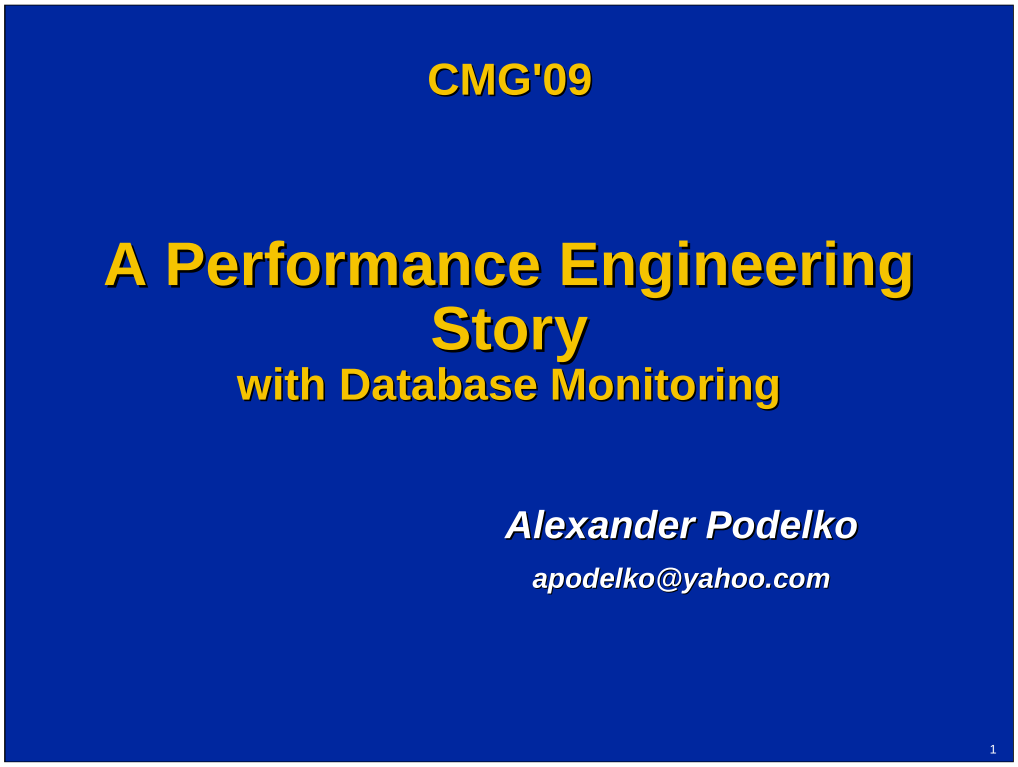

CMG'09

A Performance Engineering Story
with Database Monitoring

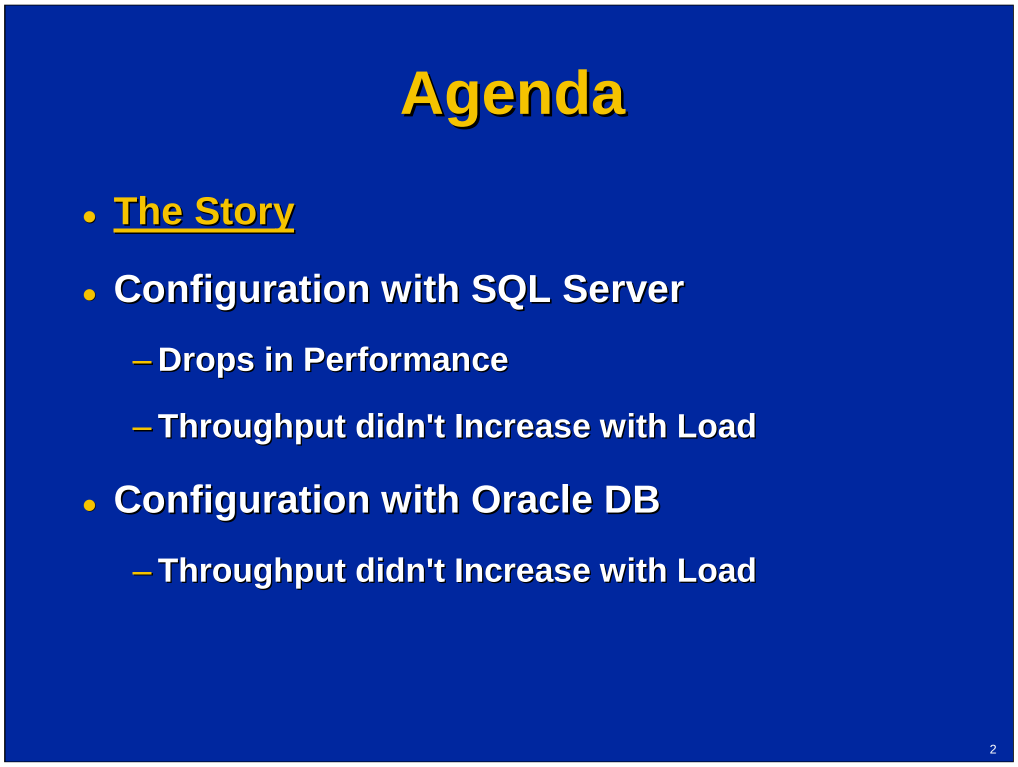
Alexander Podelko

apodelko@yahoo.com

1

Abstract:

This presentation describes a performance engineering project in chronological order. The product under investigation was a three-tier Java application which suggests the best offer to a customer according to provided criteria. The performance issues found turned out to be database-related. Two configuration options were investigated for the repository: Microsoft SQL Server and Oracle Database. PerfMon was used for initial monitoring. While performance issues looked similar for both databases, the root causes were different. Fixing the issues allowed performance to increase by about three times for both configurations.

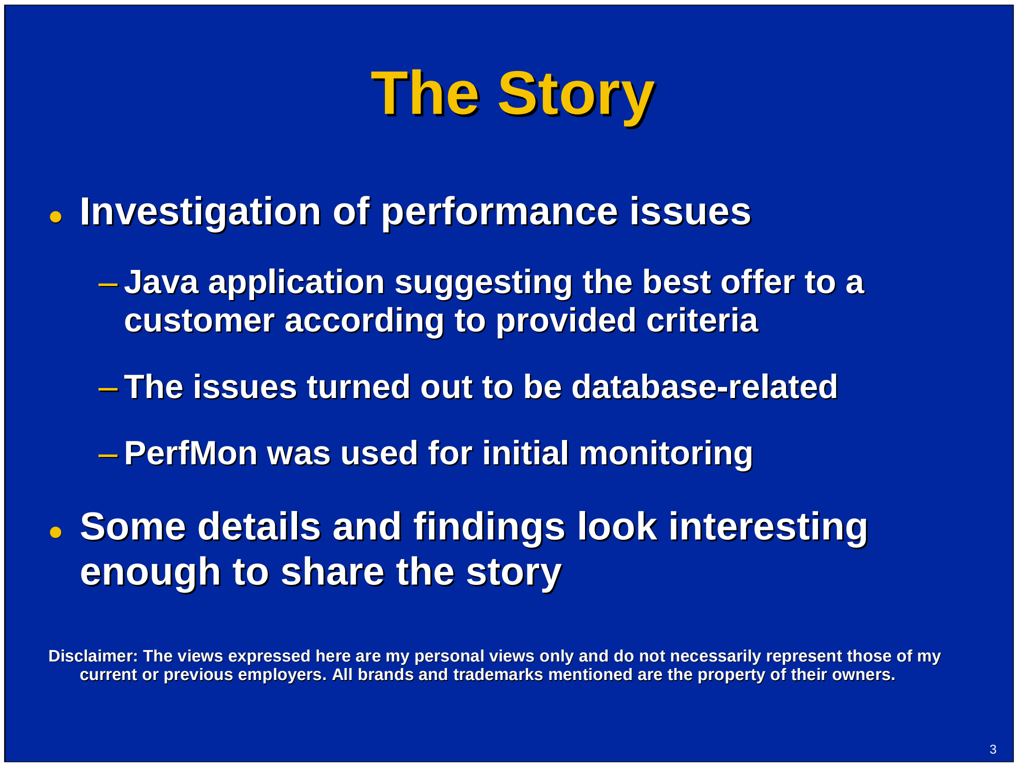


Agenda

- The Story
- Configuration with SQL Server
  - Drops in Performance
  - Throughput didn't Increase with Load
- Configuration with Oracle DB
  - Throughput didn't Increase with Load

2

We start by describing the project. Then we discuss performance analysis and troubleshooting for two different system configurations. One configuration used Microsoft SQL Server for the repository and another used Oracle DB.



The Story

- Investigation of performance issues
  - Java application suggesting the best offer to a customer according to provided criteria
  - The issues turned out to be database-related
  - PerfMon was used for initial monitoring
- Some details and findings look interesting enough to share the story

Disclaimer: The views expressed here are my personal views only and do not necessarily represent those of my current or previous employers. All brands and trademarks mentioned are the property of their owners.

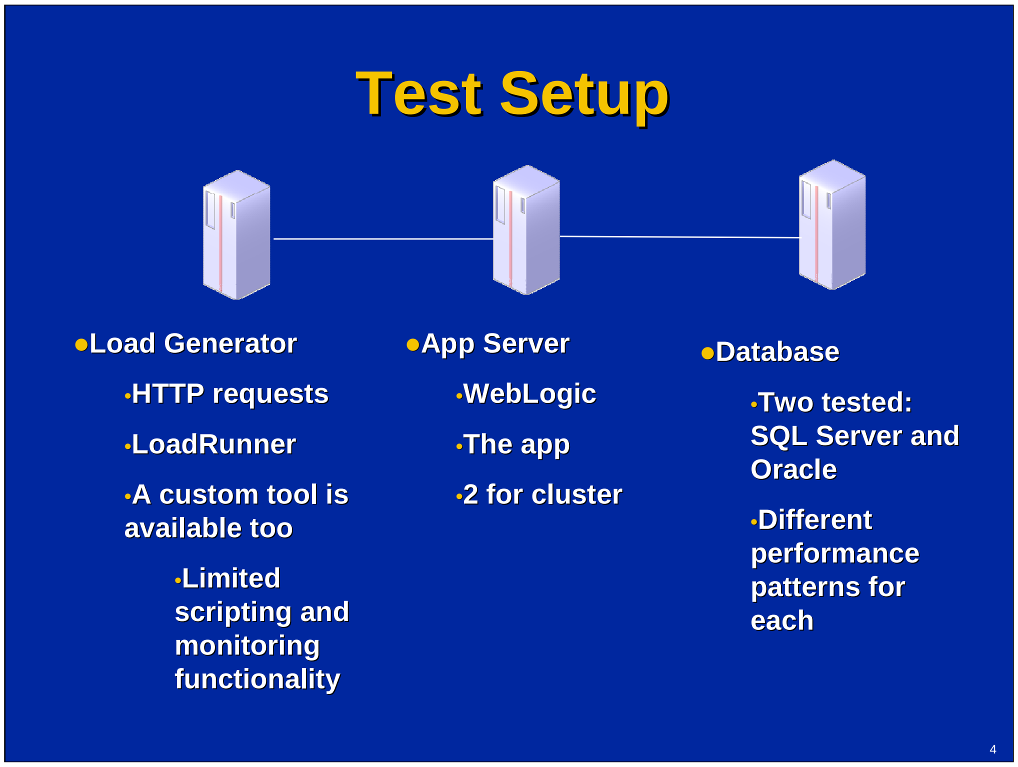
3

This presentation described one performance engineering (PE) project in chronological order. The product under investigation is a Java application suggesting the best offer to a customer according to provided criteria. The name and specific functionality of the product isn't really important to the story. The purpose of the story is to share specific PE experience and discuss some issues related to PE activities and database monitoring. Whatever information about the product is mentioned, is mentioned only to set up a meaningful context for describing the PE engagement.

The performance issues found turned out to be database-related. PerfMon was used for initial monitoring.

Some details and findings look interesting enough to share the story. While some things might be trivial, I still hope that it would contain at least a few interesting details for everybody who is interested in PE.

I want to state explicitly that the views expressed here are my personal views only and do not necessarily represent those of my current or previous employers. All brands and trademarks mentioned are the property of their owners.



Test Setup

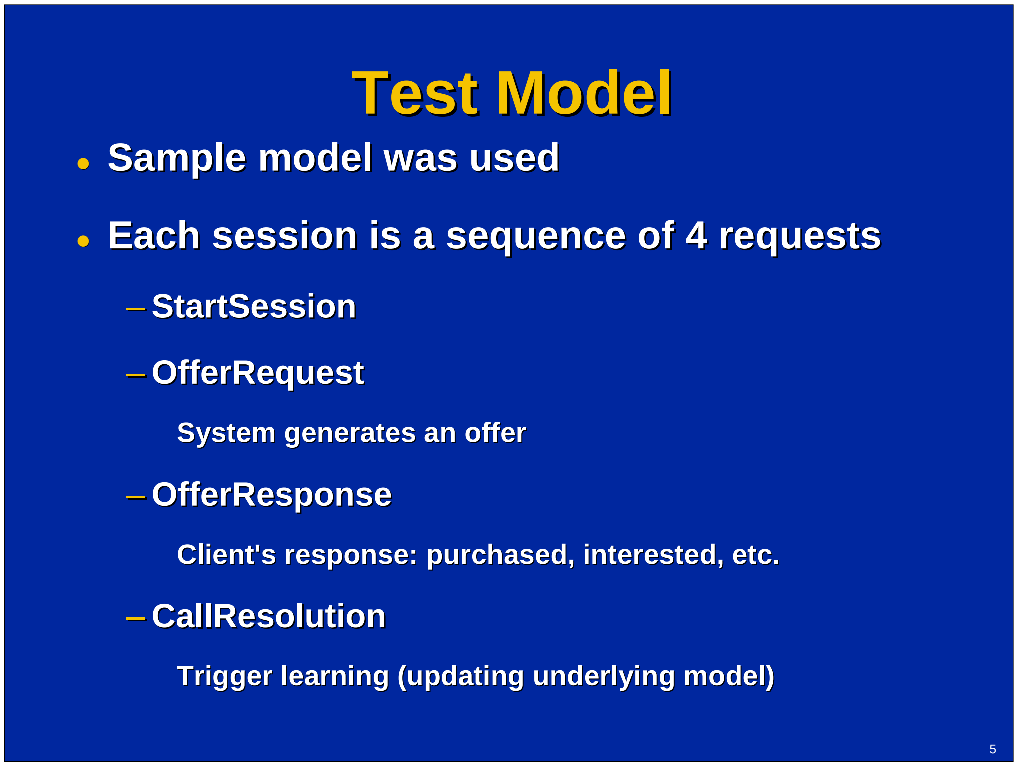
- Load Generator
  - HTTP requests
  - LoadRunner
  - A custom tool is available too
    - Limited scripting and monitoring functionality
- App Server
  - WebLogic
  - The app
  - 2 for cluster
- Database
  - Two tested: SQL Server and Oracle
  - Different performance patterns for each

The system under investigation is a three-tier Java EE (Enterprise Edition) application.

The first tier is a client, replaced by a load generation tool in this project. While there is a simple load generation tool shipped with the system, it has very limited scripting and monitoring functionality and was used only for validation. HP LoadRunner was used as the main way to generate load.

The second tier is the Java EE application itself deployed on WebLogic Application Server. The application can work with other application servers, but they were not tested during the described project. Most tests were done with one application server, but when cluster is explicitly mentioned, two identical servers were used as application servers and load was balanced between them.

The third tier is the database tier. Two options were investigated during the described project: Microsoft SQL Server and Oracle Database. Actually the described project consists of two major parts: one investigated performance issues with Microsoft SQL Server and another investigated performance issues with Oracle Database. Workload and the application were the same, even performance was close – but issues uncovered were different.


Test Model

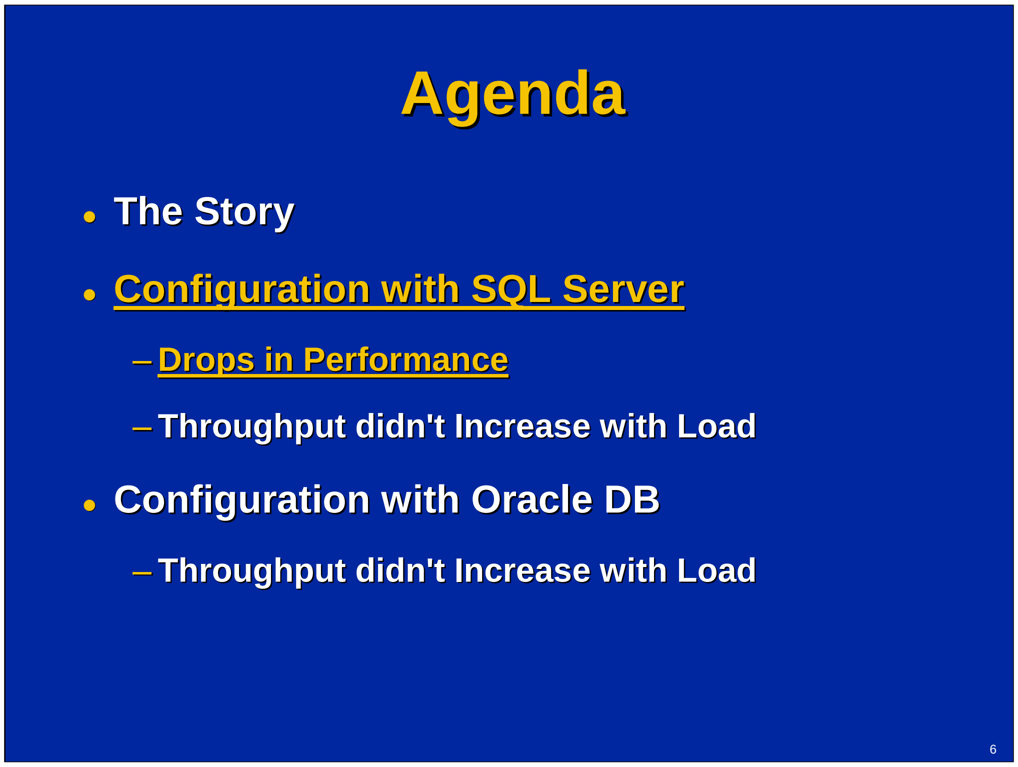
- Sample model was used
- Each session is a sequence of 4 requests
  - StartSession
  - OfferRequest
    System generates an offer
  - OfferResponse
    Client's response: purchased, interested, etc.
  - CallResolution
    Trigger learning (updating underlying model)

5

A sample model was used during the project and the workload was pretty simple: each session consists of four sequential HTTP requests. The session could be a customer representative call or a web session. As soon as the session starts (a client identifies himself to the customer representative or logs into the web site), the OfferRequest request is sent to our back-end decision-making system. The system finds the best offer based on the client information according to specific criteria. The offer gets delivered to the customer and his response (OfferResponse) is sent back to the system. When the session is ended, the CallResolution request is sent to the system, which saves all necessary information about the session.

A separate asynchronous process is running in the background to summarize all this information about the finished sessions and update the rules used to make decisions ("learning" – a kind of artificial intelligence).

For example, when you call to your bank to solve an issue, the system might suggest that the best offer for you would be Wolfram Credit Card or Super Credit Protection. You response gets recorded and eventually the decision-making model gets updated for your demographic data – so when you call next time you get an offer that would be irresistible for your demographic.



Agenda

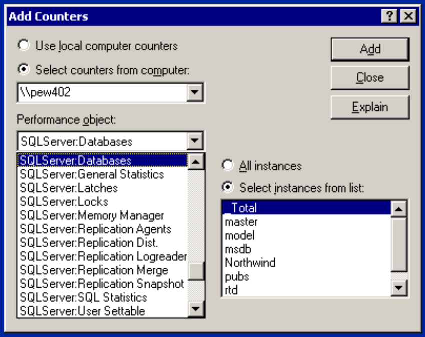
- The Story
- Configuration with SQL Server
  - Drops in Performance
  - Throughput didn't Increase with Load
- Configuration with Oracle DB
  - Throughput didn't Increase with Load

6

The first part of the project was performance analysis of the configuration using Microsoft SQL Server for the repository.

SQL Server

- PerfMon is the native way of monitoring
- SQL Server exposes a lot of counters


Add Counters

( ) Use local computer counters
(●) Select counters from computer:
\\pew402

Performance object:
SQLServer:Databases

SQLServer:Databases
SQLServer:General Statistics
SQLServer:Latches
SQLServer:Locks
SQLServer:Memory Manager
SQLServer:Replication Agents
SQLServer:Replication Dist.
SQLServer:Replication Logreader
SQLServer:Replication Merge
SQLServer:Replication Snapshot
SQLServer:SQL Statistics
SQLServer:User Settable

( ) All instances
(●) Select instances from list:

_Total
master
model
msdb
Northwind
pubs
rtd

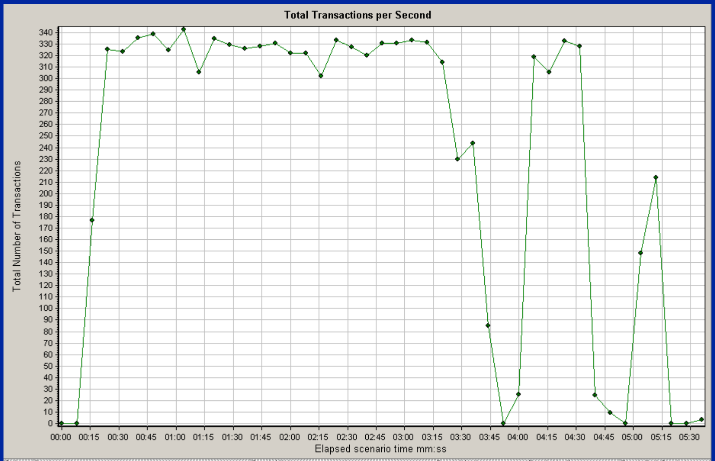
PerfMon is the native way of monitoring for Microsoft SQL Server resource usage. I am not a DBA and have limited knowledge of database-specific tools. I have found use of PerfMon for monitoring databases very helpful in my work as a performance engineer. The name PerfMon (Performance Monitor) is used during this presentation following the tradition among performance specialists, although Microsoft doesn't use it after Windows NT 4.0. Now it is referred to instead as System Monitor in Performance Console.

Using PerfMon for monitoring databases has several advantages for performance engineers / testers:
-Collecting all performance information in one place
-Getting some DB-related metrics on early stages without DBAs or DBA-level tools
-If issues are observed, there is already some information available pointing to the area for further investigation

Microsoft SQL Server exposes a lot of useful counters. There many good sources describing the topic in details, for example:
Monitoring Resource Usage (System Monitor). SQL Server 2008 Books Online. http://msdn.microsoft.com/en-us/library/ms191246.aspx
Understanding SQL Performance Counters. http://www.extremeexperts.com/sql/articles/sqlcounters.aspx

SQL Server Issue 1

- Transaction throughput drops to zero



Color	Scale	Measurement	Graph Minimum	Average	Graph Maximum	Graph Median	Graph Std. Deviation
🟩	1	Pass	0	232.854	342.625	320	134.902

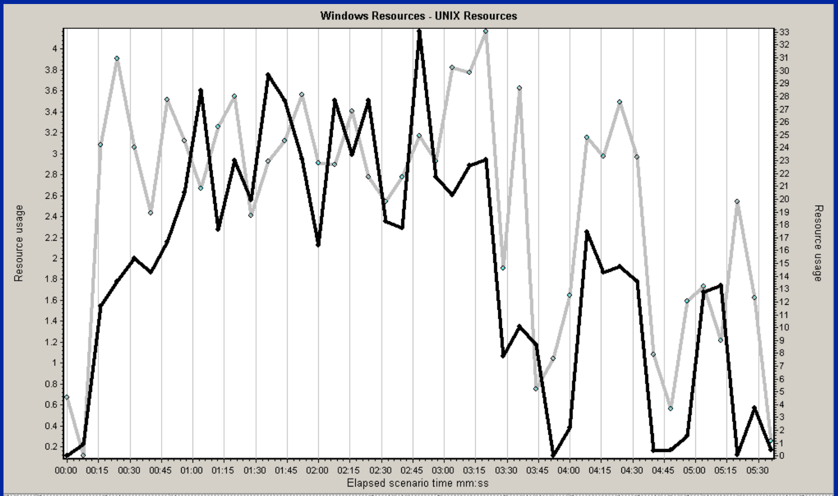
8

The initial set of tests demonstrated pretty good performance (it stayed above 300 requests/sec *when* performance was stable), but there were drops to zero throughput at random moments in time.

The initial hypothesis was that it could be the Java garbage collection, but this turned out to be incorrect. Java garbage collection is a common scapegoat for such drops in performance, but in reality it is pretty good in the latest versions of Java and rarely is a problem.

One comment for those who know PerfMon graphs well: the graphs here are not actual PerfMon graphs, but LoadRunner graphs presenting the same information. LoadRunner Resource Monitor was used to collect PerfMon information. The information is the same, but the appearance is a little different.

A Lot of Available Resources

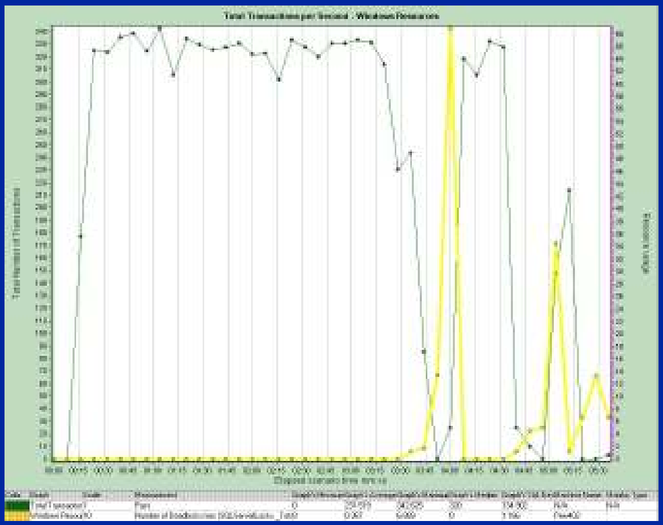


Color	Graph	Scale	Measurement	Graph's Minimum	Graph's Average	Graph's Maximum	Graph's Median	Graph's Std. Dev	Machine Name	Monitor Type
▨	Windows Resou	1	% Processor Time (Processor _Total):pew402	0.116	2.529	4.167	2.908	1.062	Pew402	
■	UNIX Resources	1	CPU Utilization (Unix Kernel Statistics):pew202	0.023	14.792	33.053	15.375	9.413	Pew202	

There were a lot of available resources on both the application and database servers. The maximum database server CPU utilization was 4.2% (see left side for the scale), and the maximum application server CPU utilization was 33% (see right side for the scale). Attention: each graph has a different scale.

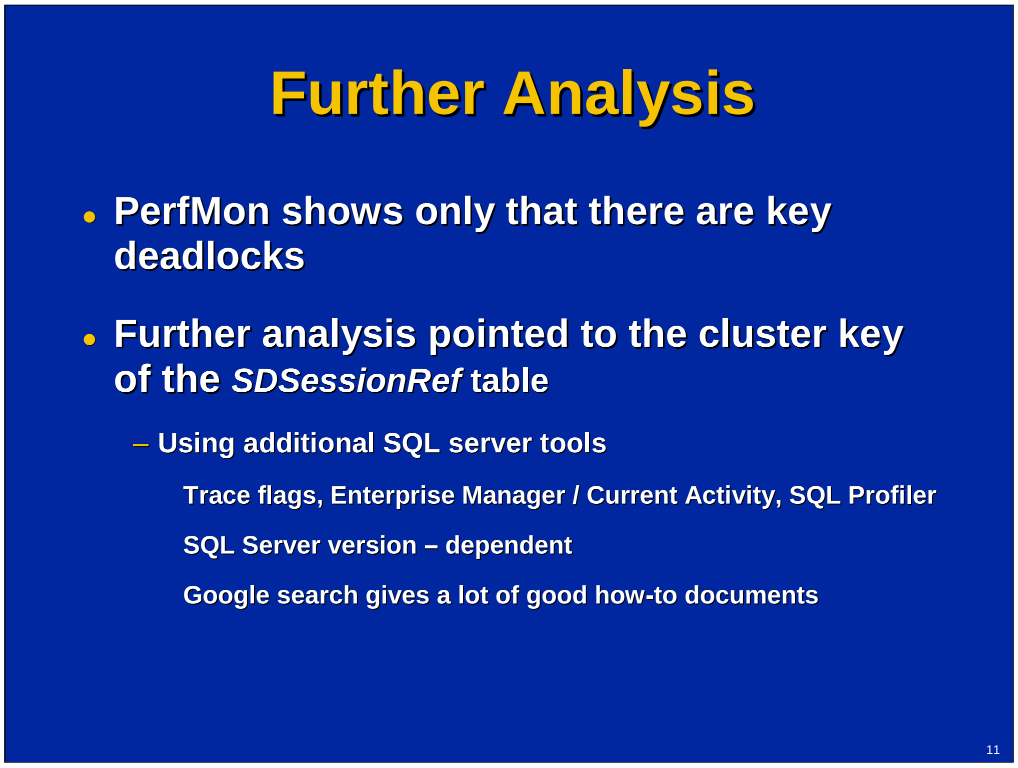
CPU utilization on both machines dropped as throughput dropped. Heap memory usage was also minimal (it was about 100MB with max heap size set to 1024M). So nothing confirmed the garbage collection theory.

Drops are Correlated with Deadlocks



10

Looking through multiple PerfMon graphs (and using the LoadRunner auto-correlation feature that worked well in this particular case) I finally got to the real cause of drops in throughput: deadlocks in the SQL Server database.



Further Analysis

- PerfMon shows only that there are key deadlocks
- Further analysis pointed to the cluster key of the *SDSessionRef* table
  - Using additional SQL server tools
    - Trace flags, Enterprise Manager / Current Activity, SQL Profiler
    - SQL Server version – dependent
    - Google search gives a lot of good how-to documents

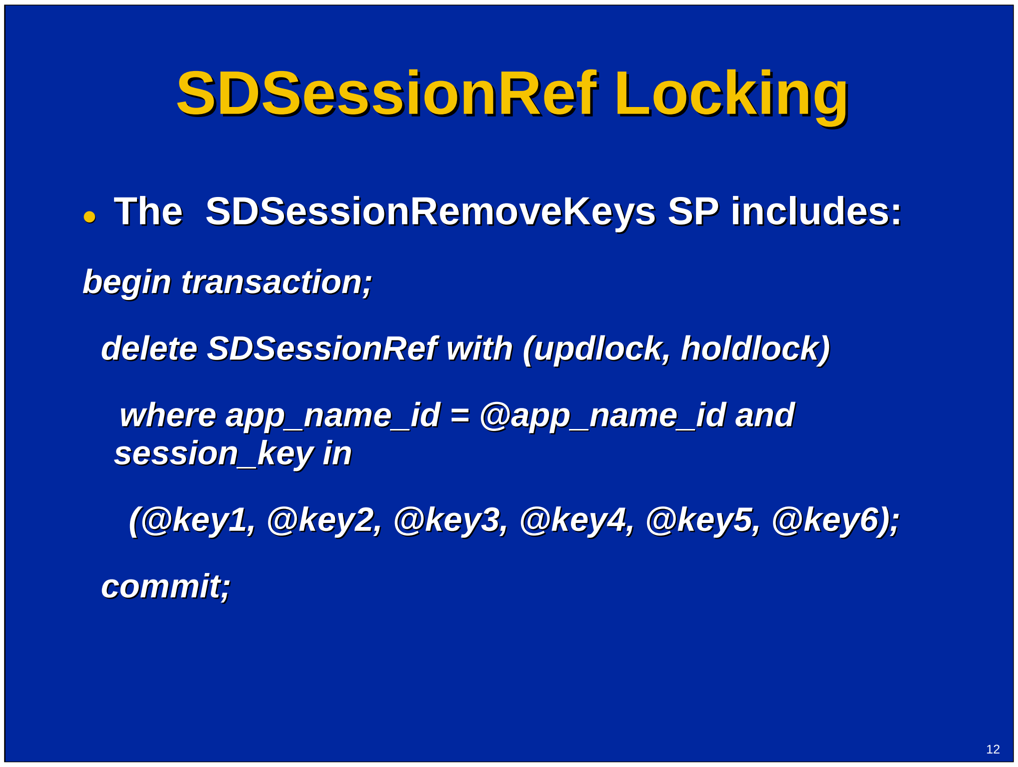
11

PerfMon shows the number of deadlocks total and by type. So the only additional information I found from PerfMon was that it was key deadlocks.

I guess that in many organizations the task of performance engineer would stop here – probably the further analysis would be conducted by a DBA. Working in development where no DBAs were involved in the project, I needed to do such analysis myself.

Further analysis pointed to the cluster key of the *SDSessionRef* table. For further analysis SQL Server-specific tools were used, such as trace flags, Enterprise Manager / Current Activity, and SQL Profiler. A Google search found me a lot of good how-to documents.

These tools are SQL Server – version dependent. For some reason, Microsoft SQL Server 2000 was used. Trace flags, for example, were completely changed in SQL Server 2005. So I don't dive into exact steps used to get this information.



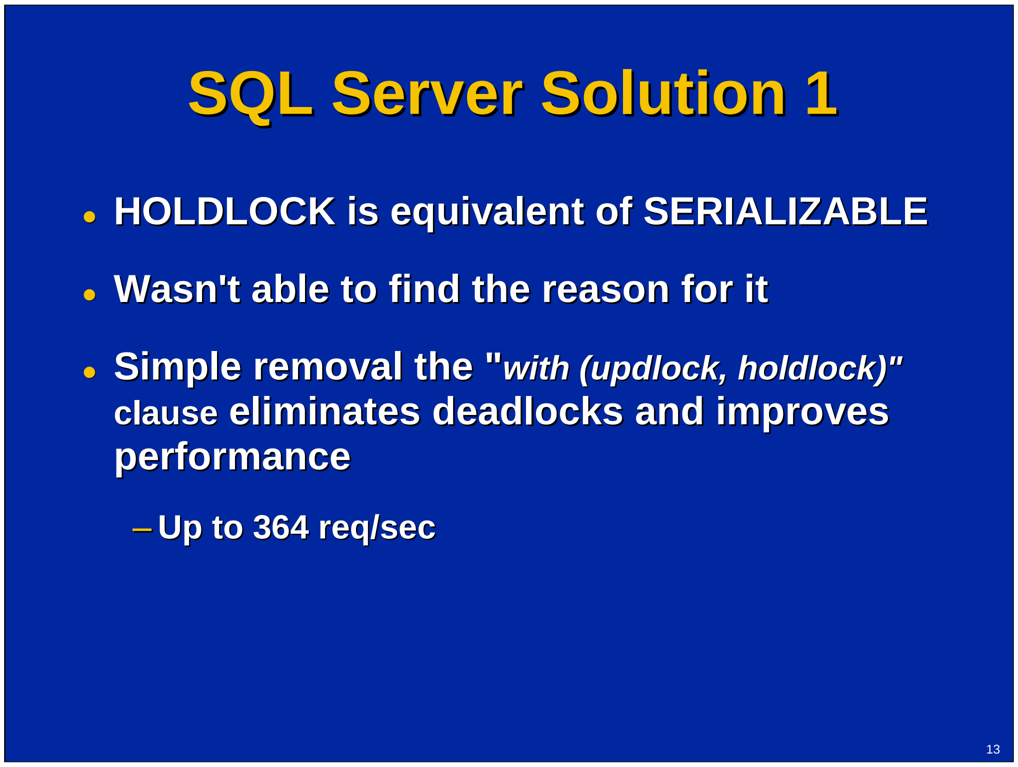
SDSessionRef Locking

- The SDSessionRemoveKeys SP includes:

```
begin transaction;
  delete SDSessionRef with (updlock, holdlock)
    where app_name_id = @app_name_id and
   session_key in
     (@key1, @key2, @key3, @key4, @key5, @key6);
  commit;
```

12

The deadlocks happened around the SDSessionRef table. Looking into the SDSessionRemoveKeys stored procedure (which was causing deadlocks) I noticed that *with (updlock, holdlock)* hints were used.



SQL Server Solution 1

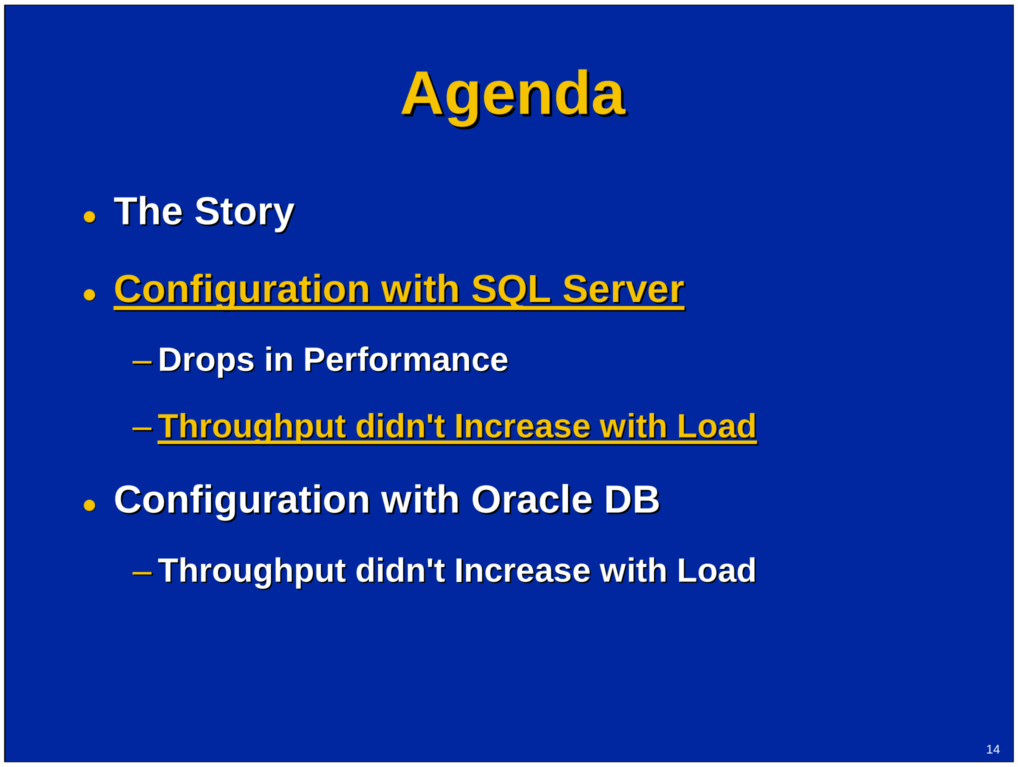
- HOLDLOCK is equivalent of SERIALIZABLE
- Wasn't able to find the reason for it
- Simple removal the "*with (updlock, holdlock)"* clause eliminates deadlocks and improves performance
  - Up to 364 req/sec

13

It was using the highest isolation level with the HOLDLOCK hint. From the SQL Server documentation: *Makes shared locks more restrictive by holding them until a transaction is completed, instead of releasing the shared lock as soon as the required table or data page is no longer needed, whether the transaction has been completed or not.*

During the following discussion, nobody was able to provide a reason for it. Simple removal of the "*with (updlock, holdlock)"* clause eliminated deadlocks and improved performance up to 364 requests/sec.

While I was able to find the problem here, suggest a solution and even test it, I realized during further discussions that the developers were uncomfortable with me going so deep. Probably I should just provide information on where the deadlocks happen and let the developers find and fix the issue.



Agenda

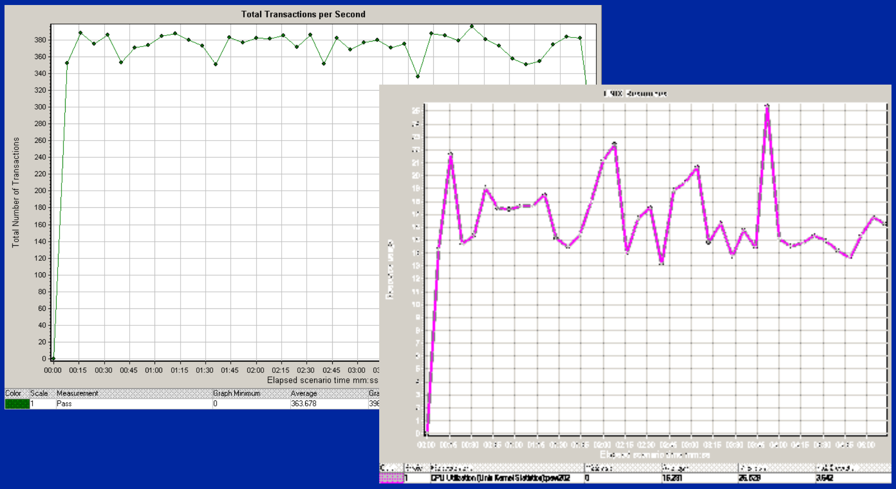
- The Story
- Configuration with SQL Server
  - Drops in Performance
  - Throughput didn't Increase with Load
- Configuration with Oracle DB
  - Throughput didn't Increase with Load

14

Solving the deadlock problem didn't fully solve the scalability issue.

SQL Server Issue 2

- A lot of resources available – but throughput doesn't increase when load increases



Color	Scale	Measurement	Graph Minimum	Average	Gra
🟩	1	Pass	0	363.678	396

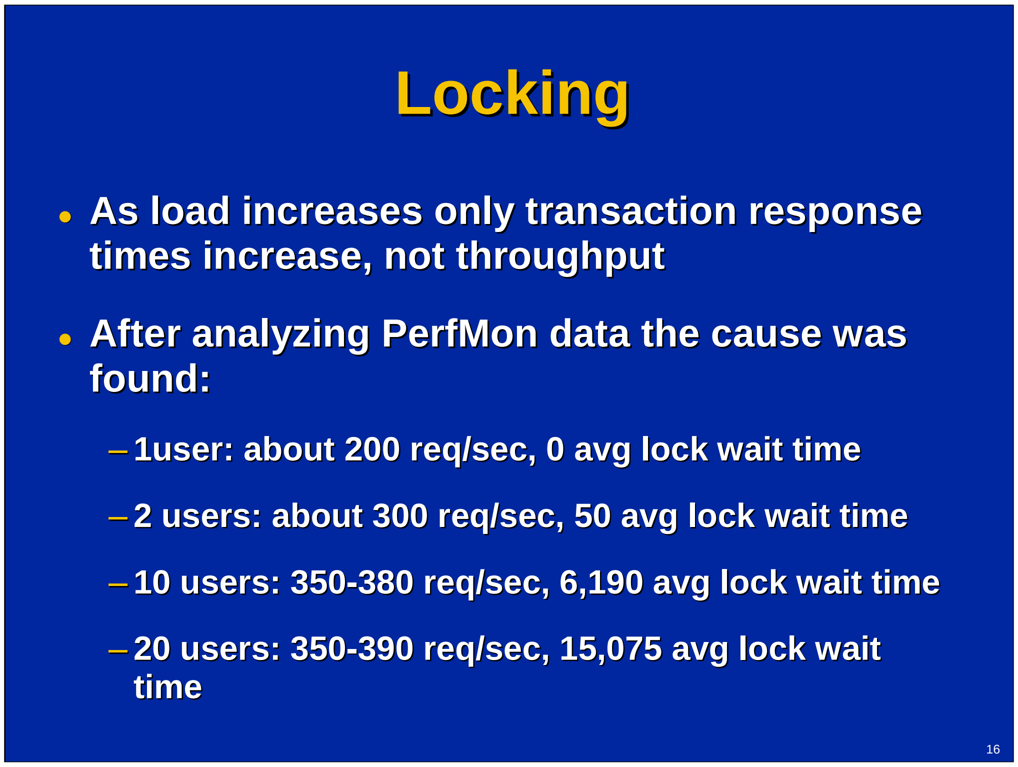
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	1	CPU Utilization (Unix Kernel Statistics):pew202	0	16.281	26.628	3.642

15

After fixing the deadlocks problem, the next issue was that throughput (the number of requests/sec) doesn't increase after a certain point when load increases. There were a lot of resources available: the CPU utilization on the application server was less than 17%.

It is important to understand that the performance of the system was actually very good. For example, on the throughput graph we see an average performance of 364 requests / sec on pretty basic equipment. If we speak about a call center, it means 91 calls /sec (as far as a call consists of 4 requests) or 327,600 calls per hour. Of course, this is for the Sample model - as models become more sophisticated, performance degrades. But the Sample model wasn't simplistic – it included all typical functionality.

The reason for the project was not customer complains about performance – actually for typical models the performance was surprisingly good – but rather the necessity of providing some sizing / capacity planning recommendations to customers. It was a paradoxical situation: from one side, the system demonstrates very good performance with minimal resources (not the situation you see very often as a performance engineer), but from another side it can't scale further or even use all resources on the system. So it is very difficult to provide any meaningful sizing recommendations. The very good performance of the system was actually the reason that the issues discussed here were not investigated before – it just wasn't a priority as compared with functionality.



Locking

- As load increases only transaction response times increase, not throughput
- After analyzing PerfMon data the cause was found:
  - 1user: about 200 req/sec, 0 avg lock wait time
  - 2 users: about 300 req/sec, 50 avg lock wait time
  - 10 users: 350-380 req/sec, 6,190 avg lock wait time
  - 20 users: 350-390 req/sec, 15,075 avg lock wait time

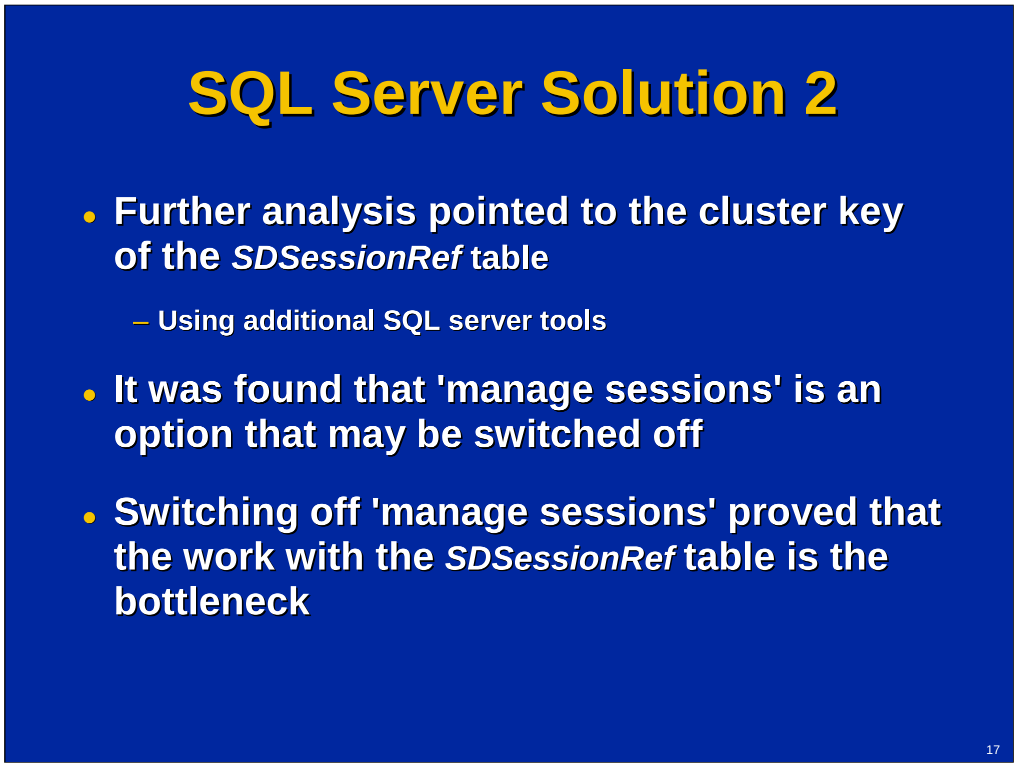
16

After a certain point, further increasing load increased only transaction response times, not throughput. Throughput stayed the same and then started to degrade.

After further analysis of PerfMon data the cause was found: with load increase users just spent more time waiting for locks.

1user: about 200 requests/sec, 0 average lock wait time
2 users: about 300 requests/sec, 50 average lock wait time
10 users: 350-380 requests/sec, 6,190 average lock wait time
20 users: 350-390 requests/sec, 15,075 average lock wait time

Lock wait time (ms): Total wait time (milliseconds) for locks in the last second.



SQL Server Solution 2

- Further analysis pointed to the cluster key of the *SDSessionRef* table
  - Using additional SQL server tools
- It was found that 'manage sessions' is an option that may be switched off
- Switching off 'manage sessions' proved that the work with the *SDSessionRef* table is the bottleneck

17

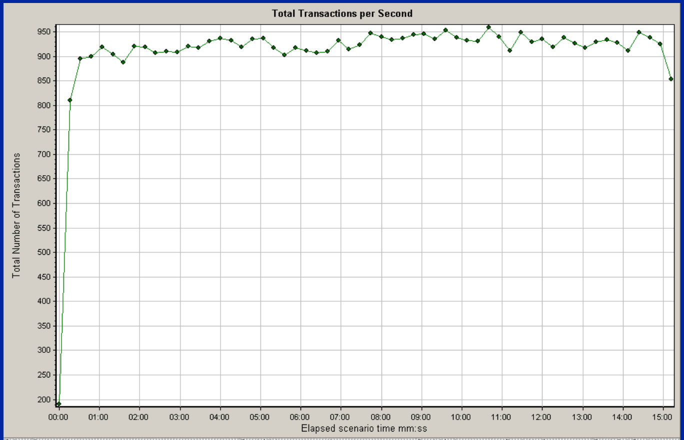
Using SQL Server specific tools it was found that users waited for locks for the cluster key of the SDSessionRef table.

In the beginning of every session a record is inserted into the SDSessionRef table and in the end this record is deleted. Considering the high number of short sessions, it is not surprising that this particular table (cluster key to be exact) became an issue.

Discussing the functionality implemented with this table it was found that actually it may be switched off if there is another way of supporting sessions (for example, by a load balancer). A test was run with the 'manage sessions' option off that fully confirmed that the table is the bottleneck.

Manage Sessions Off

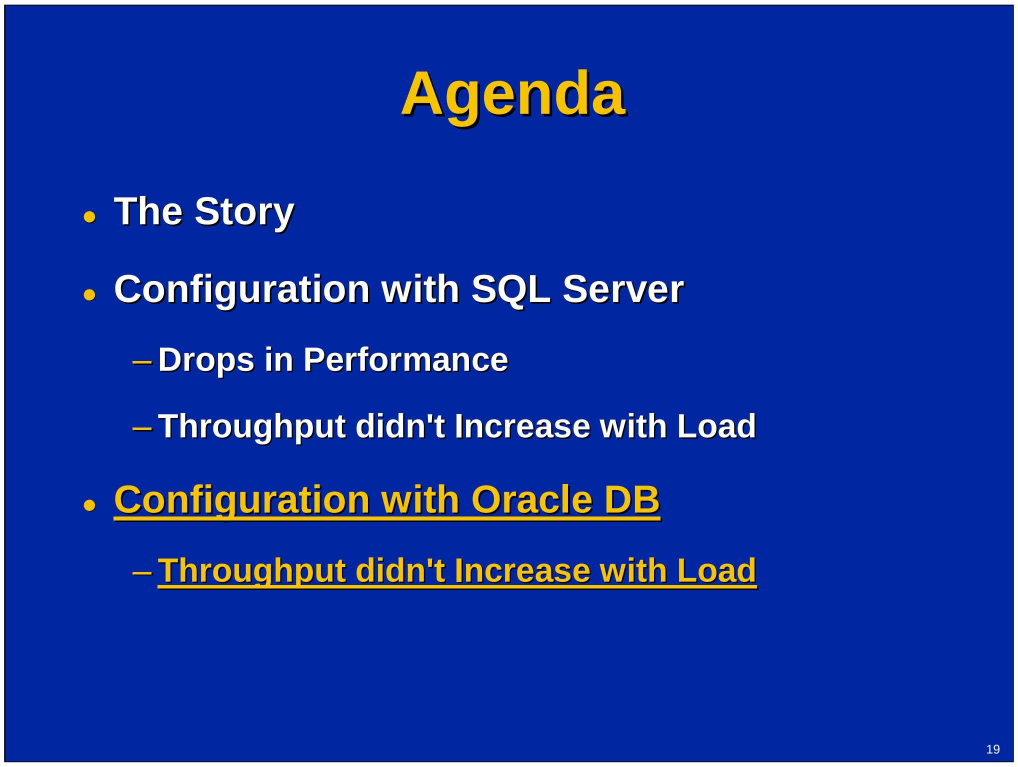
- Throughput grew to 910 req/sec



Color	Scale	Measurement	Graph Minimum	Average	Graph Maximum	Graph Median	Graph Std. Deviation
🟩	1	Pass	190.375	909.784	958.875	926.188	97.987

18

System throughput grew to 910 requests/second (from 364 requests/sec).

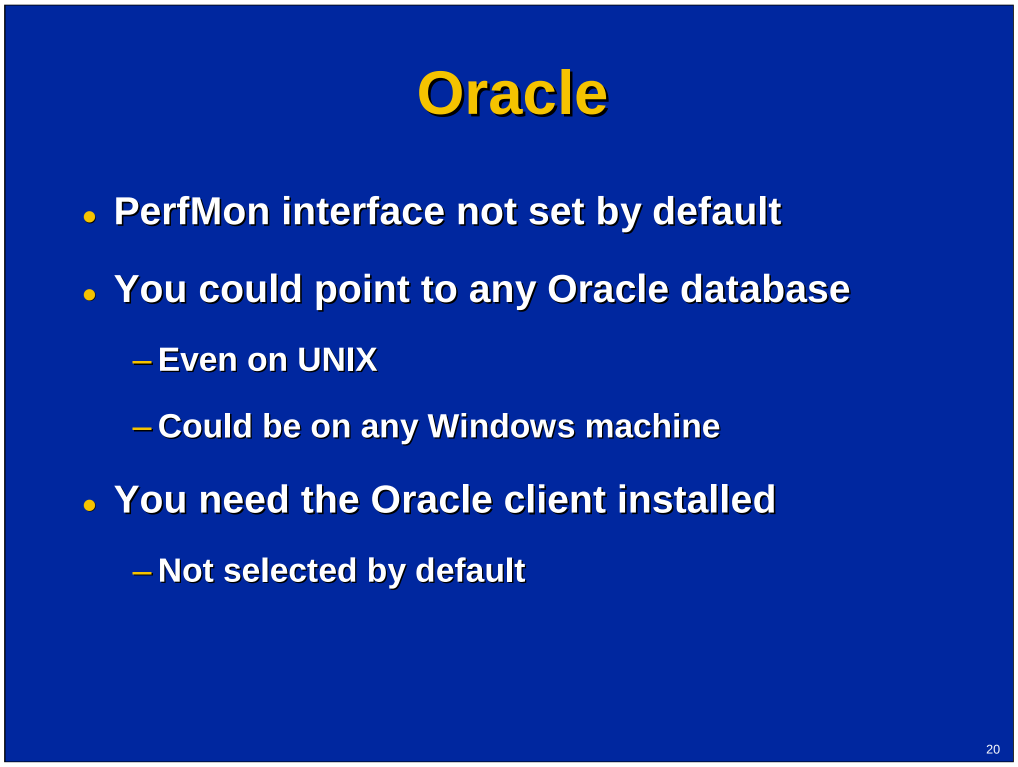


Agenda

- The Story
- Configuration with SQL Server
  - Drops in Performance
  - Throughput didn't Increase with Load
- Configuration with Oracle DB
  - Throughput didn't Increase with Load

19

Another part of the project was to test the product with Oracle Database.

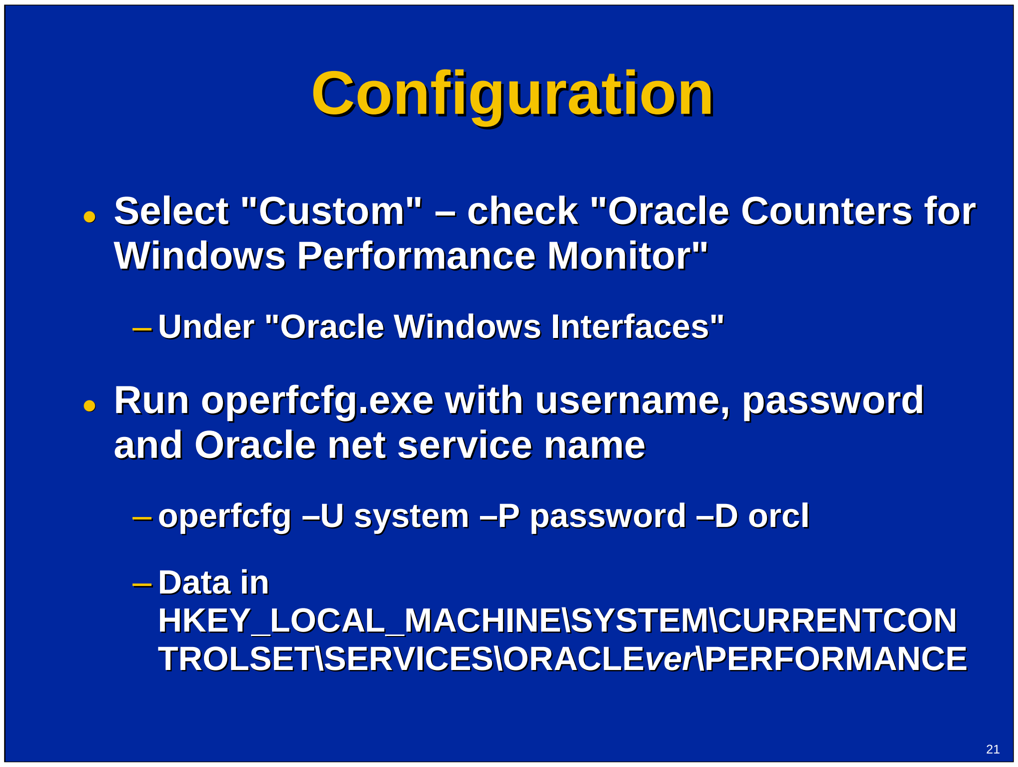


Oracle

- PerfMon interface not set by default
- You could point to any Oracle database
  - Even on UNIX
  - Could be on any Windows machine
- You need the Oracle client installed
  - Not selected by default

20

It is possible to use PerfMon to monitor any Oracle Database, even if it is on a UNIX server. PerfMon is not set by default; you need to do a custom installation and chose it explicitly.



Configuration

- Select "Custom" – check "Oracle Counters for Windows Performance Monitor"
  - Under "Oracle Windows Interfaces"
- Run operfcfg.exe with username, password and Oracle net service name
  - operfcfg –U system –P password –D orcl
  - Data in HKEY_LOCAL_MACHINE\SYSTEM\CURRENTCONTROLSET\SERVICES\ORACLE*ver*\PERFORMANCE

21

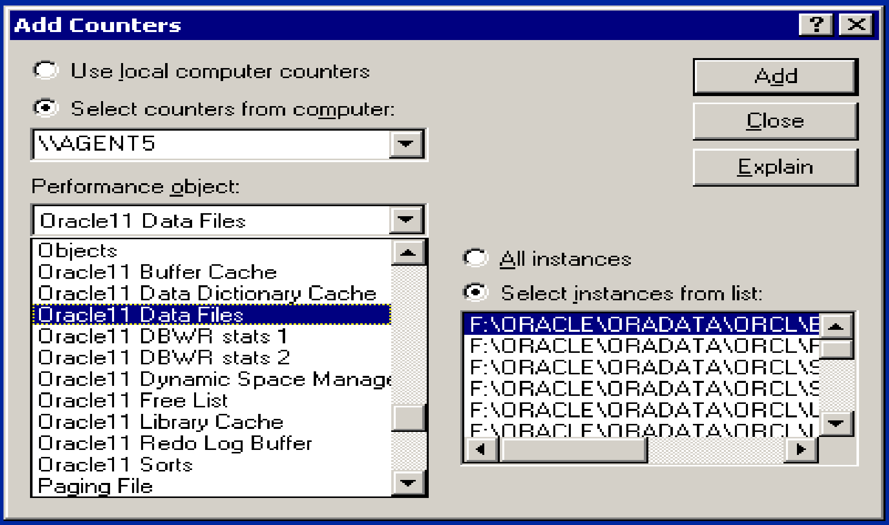
You should choose the custom installation and under "Oracle Windows Interfaces" check "Oracle Counters for Windows Performance Monitor".

After installation, you need to run operfcfg.exe with username, password, and Oracle net service name:
  operfcfg –U system –P password –D orcl

All configuration-related information is saved in the registry, HKEY_LOCAL_MACHINE\SYSTEM\CURRENTCONTROLSET\SERVICES\ORACLE*ver*\PERFORMANCE

More details could be found in Oracle documentation (Oracle Database Platform Guide [version] for Windows, chapter 6. Monitoring a Database on Windows). See also Edward Whalen's blog http://ewhalen.blogspot.com/2006/06/oracle-performance-monitoring-on.html or Thirumoorthy Chettiannan's blog http://perfhints.blogspot.com/2009/03/monitoring-oracle-databse-using-perfmon.html

And Here They Are



Use local computer counters [✓ Select counters from computer:]

\\AGENT5

Performance object:

Oracle11 Data Files

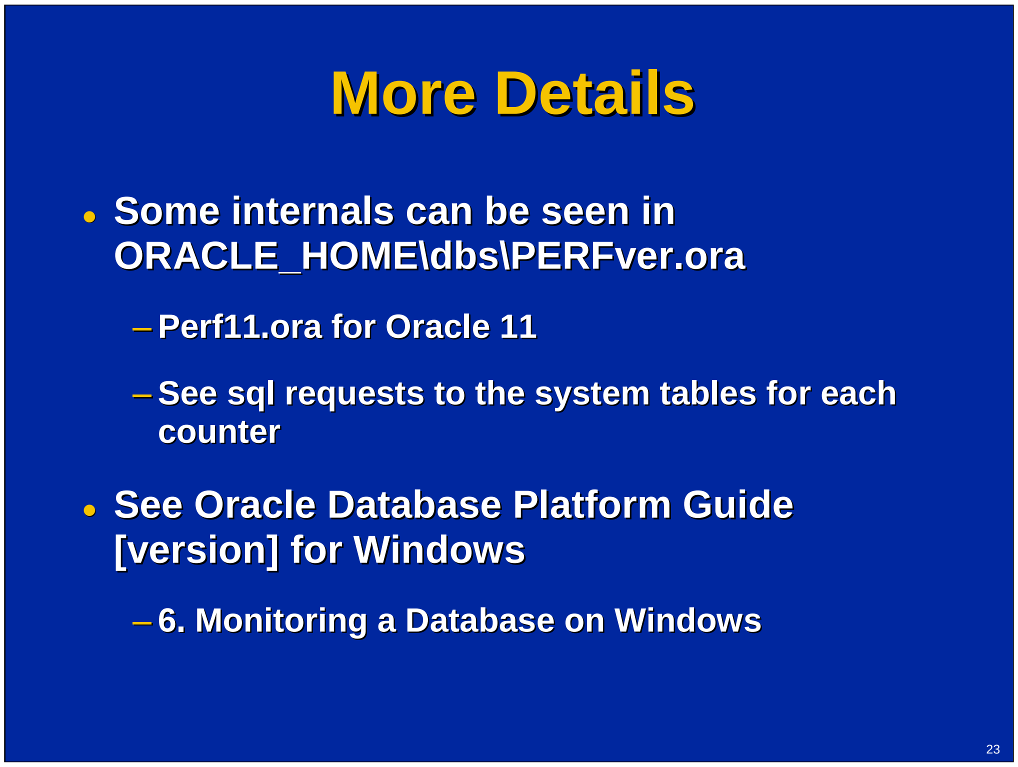
- Objects
- Oracle11 Buffer Cache
- Oracle11 Data Dictionary Cache
- Oracle11 Data Files
- Oracle11 DBWR stats 1
- Oracle11 DBWR stats 2
- Oracle11 Dynamic Space Manage
- Oracle11 Free List
- Oracle11 Library Cache
- Oracle11 Redo Log Buffer
- Oracle11 Sorts
- Paging File

All instances [✓ Select instances from list:]

- F:\ORACLE\ORADATA\ORCL\E
- F:\ORACLE\ORADATA\ORCL\F
- F:\ORACLE\ORADATA\ORCL\S
- F:\ORACLE\ORADATA\ORCL\S
- F:\ORACLE\ORADATA\ORCL\U
- F:\ORACLE\ORADATA\ORCL\U

22

And you see performance objects for Oracle for the machine where Oracle Counters for Windows Performance Monitor is installed, not the machine where the actual Oracle database is. You just point Oracle Counters for Windows Performance Monitor to where the server is with the operfcfg utility.


More Details

- Some internals can be seen in ORACLE_HOME\dbs\PERFver.ora
  - Perf11.ora for Oracle 11
  - See sql requests to the system tables for each counter
- See Oracle Database Platform Guide [version] for Windows
  - 6. Monitoring a Database on Windows

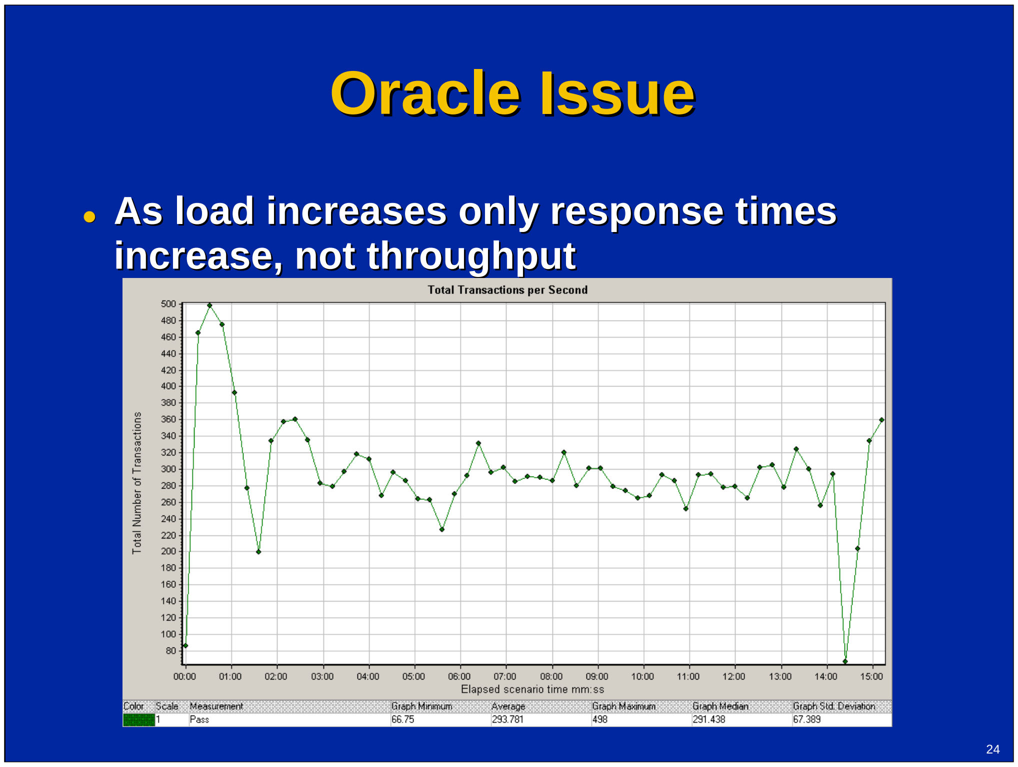
23

All counters are explained in detail in Oracle Database Platform Guide [version] for Windows, chapter 6. Monitoring a Database on Windows.

Some internals can be seen in the ORACLE_HOME\dbs\PERFver.ora file (Perf11.ora for Oracle 11g). There you can see actual sql requests to the system tables for each counter.

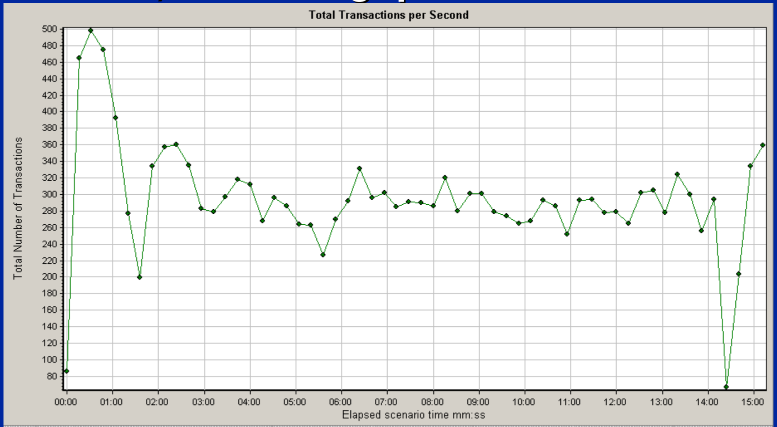
Still it is very important to understand that there is a limited number of counters exposed through the PerfMon interface. But using PerfMon for monitoring databases has several advantages for performance engineers / testers:

- -Collecting all performance information in one place
- -Getting some DB-related metrics on early stages without DBAs or DBA-level tools
- -If issues are observed, there is already some information available pointing to the area for further investigation



Oracle Issue

- As load increases only response times increase, not throughput



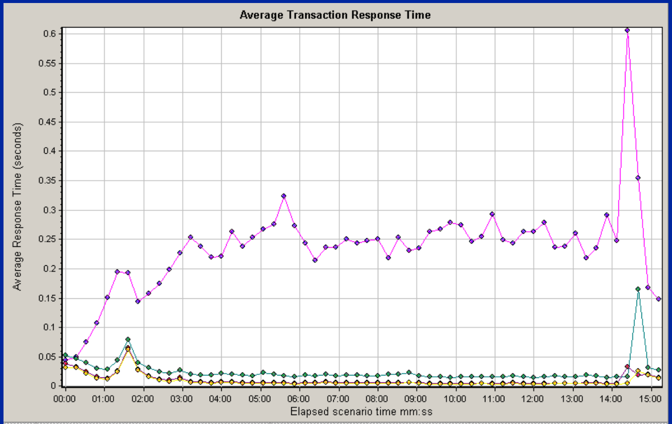
Color	Scale	Measurement	Graph Minimum	Average	Graph Maximum	Graph Median	Graph Std. Deviation
🟩	1	Pass	66.75	293.781	498	291.438	67.389

24

There were the same symptoms for Oracle Database as for SQL Server: after a certain point, only response times increase as load increases, not throughput. The maximum was an average of 294 requests / sec.

CallResolution

- Mainly CallResolution response times growing

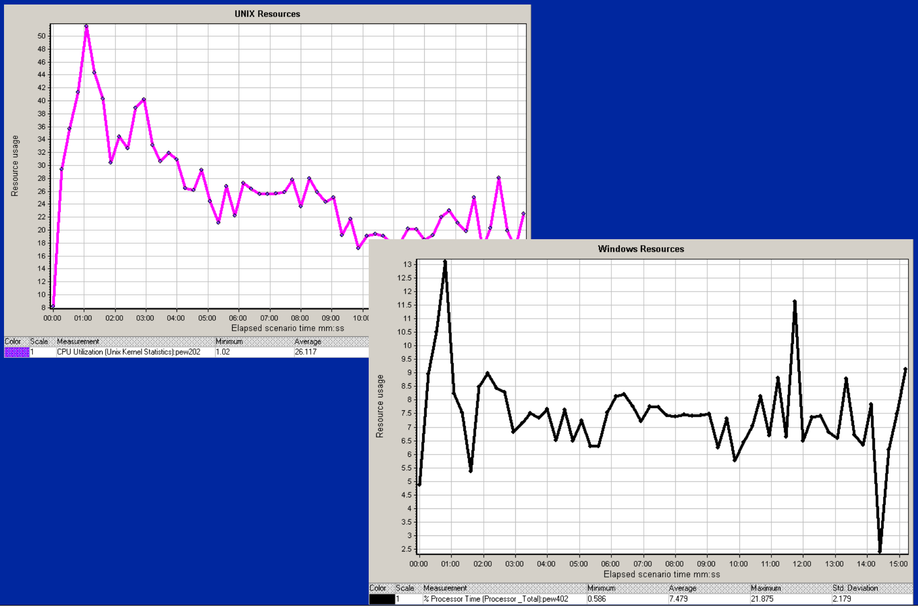


Color	Scale	Measurement	Minimum	Average	Maximum	Std. Deviation
🟪	1	CallResolution	0.012	0.223	15.885	0.194
🟩	1	CallStart	0.005	0.024	12.137	0.117
🟥	1	OfferRequest	0.002	0.01	7.234	0.033
🟨	1	OfferResponse	0.002	0.009	9.139	0.039

25

Mainly CallResolution response times were growing with load increase, response times of three other types of requests were growing insignificantly.

With Plenty of Resources Available



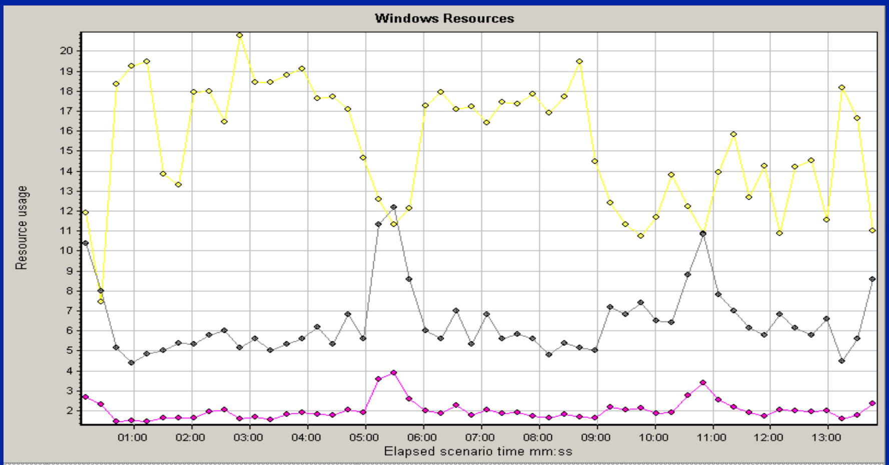
Color	Scale	Measurement	Minimum	Average
🟪	1	CPU Utilization (Unix Kernel Statistics):pew202	1.02	26.117

Color	Scale	Measurement	Minimum	Average	Maximum	Std. Deviation
⬛	1	% Processor Time (Processor _Total):pew402	0.586	7.479	21.875	2.179

26

And there were plenty of resources available: with 26.1% CPU utilization on the application server and 7.5% CPU utilization on the database server.

I/O Issue?



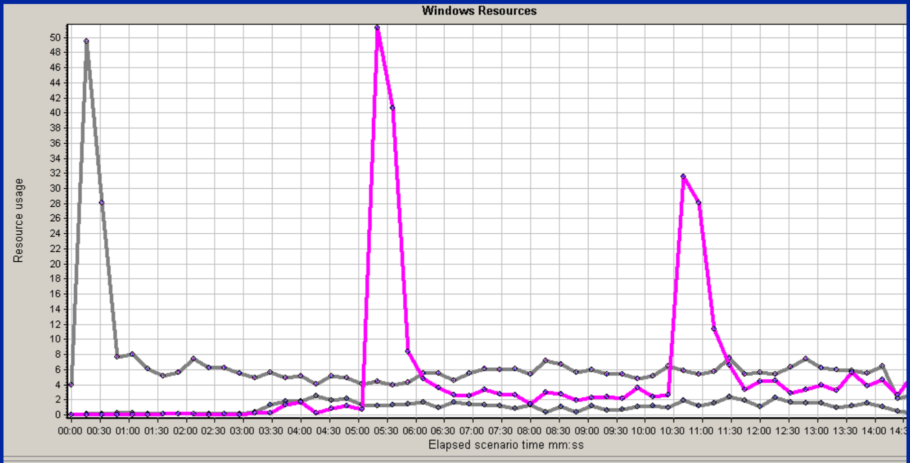
Color	Scale	Measurement	Minimum	Average	Maximum	Std. Deviation
🟨	1	% Idle Time (PhysicalDisk 1 F:):pew402	0.164	15.412	48.664	6.448
🟪	1	Avg. Disk Queue Length (PhysicalDisk 1 F:):pew40	1.134	2.023	6.88	0.759
⬛	1000	Avg. Disk sec/Transfer (PhysicalDisk 1 F:):pew402	0.003	0.006	0.024	0.003

27

There were some indications that I/O was high: Average Disk Queue Length was about 2 for the physical disk with the Oracle data files.

However other important I/O counters didn't indicate that I/O was the bottleneck: %Idle Time was 15.4% (so calculating the real disk utilization as 100-%Idle Time we get 84.6%) and Avg. Disk Sec/Transfer was 6 ms (3ms for read and 10 ms for write).

We Can See DB Files with High I/O

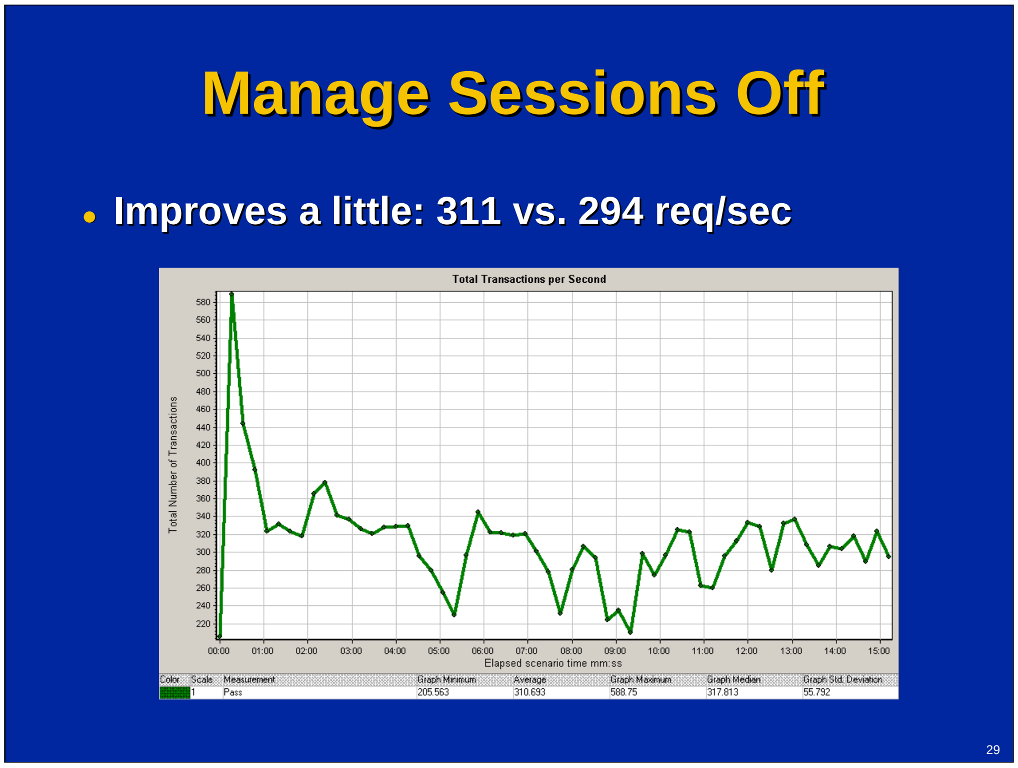


Legend

Color	Scale	Measurement	Minimum	Average	Maximum	Std. Deviation
✓ ▨	1	phyrds/sec (Oracle11 Data Files F\ORACLE\ORADATA\ORCL\USERS01.DBF):agent5	0	6.752	66.331	7.269
✓ ▨	1	phywrts/sec (Oracle11 Data Files F\ORACLE\ORADATA\ORCL\UNDOTBS01.DBF):agent5	0	1.065	4.655	1.173
✓ ▨	1	phywrts/sec (Oracle11 Data Files F\ORACLE\ORADATA\ORCL\USERS01.DBF):agent5	0	5.089	114.986	12.081

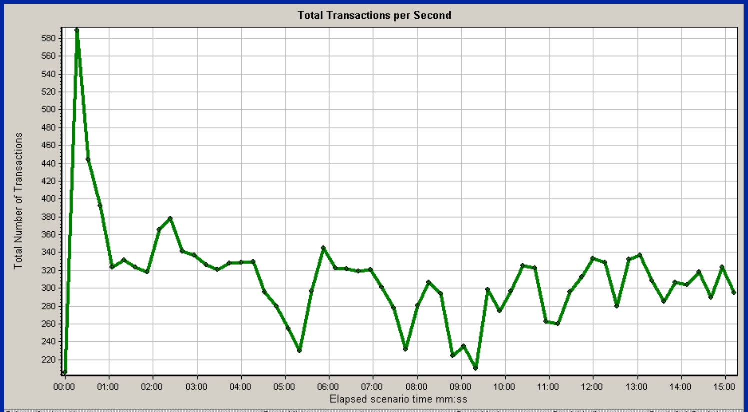
28

Oracle performance counters provide I/O information about individual database files.



Manage Sessions Off

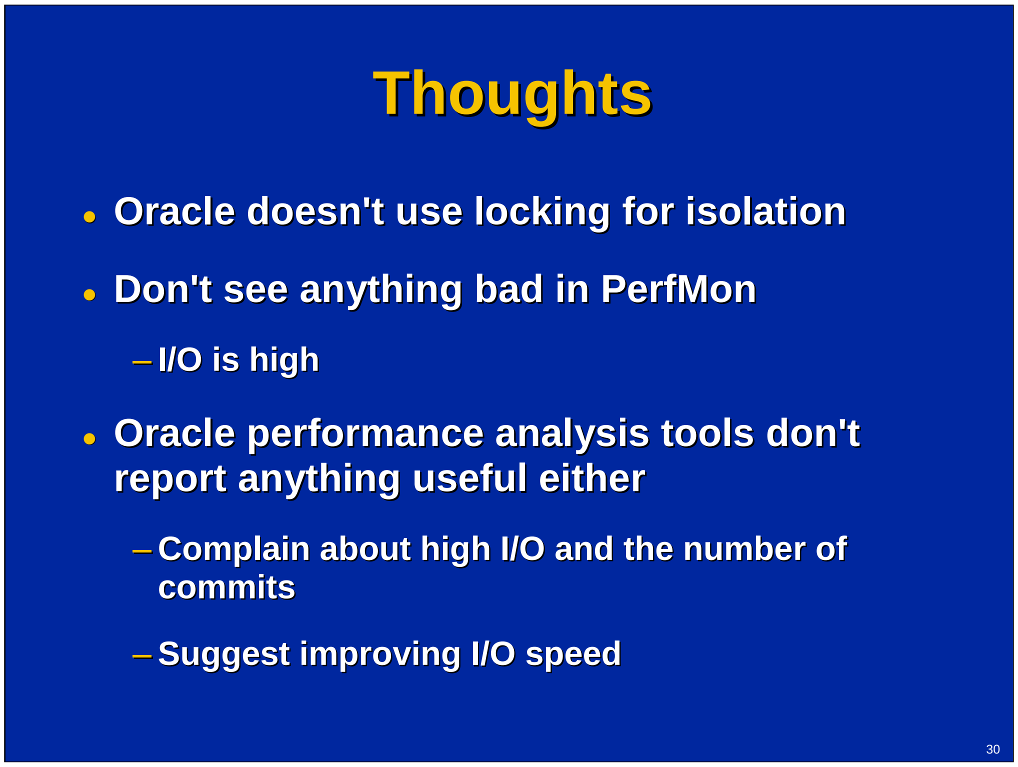
- Improves a little: 311 vs. 294 req/sec



Color	Scale	Measurement	Graph Minimum	Average	Graph Maximum	Graph Median	Graph Std. Deviation
🟩	1	Pass	205.563	310.693	588.75	317.813	55.792

29

But the root causes limiting throughput were different for Oracle Database. While for Microsoft SQL Server switching off the 'Manage Session' option improved performance drastically, for Oracle the improvement was insignificant: from 294 to 311 requests / second.



Thoughts

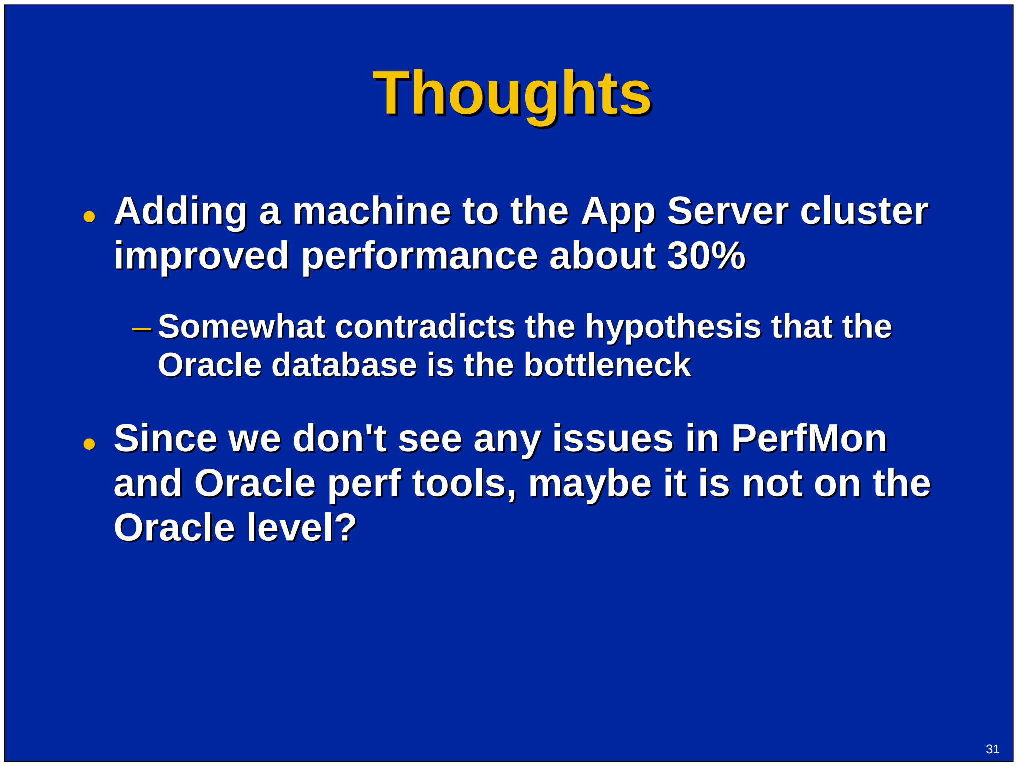
- Oracle doesn't use locking for isolation
- Don't see anything bad in PerfMon
  - I/O is high
- Oracle performance analysis tools don't report anything useful either
  - Complain about high I/O and the number of commits
  - Suggest improving I/O speed

30

Oracle uses versioning, not locking for isolation – so we obviously don't have locking bottleneck around the session table.

Don't see anything bad in PerfMon except high I/O.

Oracle performance analysis tools don't report anything useful either: complain about high I/O and the number of commits, suggest improving I/O speed.



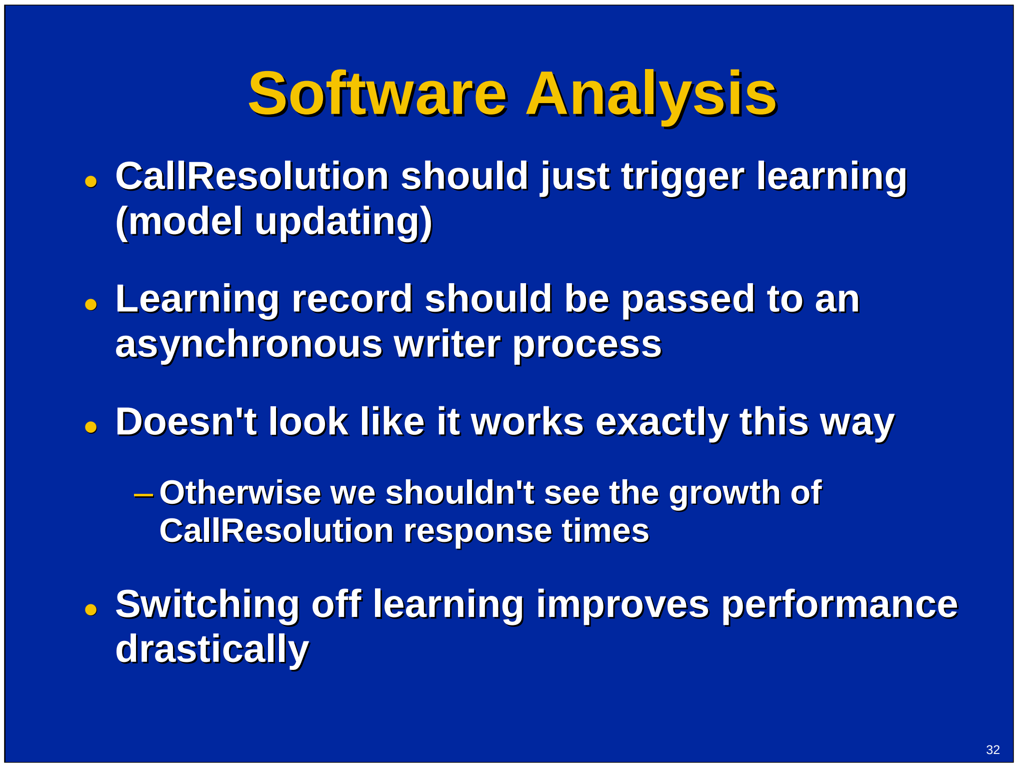
Thoughts

- Adding a machine to the App Server cluster improved performance about 30%
  - Somewhat contradicts the hypothesis that the Oracle database is the bottleneck
- Since we don't see any issues in PerfMon and Oracle perf tools, maybe it is not on the Oracle level?

31

Adding a machine to the App Server cluster improved performance about 30% that somewhat contradicts the hypothesis that the Oracle database is the bottleneck.

Since we don't see any issues in PerfMon and Oracle perf tools, maybe it is not on the Oracle level?



Software Analysis

- CallResolution should just trigger learning (model updating)
- Learning record should be passed to an asynchronous writer process
- Doesn't look like it works exactly this way
  - Otherwise we shouldn't see the growth of CallResolution response times
- Switching off learning improves performance drastically

32

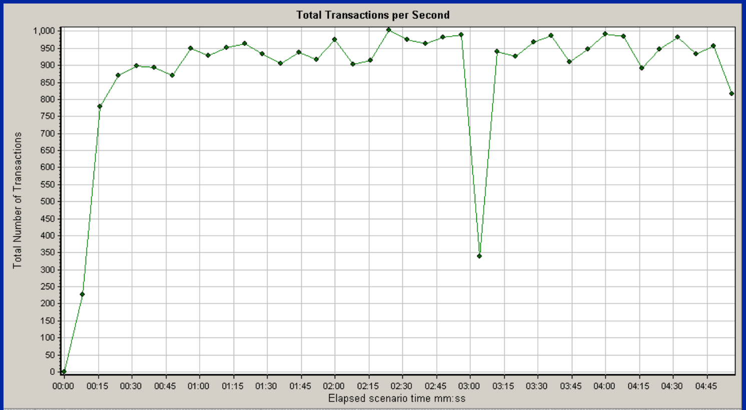
CallResolution should just trigger "learning" (model updating). "Learning" is processing of results of all transactions and updating the information used for decision making. The learning record should be passed to an asynchronous writer process.

It doesn't look like it works exactly this way; otherwise we shouldn't see the growth of CallResolution response times.

Switching off "learning" improves performance drastically.

Switching Off Learning

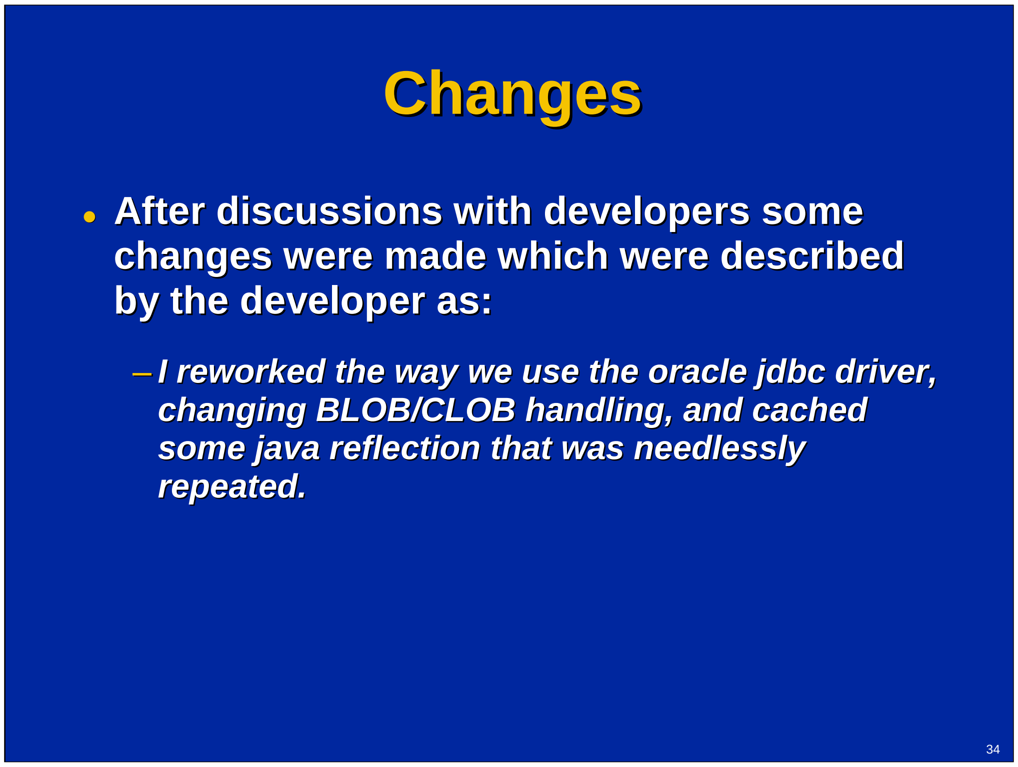
- Improves performance drastically



Color	Scale	Measurement	Graph Minimum	Average	Graph Maximum	Graph Median	Graph Std. Deviation
🟩	1	Pass	0	875.603	1003.375	937.625	210.047

33

Switching off "learning", the processing of results of all transactions and updating the information used for decision making, improves the performance greatly (up to 876 requests / sec). Although it is definitely not an option for the real deployments: it disables one of the main advantages of the system, self-learning.



Changes

- After discussions with developers some changes were made which were described by the developer as:
  - *I reworked the way we use the oracle jdbc driver, changing BLOB/CLOB handling, and cached some java reflection that was needlessly repeated.*

34

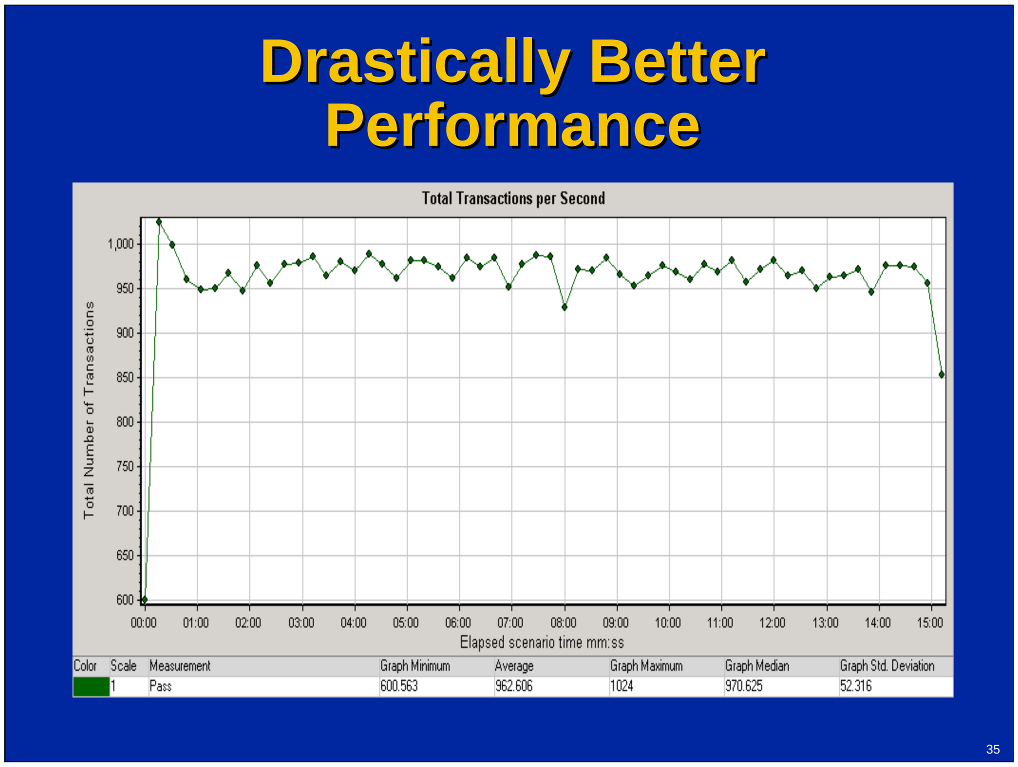
After discussions with developers some changes were made which drastically improved performance. The changes were described by the developer as:

I reworked the way we use the oracle jdbc driver, changing BLOB/CLOB handling, and cached some java reflection that was needlessly repeated.

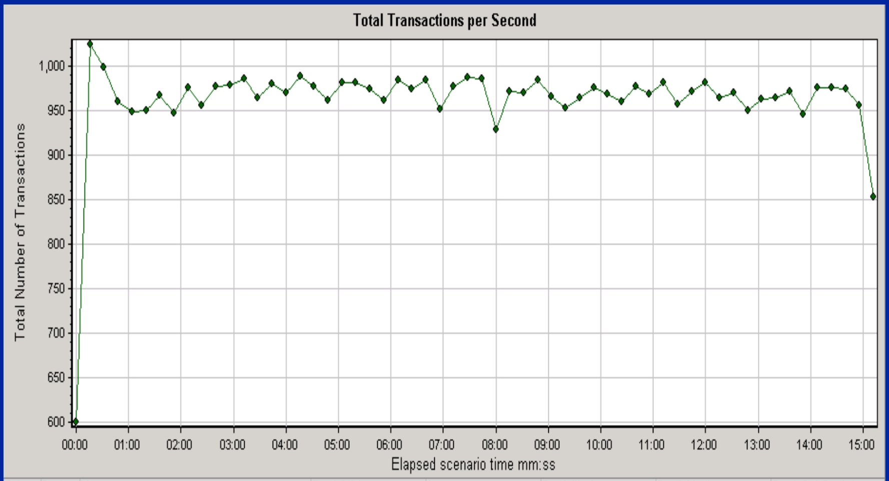
Considering my understanding that I already crossed the line in this project by going too deeply into details, I didn't ask for further clarifications. The result was definitely a great success, but the developers could be uncomfortable revealing all findings.

BLOB (Binary Large Objects) / CLOB (Character Large Object) used in the system are not very large and were never indicated as an issue.

Considering the complains of Oracle diagnostics tools about the high number of commits, one question was how commits were implemented. According, for example, to Coskan Gundogar's blog http://coskan.wordpress.com/2007/03/14/autocommit-with-jdbc-connections/ it should be set properly at JDBC level – it is autocommit by default. Having autocommit for each SQL request would kill any performance advantage of batch asynchronous processing.



Drastically Better Performance

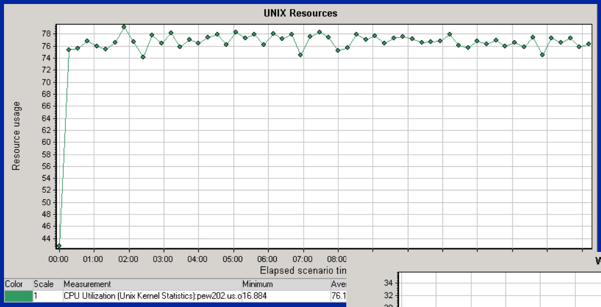


Color	Scale	Measurement	Graph Minimum	Average	Graph Maximum	Graph Median	Graph Std. Deviation
🟩	1	Pass	600.563	962.606	1024	970.625	52.316

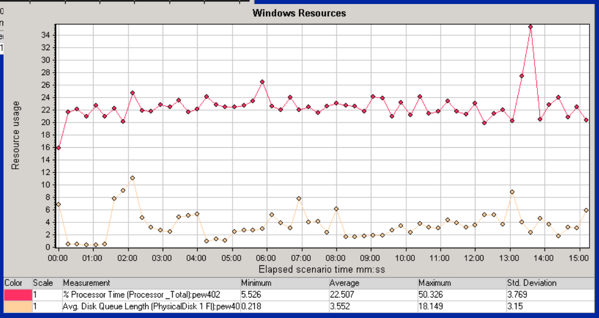
35

As the result of the changes performance drastically improved to 963 requests/sec – and this time with full functionality including "learning".

New Resource Consumption



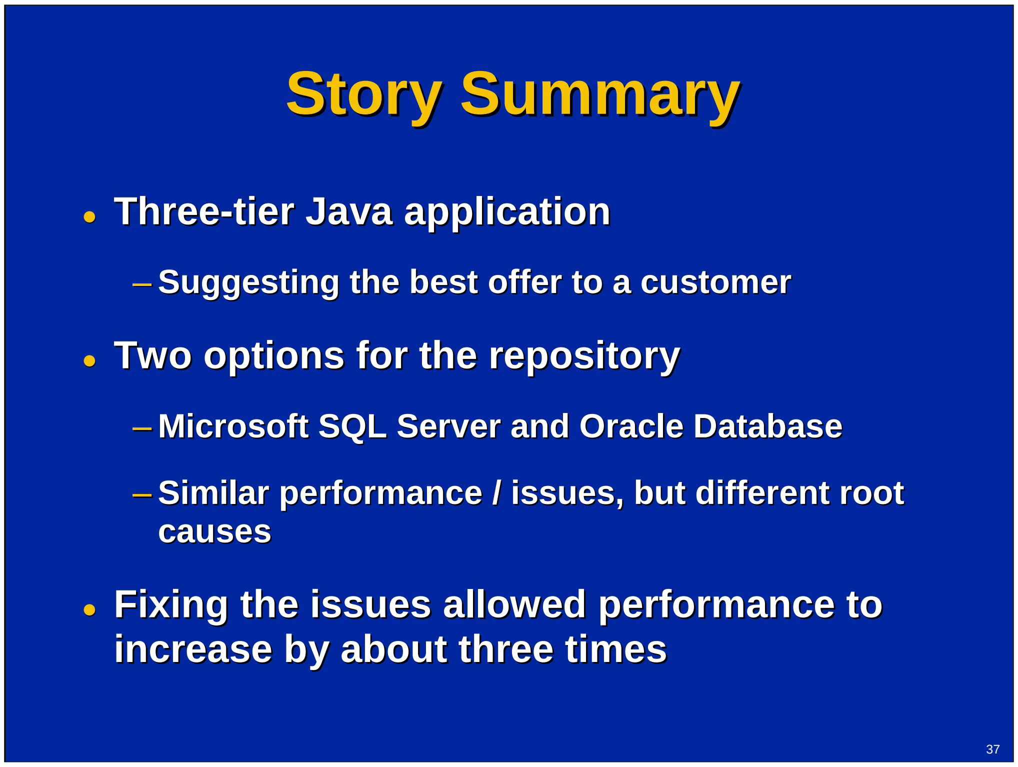
Color	Scale	Measurement	Minimum	Ave
🟩	1	CPU Utilization (Unix Kernel Statistics):pew202.us.o	16.884	76.1



Color	Scale	Measurement	Minimum	Average	Maximum	Std. Deviation
🟥	1	% Processor Time (Processor _Total):pew402	5.526	22.507	50.326	3.769
🟧	1	Avg. Disk Queue Length (PhysicalDisk 1 F:):pew40	0.218	3.552	18.149	3.15

36

Application server CPU utilization grew to 76.1%, database server CPU utilization – to 22.5%, average disk queue – to 3.5.

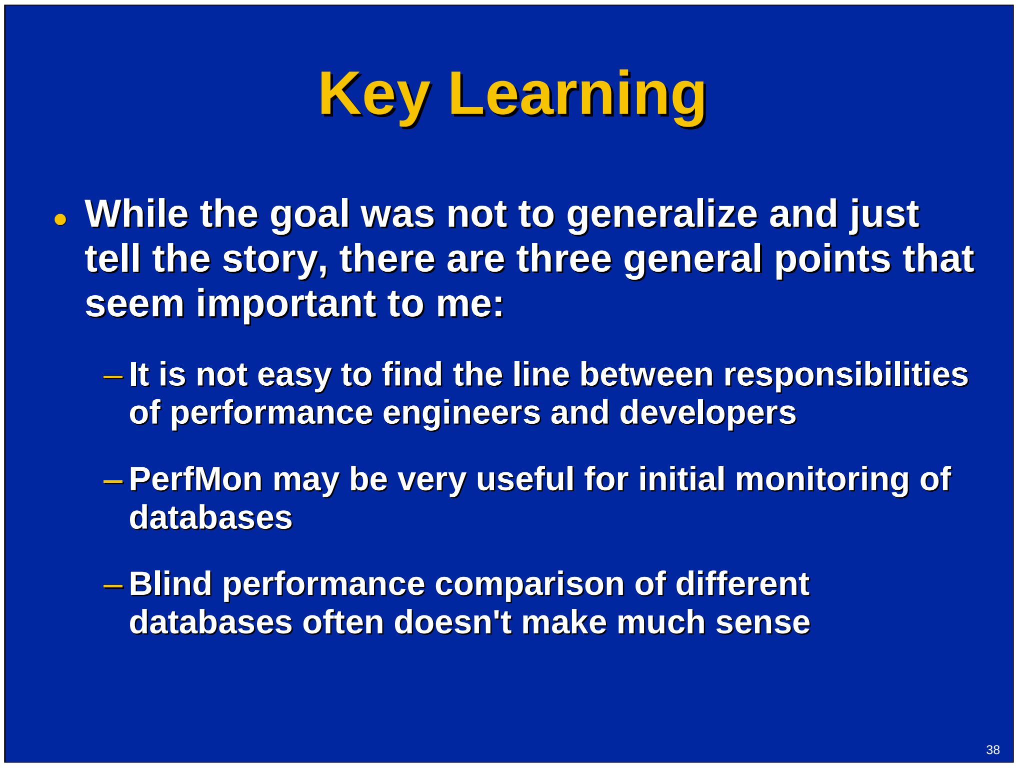


Story Summary

- Three-tier Java application
  - Suggesting the best offer to a customer
- Two options for the repository
  - Microsoft SQL Server and Oracle Database
  - Similar performance / issues, but different root causes
- Fixing the issues allowed performance to increase by about three times

37

This presentation described a performance engineering project in chronological order. The product under investigation was a three-tier Java application which suggests the best offer to a customer according to provided criteria. The performance issues found turned out to be database-related. Two configuration options were investigated for the repository: Microsoft SQL Server and Oracle Database. PerfMon was used for initial monitoring. While performance issues looked similar for both databases, the root causes were different: locking for SQL Server and coding for Oracle Database. Fixing the issues allowed to increase performance about three times for both configurations.



Key Learning

- While the goal was not to generalize and just tell the story, there are three general points that seem important to me:
  - It is not easy to find the line between responsibilities of performance engineers and developers
  - PerfMon may be very useful for initial monitoring of databases
  - Blind performance comparison of different databases often doesn't make much sense

38

While the goal was not to generalize and just tell the story, there are three general points that seem important to me:

1) It is not easy to find the line between responsibilities of performance engineers and developers, and how far performance engineer should go with investigations. It looks like going too deep may hurt too.

2) PerfMon may be very useful for initial monitoring of databases. Using PerfMon for monitoring databases has several advantages for performance engineers / testers:
-Collecting all performance information in one place.
-Getting some DB-related metrics on early stages without DBAs or DBA-level tools
-If issues are observed, there is already some information available pointing to the area for further investigation.

3) Blind performance comparison of different databases often doesn't make much sense. We had very similar issues for two databases, but further investigation revealed that the root causes were completely different. The architectures of databases are so different, so the question "which is faster" doesn't make much sense. The question is how good the specific product is using advantages of each database and how well it is tuned.

Questions?

Alexander Podelko

apodelko@yahoo.com

http://www.alexanderpodelko.com

39