 - Customer un-friendly
 - High cost
- Speedup needed before introduction of number porting
- Major redesign was needed
- Result: permanent system down

Goal

Provide an answer to the question:
How can I speedup my enterprise Java app?

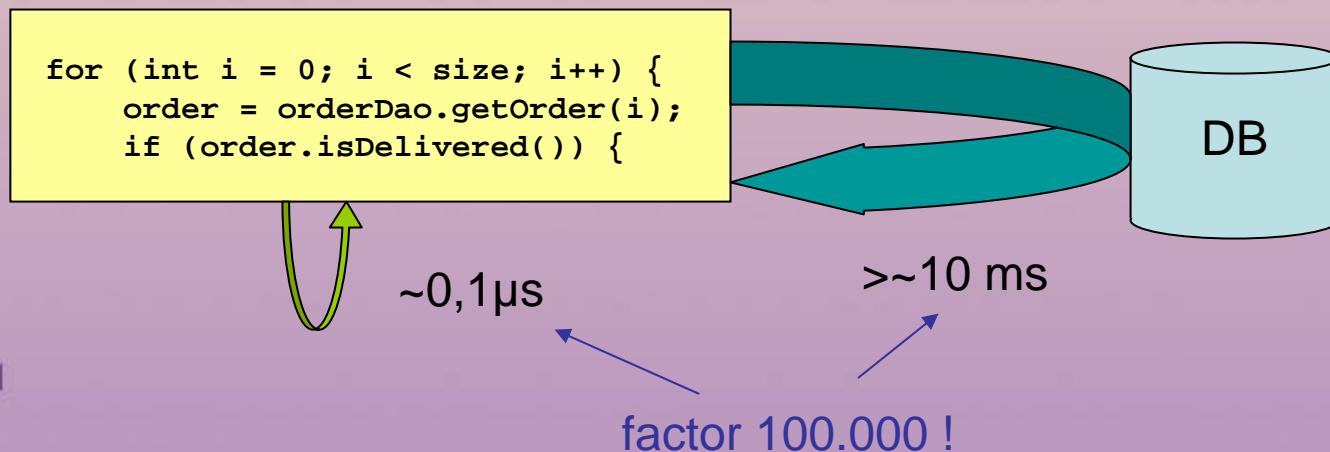


Agenda

- Enterprise Java performance playground
- Architectural vs. implementation level
- How to achieve bad performance
- Evidence based tuning for high speed
- Tools overview
- Largest Dutch web shop case
- Conclusions

Enterprise Java performance playground

- Java is not slow anymore
- Importance how you do it: algorithms
- Typical enterprise app: most time spent in calls out of Java process
 - Remote calls
 - Database access



Architecture enables performance and scalability

- Key choices
 - Application distribution approach
 - Persistent data access approach: O-R mapping
 - Data conversion: XML, encodings
 - Presentation tier technologies
 - Use of proprietary features of db, app server
 - Data caching
- It is all about:
 - remote calls
 - data conversion between models

Architecture enables, implementation makes it happen

- Minimize remote calls and other I/O
- Speed-up data conversion
- Code optimizations

Architecture enables, implementation makes it happen

- Minimize remote calls and other I/O
 - Remote EJB's: use of DTO's
 - Direct JDBC: use of batch updates
 - Hibernate: use of query caching
- Speed-up data conversion
 - XML usage: use JiBX instead of JAXB
 - URL encoding: use commons codec instead of java.net
 - Java serialization: use custom instead of default
- Code optimizations
 - Algorithms, collections
 - Threading and synchronization
 - Initializing only once

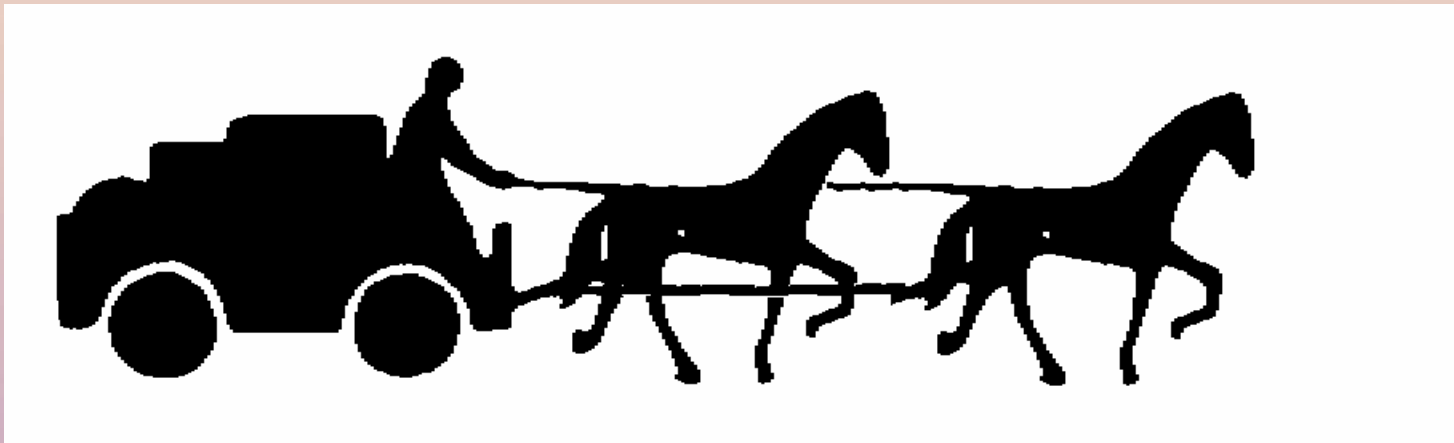
Architecture enables performance & scalability

- The architecture



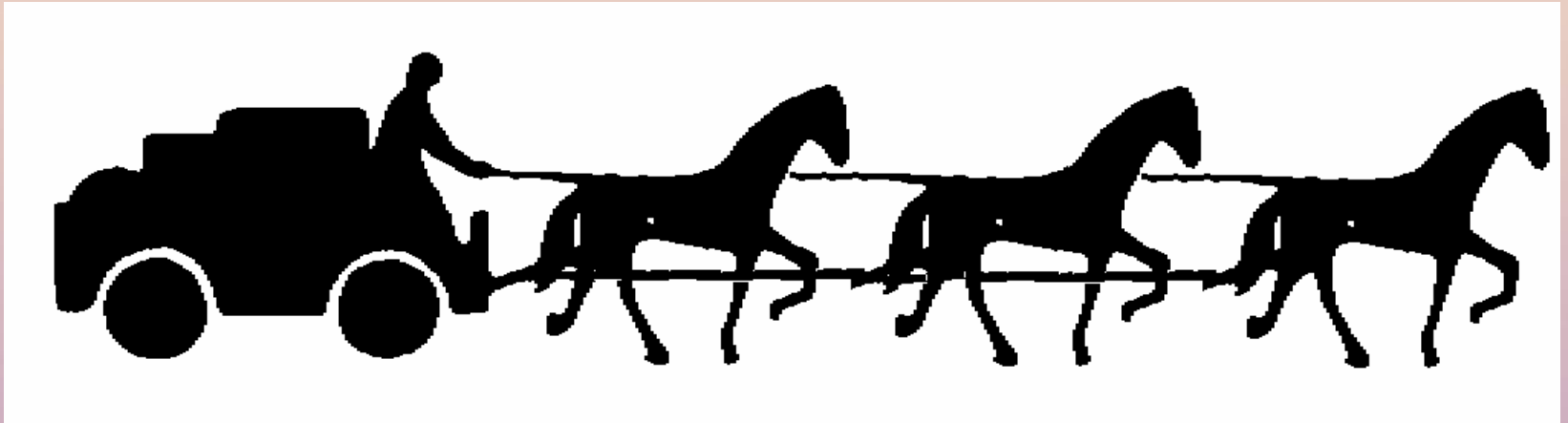
Architecture enables performance & scalability

- The architecture is scalable



Architecture enables performance & scalability

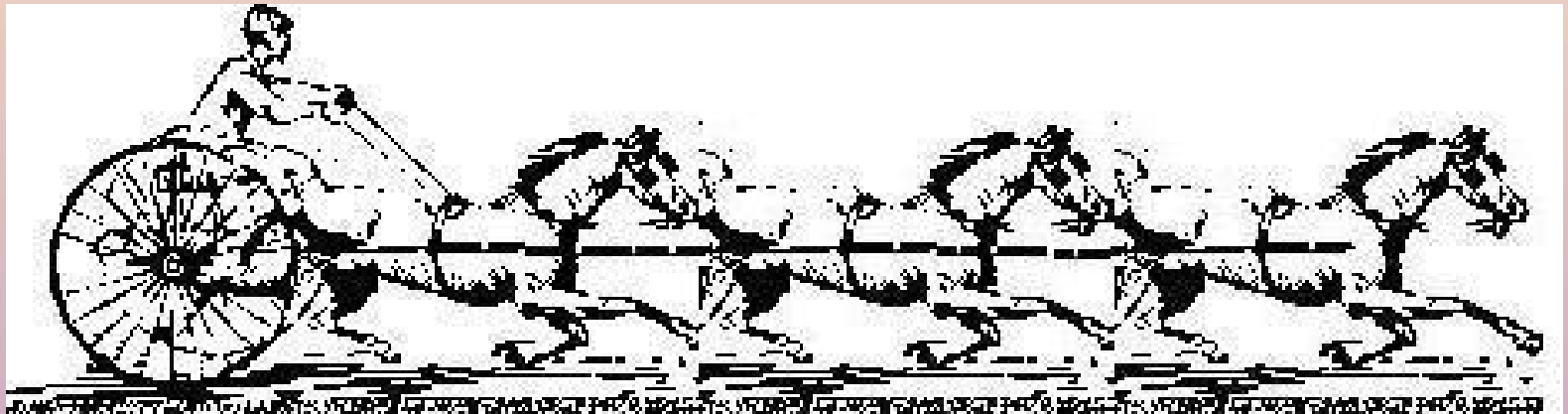
- The architecture is scalable



- How one implements the architecture makes a difference

Architecture enables performance & scalability

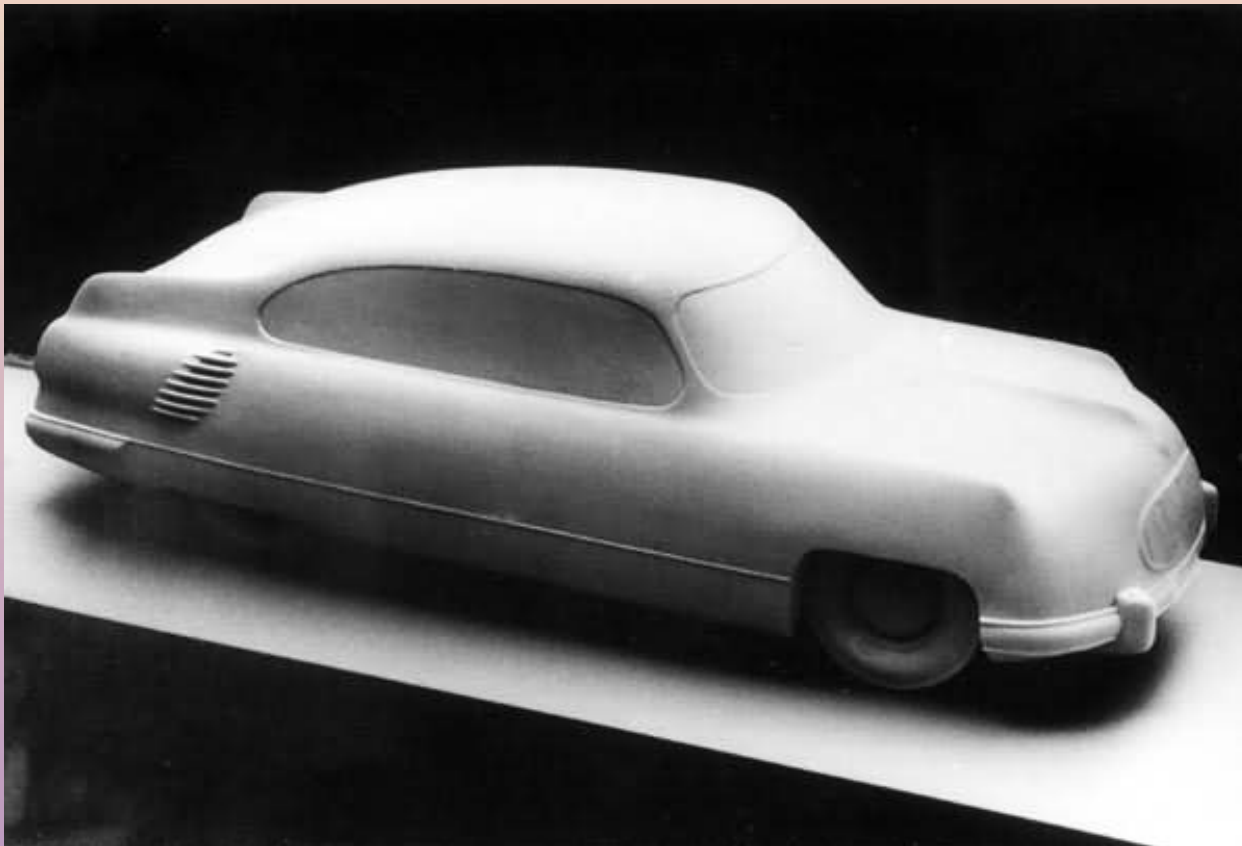
- How one implements the architecture makes a difference



- But the requirement of 120 km/h will never be met!
- Major redesign needed

Architecture enables performance & scalability

- A more suitable architecture for 120 km/h:



Architecture enables performance & scalability

- More demanding requirements can be met with the right implementation



- Bugatti EB Veyron 16.4
- Top 407 km/h
- 0-300 in 14 s.

How to achieve bad performance

- Many J2EE apps have hard to solve performance problems
- How to achieve this?
 - Ignore the problems too long
 - Don't follow best practices
 - Optimize counterproductively



How to achieve bad performance

- Project teams
 - *Ignore the problems too long*
 - Don't follow best practices
 - Optimize counterproductively



Ignore problems until production

- Project leader: “First we get the functionality right, then we will start worrying about performance.”
- Performance is often ignored until the final test, a week before production, or later!
- To meet requirements you likely need a major re-design: costly
- Solved by:
 - Verify the architecture in a POC: test a vertical slice
 - Test performance continuously

Ignore performance problems

- Architect: “Most important is a *pure*, flexible, scalable architecture: we distribute our application components and loosely couple with web services.”
- Nonsense!
 - Those who pay the bill care about efficient working, not purity;
 - Speed is traded for scalability: remote calls and data conversion can be killing;
 - How is flexibility gained?

How to achieve bad performance

- Project teams
 - Ignore the problems too long
 - ***Don't follow best practices***
 - Optimize counterproductively



Don't follow best practices

- Best practices are like:
 - Use Map instead of List for lookups
 - Use I/O buffering: `InputStreamBuffer`
 - Use SQL batch updates
 - Use db table indexes
 - Use `StringBuffer` for String concatenation?
- Be aware: a best practice may not work in your situation

Best practices can be premature optimizations with trade-offs

- Developer: “String performs badly, we use StringBuffer for concatenation everywhere.”
- Many books: “*use StringBuffer for*
`toString() { return "[" + getName() + "]" ; }`
- *faster is*
`toString() {
 StringBuffer buf = new StringBuffer("[");
 buf.append(getName()).append("]");
 return buf.toString();
}`”
- However: same bytecode!
- Maintainability and developer time is traded for a non-existing performance gain
- “*Premature optimization is the root of all evil*” - Hoare and Knuth

How to achieve bad performance

- Project teams
 - Ignore the problems too long
 - Don't follow best practices
 - **Optimize counterproductively**



Striking counterproductive optimizations

- Let J2EE container deal with O-R mapping with entity bean finders
 - 1+N query problem
- Updating state remotely by sending only delta's like RemoveAttribute, AddAttribute
 - many small objects with much serialization and communication overhead
- Incremental garbage collection with -Xincgc
- System.gc() in HttpSession.invalidate
- Clustering a content management web app

More counterproductive optimizations

- Object pooling depending on JVM
- Thread pooling depending on JVM
- Optimizations are very time and environment dependent
- So: don't trust your knowledge or intuition, but measure that your optimization solves the problem!

“Facts, Hercule, facts! Nothing matters but the facts. Without them the science of criminal investigation is nothing more than a guessing game.” – Clouseau - A Shot in the Dark



Evidence based tuning for high speed

- Don't make assumptions but measure
- Have quantified requirements, prove they are met or not
- Prove the architecture performs in a POC
- Continuously test performance during development
 - to get a first impression
 - for quick feedback on changes
- Test representatively in a dedicated environment
 - with production-like load
 - on a production-like environment
 - with production-like data
- Fix problems when found
- Prove the effect of each optimization
- Monitor the performance of your app in production

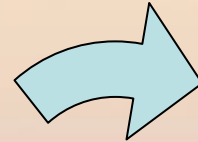
Evidence based tuning process

Measure

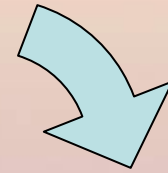
somewhere in app life cycle



Identify



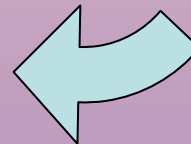
Isolate



Diagnose



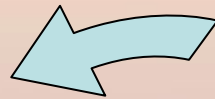
Choose



Improve



Verify



Requirement met



Tools for measuring

- **OS profilers** to see resource usage: CPU, memory, I/O, threading
- **Database analyzers** to see query behavior in the database
- **Network analyzers** to see network traffic
- **App server monitors** to see app server resource usage: heap, connection pool, stmt cache,...
- **J2EE analyzers** to see J2EE behavior under load
- **Instrumentation monitors** to see timing at the instrumented locations
- **Java profilers** to see detailed CPU and memory behavior in JVM
- **Continuous test tools** to quickly get timing feedback of code changes
- **Injector tools** to generate usage load

Case: speeding up a web shop

Wehkamp, 's lands meest actuele thuiswinkel online - Opera

File Edit View Bookmarks Tools Window Help

New page W Wehkamp, 's lands ...

W http://www.wehkamp.nl/

Google search

wehkamp.nl + Home + Winkelmand trefwoord, bestelnr. zoek >

Mode voor haar + Kindermode + Fit, mooi & gezond + Beeld & geluid + Huishouden + Games & consoles + DVD + Opruiming

Mode voor hem + Wonen & slapen + Hobby, spel & vrije tijd + PC & toebehoren + GSM & telefoon + Boeken + Geldzaken + Fotoservice

Gratis strandtas

Gratis fietscomputer

15% korting op deze mode

De allerbeste Oranje outfits

Laat de leeuw niet in zijn hempje staan!
Scoor nu alvast de allerbeste Oranje outfits

klik hier!

ALLES ORANJE
De voorpret is begonnen

NAAR DE ORANJESHOP [KLIK HIER](#)

Mijn wehkamp.nl
Stel uw eigen webwinkel samen. Profiteer van persoonlijk voordeel. [Meld u hier aan](#)

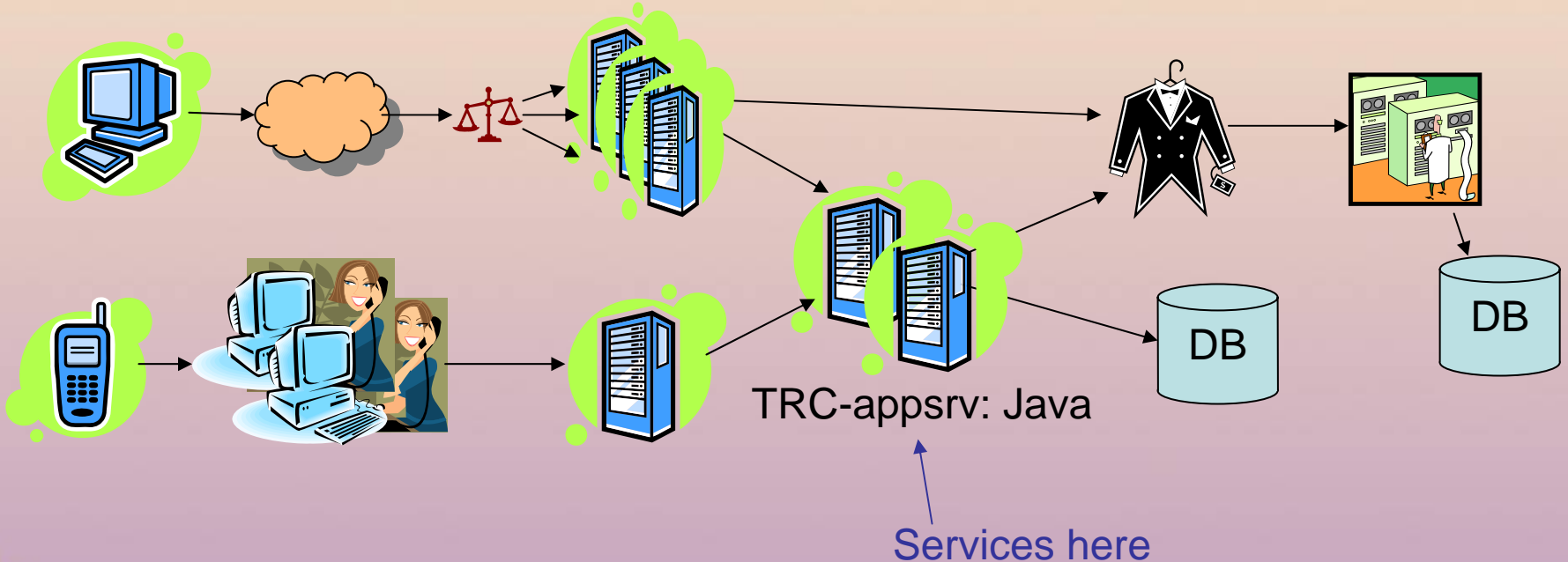
U kunt hier inloggen
E-mailadres:
Wachtwoord: [LOGIN](#)
Wachtwoord vergeten? [Klik hier](#)

IT Architects

30

Wehkamp TRC architecture

web server: ASP



Architecture enabled performance

- Architecture enabled meeting performance requirements
 - Front load balancing over 32 web servers and 4 WebSphere nodes
 - No real remote calls within Java application: no distribution of EJB's
 - Recent introduction of Spring enabled real local calls between POJOs
- Not strong: separation in technology between web tier and app tier
 - Duplication of calls: e.g. inventory calls in shopping cart and order
 - Remote call needed with data conversion
 - No performance analyses tooling support for the whole
- Weak point: mainframe still in use
 - Remote call needed
 - Limited performance analyses tooling support, hard to optimize



Speedup results at Wehkamp

- Several services optimized to meet requirements
- Most expensive: voegKlantToe (addCustomer);
verwerkenNieuweOrder (processNewOrder): from 7 s. to 1.5 s.
- 45% db-queries, 45% tux-calls, 10% java code
- Top optimizations:
 - Reduce number of db queries
 - Optimize most expensive db queries
 - Reduce number of Tux/mainframe calls
 - Optimize most expensive Tux/mainframe calls
- App orchestrates: many code changes needed for this

Tools used to achieve the speedup

- Unix OS profiler: vmstat, nmon
- Database analyzer: Quest Toad, Oracle Statspack
- Network analyzer: Ethereal
- App server monitor: Tivoli Performance Monitor
- J2EE analyzer: Quest PerformAsure
- Instrumentation monitor: JAMon API
- Java profiler: Quest JProbe
- Continuous test tool: jUnitPerf
- Injector tool: Apache JMeter

JMeter for generating load and black box testing

The screenshot shows the Apache JMeter interface. The left sidebar displays a test plan tree for 'TRC performance tests'. The main window is titled 'View Results Tree' and shows the following details:

Name: View Results Tree

Write All Data to a File

Filename: Log Errors Only

View:

verwerkenNieuweOrder:

securityCredential:

klantnummer:

affiliateId:

gekozenKialaId:

WINKELMANDJE:

	bron	artikelnr	bmaata	aantal	foto	eos	VerzW	bezdat	def-bdat	ritnr	dist	vnr	b-M	rek-M *
orderregel 1:	86	119470	456	1		N	SV				2	1		
orderregel 2:	86	119470	456	1		N	SV		N		2	2		
orderregel 3:	86	122887	038	1		N	EV				2	3		
orderregel 4:												4		
orderregel 5:												5		
orderregel 6:												6		

Show Text Render HTML Render XML Download embedded resources

JMeter mail to stakeholders

perftests - Microsoft Outlook

Bestand Bewerken Beeld Ga naar Extra Acties Help

perftests

automatische-jme... Resultaten van performance tests wehkamp_test_once,wehkamp_test_warmup,wehkamp_test_performance... do 9-3-2006 21:21

Van: automatische-jmeter-tests@wehkamp.nl **Aan:** ...

Onderwerp: Resultaten van performance tests wehkamp_test_once,wehkamp_test_warmup,wehkamp_test_perform... **CC:** ...

[Up](#)

Test runs executed at 09/03/2006 21:00:01

This report shows the summaries of all test runs executed at 09/03/2006 21:00:01. Detailed information can be found by following the links.

Legend

- Duration failure
- Semantic failure
- Error

Summaries

Test run	Tests	Results	Average Time	Min Time	Max Time
wehkamp_test_once	11	8 (green) 3 (yellow)	524 ms	78 ms	1203 ms
wehkamp_test_warmup	37	33 (green) 4 (yellow)	708 ms	31 ms	5125 ms
wehkamp_test_performance	392	355 (green) 29 (yellow) 8 (red)	566 ms	15 ms	15687 ms
wehkamp_test_load2	374	349 (green) 19 (yellow) 5 (red) 1 (red)	564 ms	15 ms	14015 ms
wehkamp_test_load4	1343	1237 (green) 73 (yellow) 33 (red)	512 ms	0 ms	13031 ms
wehkamp_test_load8	342	304 (green) 27 (yellow) 7 (red) 4 (red)	553 ms	15 ms	11625 ms
wehkamp_test_load16	2537	2137 (green) 324 (yellow) 46 (red) 30 (red)	750 ms	15 ms	13141 ms

1379 items

JAMon for monitoring statistics

Browser address bar: http://.../JAMonView.jsp?sortCol=4&sortOrder=desc&filter=

Page tabs: JAMon View Page, JAMon Admin Page

Refresh Filter:

added ←

JAMon logo

8-5-06 18:29:45

service name ←

Monitor Label	Hits	Avg ms.	Total ms.	Std Dev ms.	Min ms.	Max ms.	Active	Avg Active	Max Active	First access	Last access
nl.wehkamp.trc.channelservices.internet.service.order.WebOrderService.verwerkenNieuweOrder	4.571	1.508	6.896.962	1.620	0	22.064	0	1,2	4	8-5-06 8:41:45	8-5-06 18:29:45
nl.wehkamp.trc.channelservices.internet.service.klant.WebKlantService.getFinancieleGegevens	14.342	449	6.445.800	671	79	13.916	0	1,2	6	8-5-06 8:41:23	8-5-06 18:29:45
nl.wehkamp.trc.channelservices.internet.service.order.WebOrderService.bepalenOrderKorting	24.796	214	5.310.901	193	4	7.469	0	1,2	6	8-5-06 8:41:28	8-5-06 18:29:45
nl.wehkamp.trc.dao.oracle.OracleOrderDAO-usingDBConnection	35.753	142	5.084.669	635	0	21.417	0	1,1	4	8-5-06 8:41:20	8-5-06 18:29:45
nl.wehkamp.trc.proxy.tuxedo.TuxedoTijdVoorraadProxy-TRC_TV_REQUEST	33.565	127	4.274.147	123	24	1.434	0	1,1	4	8-5-06 8:41:28	8-5-06 18:29:45
nl.wehkamp.trc.dao.oracle.OracleOrderRegelDAO-usingDBConnection	64.620	63	4.096.141	507	1	113.416	0	1,1	5	8-5-06 8:41:20	8-5-06 18:29:45
nl.wehkamp.trc.dao.oracle.OracleMoneyAtHomeDAO-usingDBConnection	72.217	51	3.712.885	33	0	7.990	0	1,1	5	8-5-06 8:41:23	8-5-06 18:29:45
nl.wehkamp.trc.dao.oracle.OracleFinancieleMutatieDAO-usingDBConnection	69.380	44	3.106.858	178	0	3.305	0	1,1	4	8-5-06 8:41:23	8-5-06 18:29:45
nl.wehkamp.trc.channelservices.callcenter.rekening.CcRekeningService.raadplegenFinancieleStatus	3.267	932	3.047.569	818	65	10.829	0	1,1	3	8-5-06 8:41:26	8-5-06 18:29:45
nl.wehkamp.trc.dao.oracle.OracleContactEventDAO-usingDBConnection	45.675	53	2.465.060	231	0	17.082	0	1,1	4	8-5-06 8:41:22	8-5-06 18:29:45
nl.wehkamp.trc.channelservices.callcenter.contact.CcContactService.findContactEvents	4.030	507	2.043.720	752	1	17.083	0	1,1	3	8-5-06 8:41:30	8-5-06 18:29:45
nl.wehkamp.trc.channelservices.internet.service.rekening.WebRekeningService.conroleerBetaalvoorkeurNaarTerminj	4.534	437	1.985.704	587	0	8.114	0	1,1	3	8-5-06 8:41:30	8-5-06 18:29:45
nl.wehkamp.trc.dao.oracle.OracleRekeningOverzichtDAO-usingDBConnection	52.196	37	1.971.812	108	0	1.754	0	1,1	4	8-5-06 8:41:22	8-5-06 18:29:45
nl.wehkamp.trc.dao.oracle.OracleKlantVoorkeurDAO-usingDBConnection	100.546	18	1.868.090	22	0	2.933	0	1,1	4	8-5-06 8:41:16	8-5-06 18:29:45
nl.wehkamp.trc.channelservices.internet.service.dienst.WebDienstService.moneyAtHomeToegestaan	8.255	221	1.827.438	224	84	5.048	0	1,1	4	8-5-06 8:41:29	8-5-06 18:29:45
nl.wehkamp.trc.dao.oracle.OracleKlantDAO-usingDBConnection	188.967	7	1.441.974	45	0	9.964	0	1	4	8-5-06 8:41:16	8-5-06 18:29:45
nl.wehkamp.trc.channelservices.callcenter.rekening.CcRekeningService.raadplegenFinancieelOverzicht	759	1.888	1.433.085	1.316	25	6.191	0	1	3	8-5-06 8:41:33	8-5-06 18:28:00
nl.wehkamp.trc.proxy.tuxedo.TuxedoFinancieelOverzichtProxy-TRC_VWHIST	759	1.879	1.426.853	1.311	25	6.181	0	1	3	8-5-06 8:41:33	8-5-06 18:28:00
nl.wehkamp.trc.dao.oracle.OracleArtikelDAO-usingDBConnection	456.505	2	1.093.524	5	0	1.174	0	1	4	8-5-06 8:41:28	8-5-06 18:29:45
nl.wehkamp.trc.channelservices.internet.service.order.WebOrderService.ophalenNogTeLeverenOrderregels	4.601	230	1.059.518	380	27	6.081	0	1	3	8-5-06 8:41:25	8-5-06 18:29:45
nl.wehkamp.trc.channelservices.callcenter.order.CcOrderService.zoekOrderOrderRegel	2.068	507	1.050.060	1.005	1	15.289	0	1	3	8-5-06 8:41:20	8-5-06 18:29:45
nl.wehkamp.trc.dao.oracle.OracleContactDAO-usingDBConnection	26.331	38	1.001.928	111	0	9.418	0	1	4	8-5-06 8:41:22	8-5-06 18:29:45
nl.wehkamp.trc.channelservices.internet.service.klant.WebKlantService.getKlantwoonadres	14.867	65	975.655	114	3	1.790	0	1	3	8-5-06 8:41:23	8-5-06 18:29:45
nl.wehkamp.trc.channelservices.vrs.service.order.VrsOrderService.verwerkenNieuweOrder	496	1.850	917.934	1.487	187	11.277	0	1	3	8-5-06 8:50:00	8-5-06 18:29:30
nl.wehkamp.trc.channelservices.callcenter.order.CcOrderService.beoordelenOrder	1.015	856	869.817	892	76	14.557	0	1	2	8-5-06 8:41:44	8-5-06 18:29:45
nl.wehkamp.trc.channelservices.callcenter.klant.CcKlantService.searchKlantByKlantnummer	4.542	160	728.670	135	1	3.799	0	1	3	8-5-06 8:41:19	8-5-06 18:29:45
nl.wehkamp.trc.channelservices.internet.service.order.WebOrderService.bepalenBezorgkosten	41.758	16	684.014	70	1	6.939	0	1	8	8-5-06 8:41:16	8-5-06 18:29:45
nl.wehkamp.trc.channelservices.internet.service.klant.WebKlantService.getKlantbasisgegevens	4.230	150	635.052	202	7	1.828	0	1	3	8-5-06 8:41:22	8-5-06 18:29:45
nl.wehkamp.trc.dao.ldap.LdapAgentDAO-usingLDAPConnection	897	693	621.668	284	125	2.095	0	1	2	8-5-06 8:41:32	8-5-06 18:29:30
nl.wehkamp.trc.channelservices.callcenter.order.CcOrderService.samenstellenOrderRegel	2.930	211	620.416	147	4	2.020	0	1	2	8-5-06 8:42:15	8-5-06 18:29:45
nl.wehkamp.trc.dao.oracle.logging.BusinessRuleLoggingBatch-usingDBConnection	5.769	103	598.780	80	3	2.414	0	1	1	8-5-06 8:41:24	8-5-06 18:29:45
nl.wehkamp.trc.dao.oracle.OracleWbgDAO-usingDBConnection	29.470	19	580.919	89	0	5.123	0	1	3	8-5-06 8:41:16	8-5-06 18:29:45

Instrumenting with JAMon API

```
Java - GoodJamonPerformanceMonitorInterceptor.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
WebOrderServiceImpl.java WebAfspraakServiceNoContainerT... TijdVoorraadProxy.java *GoodJamonPerformanceMonitorInt...
<p>
This interceptor below intends to fix that.
@author Erwin Bolwidt
public class GoodJamonPerformanceMonitorInterceptor implements MethodInterceptor, Serializable {
    public Object invoke(MethodInvocation invocation) throws Throwable {
        String name = invocation.getMethod().getDeclaringClass().getName() + "."
            + invocation.getMethod().getName();
        Monitor monitor = MonitorFactory.start(name);
        try {
            return invocation.proceed();
        } finally {
            monitor.stop();
        }
    }
}
```

JAMon API call

JAMon report for history

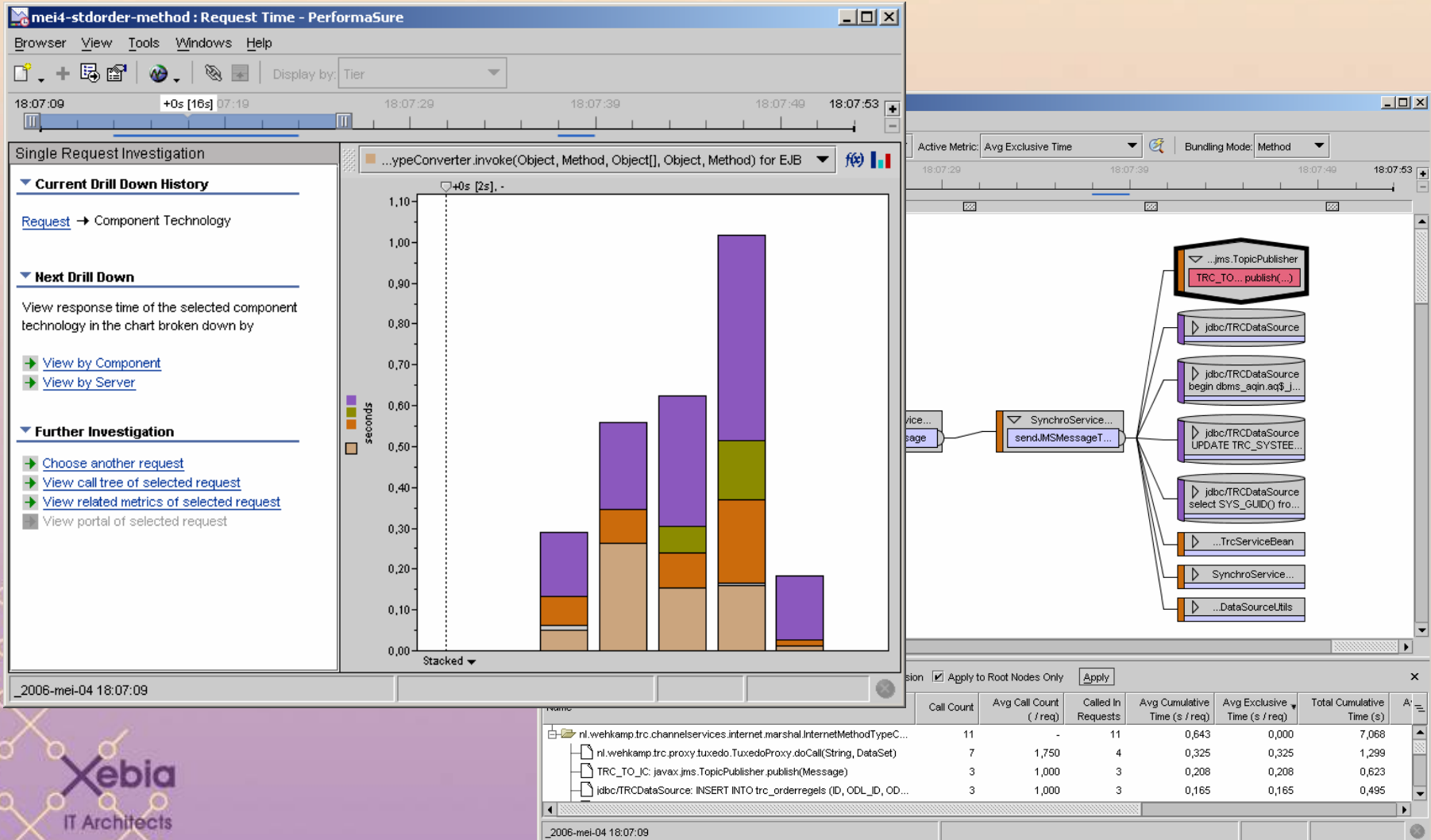


Requirement:
1.5 s.

Sensor for monitoring statistics and viewing call trees

Id	Hits	Avg ms.	Std Dev ms.	Total ms.	Min ms.	Max ms.	First Access
/sensor-sample-app/overview.htm	1	57	0	57	57	57	2006-06-11 16:12:59:83
net.sf.sensor.sample.springroller.dao.impl.memory.MemoryArticleDao.getAllArticles()	1	2	0	2	2	2	2006-06-11 16:12:59:83
/sensor-sample-app/timing/statistics.htm	2	74	60	148	31	117	2006-06-11 16:13:06:34
test timer 1	3	0	0	0	0	0	2006-06-11 16:13:06:27
test timer 2	3	0	0	0	0	0	2006-06-11 16:13:06:27
test timer 3	3	0	0	0	0	0	2006-06-11 16:13:06:27
test timer 4	3	0	0	0	0	0	2006-06-11 16:13:06:27
test timer 5	3	0	0	0	0	0	2006-06-11 16:13:06:27
test timer 6	3	0	0	0	0	0	2006-06-11 16:13:06:27
test timer 2	3	0	0	0	0	0	2006-06-11 16:13:06:27
/sensor-sample-app/css/TimerStatistics.css	2	0	1	1	0	1	2006-06-11 16:13:06:40
/sensor-sample-app/add.htm	1	8	0	8	8	8	2006-06-11 16:16:30:40
/sensor-sample-app/show.htm	1	40	0	40	40	40	2006-06-11 16:16:37:73
/sensor-sample-app/addDummy.htm	1	32	0	32	32	32	2006-06-11 16:16:46:19

PerformAsure for analyses under load



Tools complement each other

- JProbe
 - Pro: High detail: code line level
 - Con: High overhead
- JAMon API
 - Pro: Low overhead < 1%
 - Con: No call tree; built-in measure points
- Sensor
 - Pro: Low overhead, call tree available
 - Con: not production ready yet; built-in measure points
- PerformAsure
 - Pro: reasonable overhead, slick UI, measures all on JVM
 - Con: not cheap, configuration, wait time

Conclusions & recommendations

- Bad performance can lead to complete failure
- Bad performance is easily achieved
- Architecture mainly determines performance
- Consider whole application life cycle: requirements, POC, continuous testing, representative testing, monitoring
- Tuning based on evidence is key: measure and prove!
- Various tools support evidence based performance tuning

“Meten is weten”
(measuring means knowing)



References

- <http://www.javaperformancetuning.com>
- <http://www.jamonapi.com>
- <http://sensor.sourceforge.net/>
- <http://jakarta.apache.org/jmeter/>
- http://www.quest.com/performance_management
- Want to know more?
 - Follow the Java Performance Tuning course “Speeding up Java applications” by Kirk Pepperdine, 19-22 September.

Q & A

