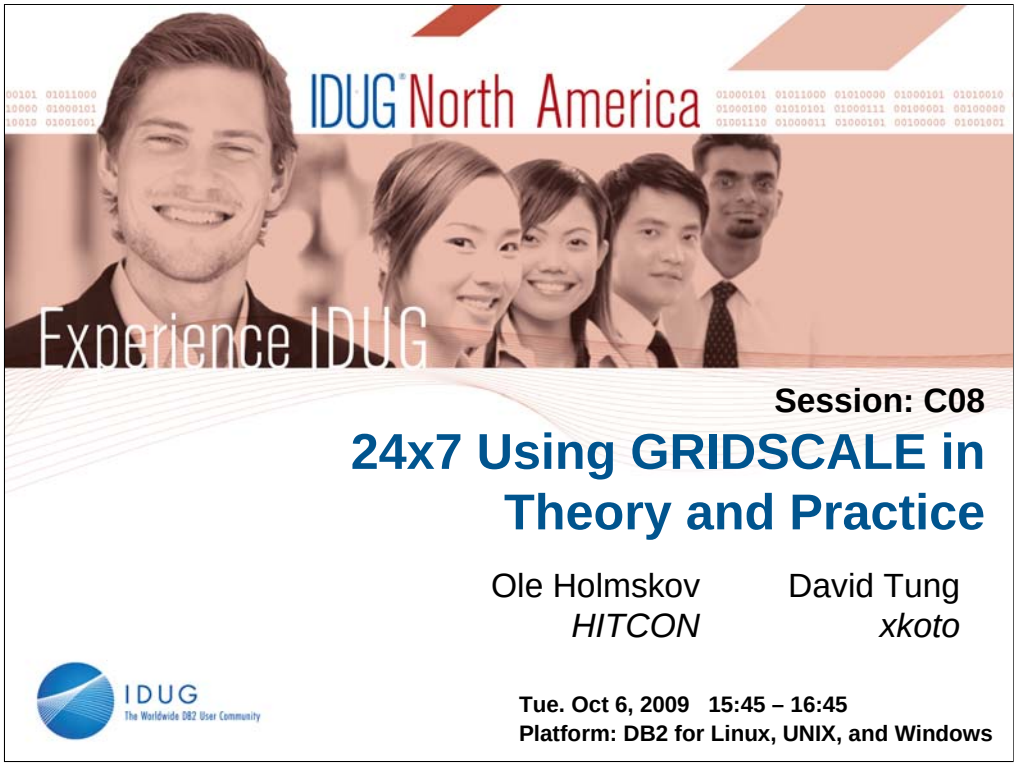


Gridscale offers a solution for scaling and protecting you from failures when using DB2 for LUW. Now - next question is - DOES IT REALLY WORK? This presentation walks through how the solution works and walks you through an actual client enablement of the solution (7000 online users running +200 TPS, 30000 SPS on 4 TB DB2 on AIX). Pitfalls, caveats, test results - all is HERE. This is the presentation where the "numbers hits the road".

Agenda

- How it works - technical walkthrough
- DSV Environment
- DSV Requirements
- Why at DSV
- Current setup
- Challenges
- Evaluation
- Status/Still to do
- Future considerations

Objective1: Gridscale walkthrough

Objective2: Client setup and objectives

Objective3: Testing scalability and failover

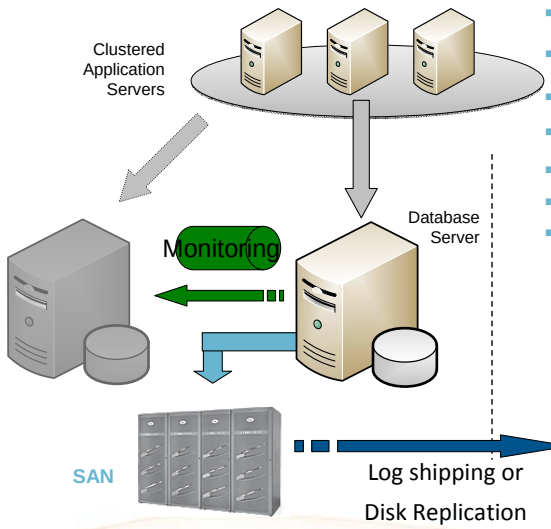
Objective4: Deployment

Objective5: Performance measurements

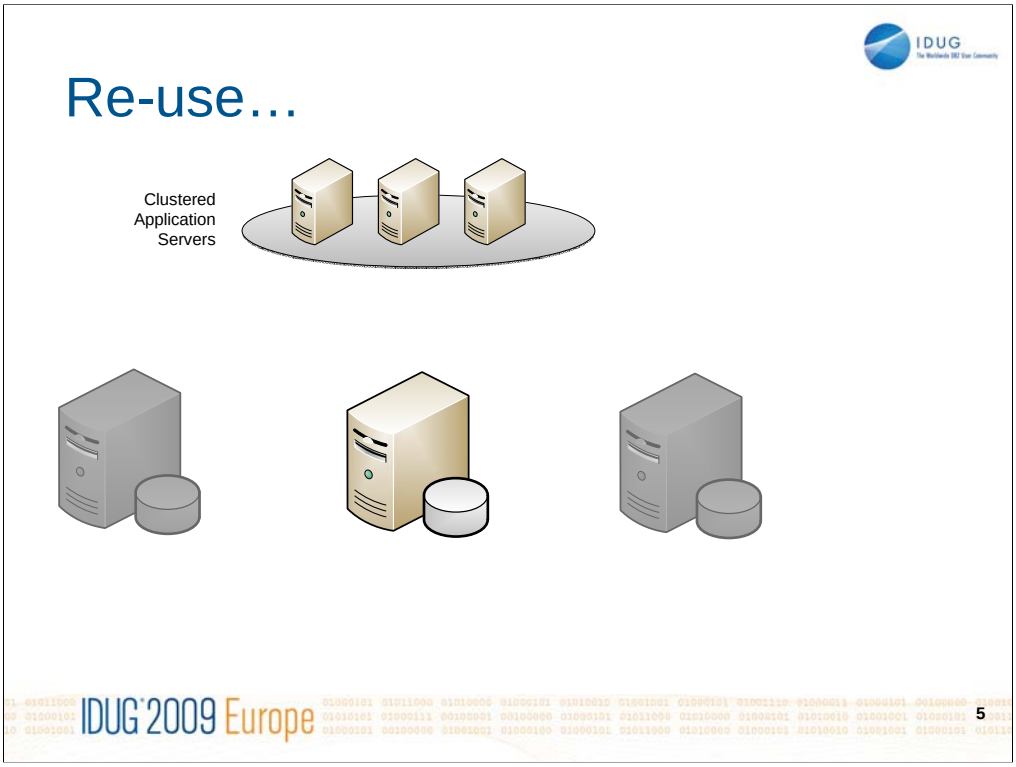
How it works - technical walkthrough

- Traditional Approaches to Availability
- GRIDSCALE Approach
- Write Processing
- Read Processing
- Database Failure Recovery Scenario
- Database Maintenance (Central versus Rolling)
- Performance

Traditional Approaches to Availability

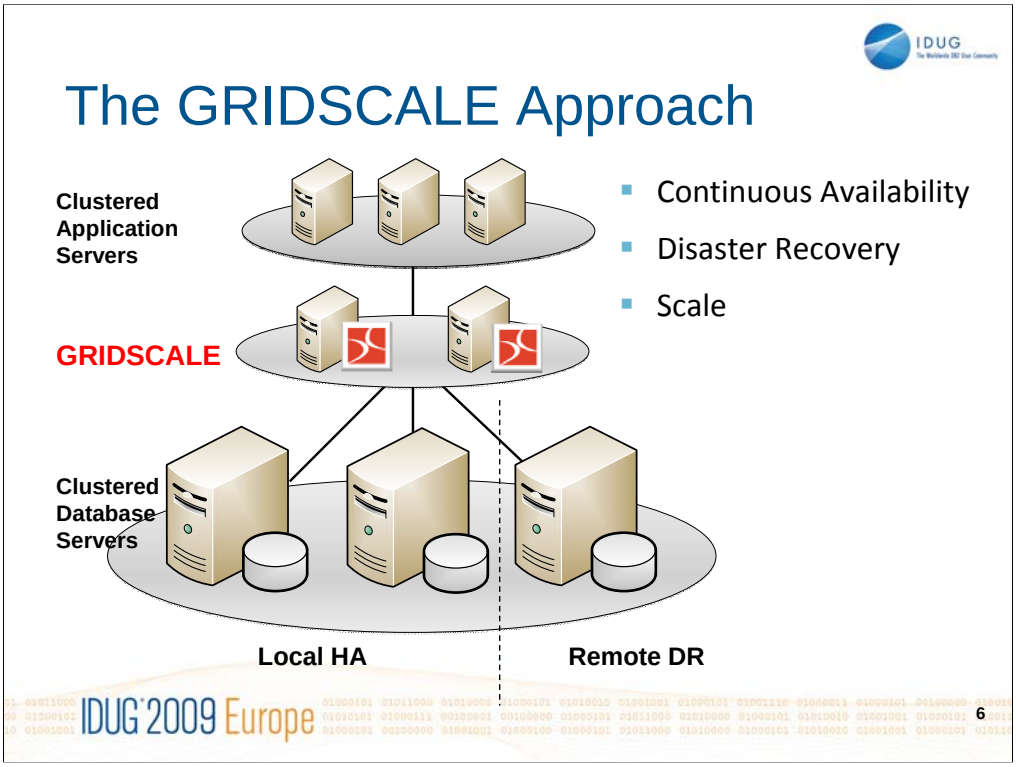


- Idle Standby second server
- Failover Monitoring
- Shared Storage
- Client Re-direct
- Second DR technology
- Second SAN
- Third idle server



GRIDSCALE can re-use the existing hardware, and light it up.

It’s more reliable because it’s simpler and more flexible, more complete solution.

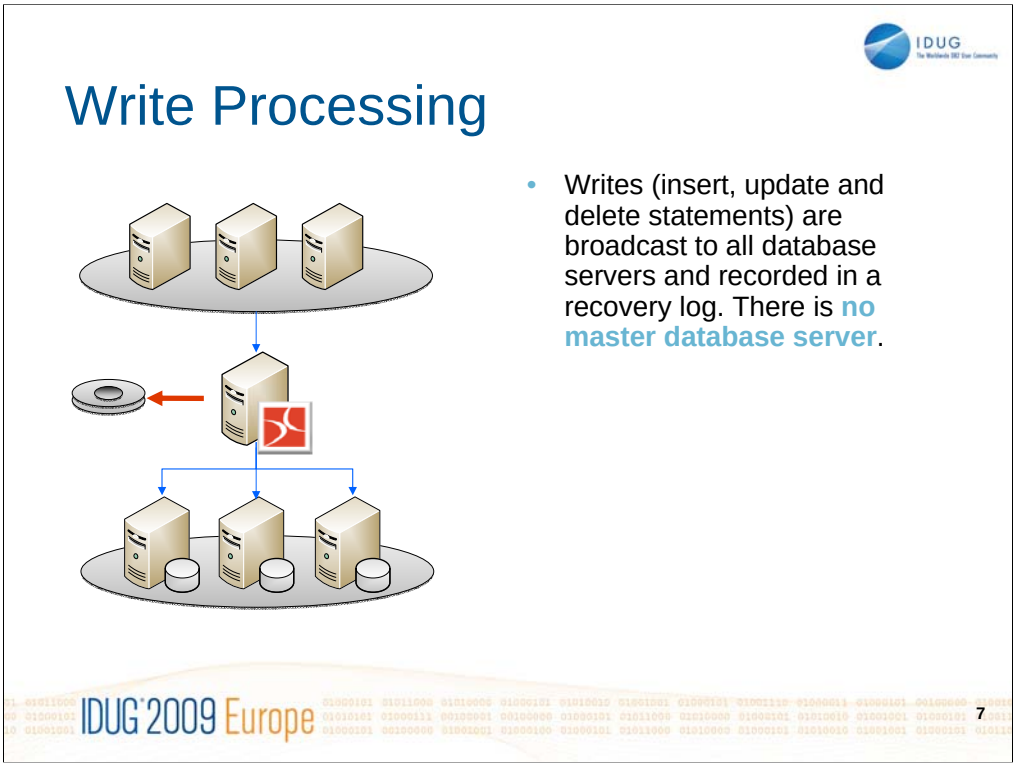


GRIDSCALE can re-use the existing hardware, and light it up.

It's more reliable because it's simpler and more flexible, more complete solution.

You can have a SAN or not, we are agnostic.

You might even have different hardware.



First response returned – not first commit

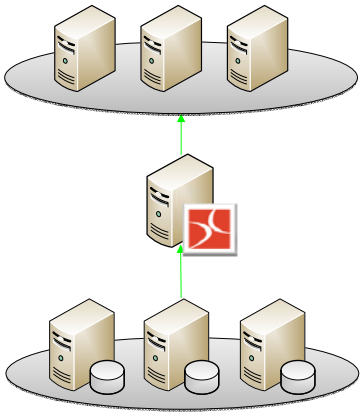
1st successful response GRIDSCALE will send back to the application so the application can continue processing. GRIDSCALE also maintains responses from the other servers. Hash of the responses, compared to the first response.

GRIDSCALE will send back a 1st success or last error so that to make sure that the cluster is resilient.

If the first response is a failure, then GRIDSCALE waits for a second response.

If a subsequent read that come is dependent on the write, it will only be routed to the SERVERS that have given a response.

Write Processing



- Writes (insert, update and delete statements) are broadcast to all database servers and recorded in a recovery log. There is **no master database server**.
- Each database server processes the statement asynchronously. The responses from the first database server to process the statement successfully is sent back to the application. Allowing the application to **continue as soon as possible**.

IDUG 2009 Europe

8

First response returned – not first commit

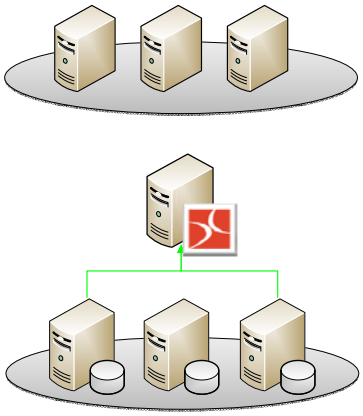
1st successful response GRIDSCALE will send back to the application so the application can continue processing. GRIDSCALE also maintains responses from the other servers. Hash of the responses, compared to the first response.

GRIDSCALE will send back a 1st success or last error so that to make sure that the cluster is resilient.

If the first response is a failure, then GRIDSCALE waits for a second response.

If a subsequent read that come is dependent on the write, it will only be routed to the SERVERS that have given a response.

Write Processing



- Writes (insert, update and delete statements) are broadcast to all database servers and recorded in a recovery log. There is **no master database server**.
- Each database server processes the statement asynchronously. The responses from the first database server to process the statement successfully is sent back to the application. Allowing the application to **continue as soon as possible**.
- The responses from the remaining database servers are compared by GRIDSCALE for **consistency**.

IDUG 2009 Europe 9

First response returned – not first commit

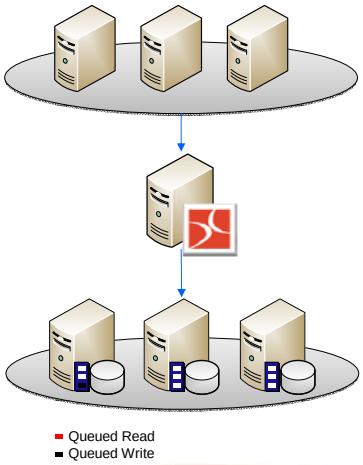
1st successful response GRIDSCALE will send back to the application so the application can continue processing. GRIDSCALE also maintains responses from the other servers. Hash of the responses, compared to the first response.

GRIDSCALE will send back a 1st success or last error so that to make sure that the cluster is resilient.

If the first response is a failure, then GRIDSCALE waits for a second response.

If a subsequent read that come is dependent on the write, it will only be routed to the SERVERS that have given a response.

Read Processing



- Reads (selects) are only sent one database server.
- Reads are **load balanced** to the least busy server that is up to date for that read.
- GRIDSCALE uses a shortest queue load balancing algorithm – this automatically takes into account differences due to servers and network latencies.

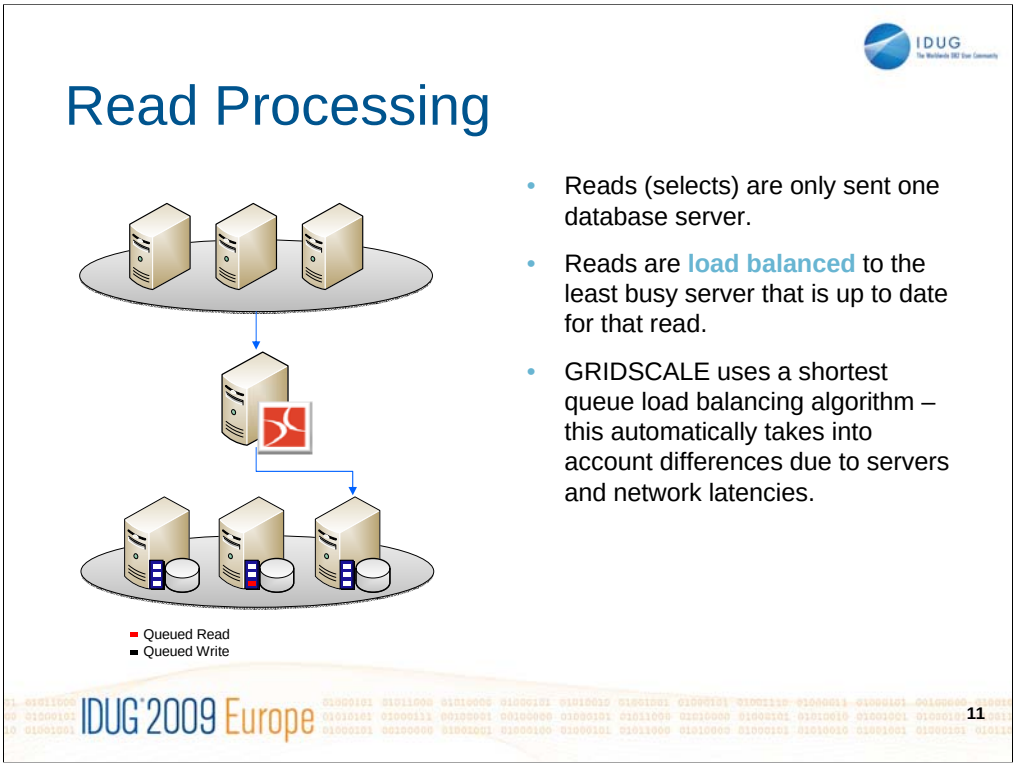
First response returned – not first commit

1st successful response GRIDSCALE will send back to the application so the application can continue processing. GRIDSCALE also maintains responses from the other servers. Hash of the responses, compared to the first response.

GRIDSCALE will send back a 1st success or last error so that to make sure that the cluster is resilient.

If the first response is a failure, then GRIDSCALE waits for a second response.

If a subsequent read that come is dependent on the write, it will only be routed to the SERVERS that have given a response.



First response returned – not first commit

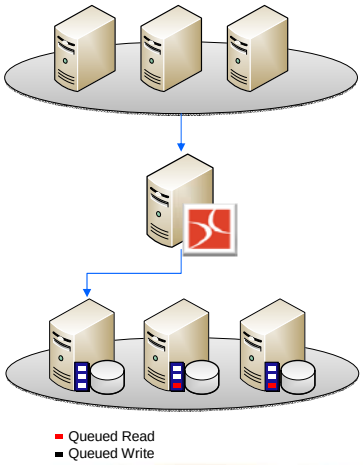
1st successful response GRIDSCALE will send back to the application so the application can continue processing. GRIDSCALE also maintains responses from the other servers. Hash of the responses, compared to the first response.

GRIDSCALE will send back a 1st success or last error so that to make sure that the cluster is resilient.

If the first response is a failure, then GRIDSCALE waits for a second response.

If a subsequent read that come is dependent on the write, it will only be routed to the SERVERS that have given a response.

Read Processing



- Reads (selects) are only sent one database server.
- Reads are **load balanced** to the least busy server that is up to date for that read.
- GRIDSCALE uses a shortest queue load balancing algorithm – this automatically takes into account differences due to servers and network latencies.

First response returned – not first commit

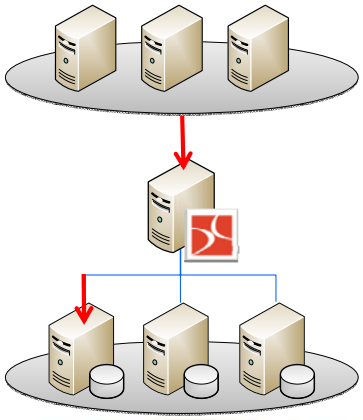
1st successful response GRIDSCALE will send back to the application so the application can continue processing. GRIDSCALE also maintains responses from the other servers. Hash of the responses, compared to the first response.

GRIDSCALE will send back a 1st success or last error so that to make sure that the cluster is resilient.

If the first response is a failure, then GRIDSCALE waits for a second response.

If a subsequent read that come is dependent on the write, it will only be routed to the SERVERS that have given a response.

Database Failure Recovery Scenario

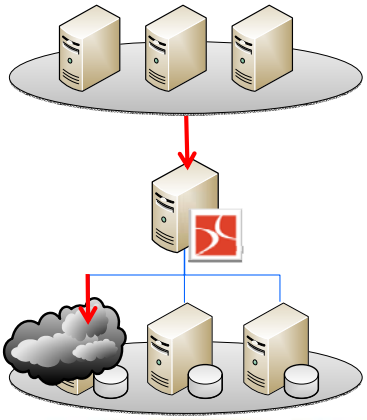


1. Application sends a READ request.

GRIDSCALE completely insulates the client from failures on the database layer.

The recovery of a failed server is automatic, and occurs whenever the database server reconnects.

Database Failure Recovery Scenario

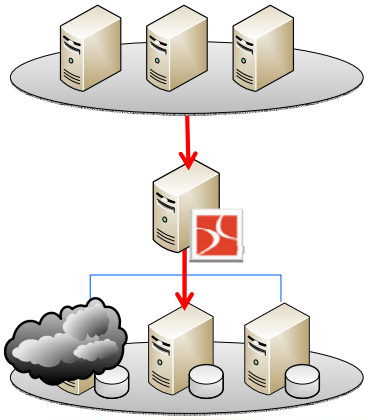


1. Application sends a READ request.
2. Database server fails after receiving a READ request.

GRIDSCALE completely insulates the client from failures on the database layer.

The recovery of a failed server is automatic, and occurs whenever the database server reconnects.

Database Failure Recovery Scenario

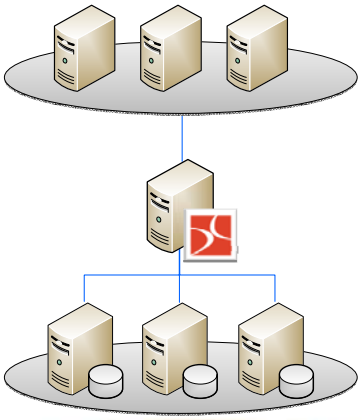


1. Application sends a READ request.
2. Database server fails after receiving a READ request.
3. GRIDSCALE redirects the read to another server.

GRIDSCALE completely insulates the client from failures on the database layer.

The recovery of a failed server is automatic, and occurs whenever the database server reconnects.

Database Failure Recovery Scenario

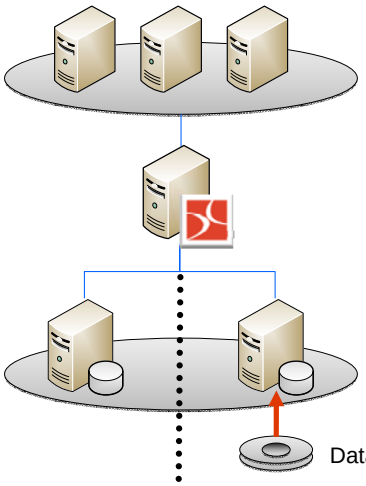


1. Application sends a READ request.
2. Database server fails after receiving a READ request.
3. GRIDSCALE redirects the read to another server.
4. When it comes back online, GRIDSCALE automatically re-syncs the failed server.

GRIDSCALE completely insulates the client from failures on the database layer.

The recovery of a failed server is automatic, and occurs whenever the database server reconnects.

Adding / Restoring a failed Database Server



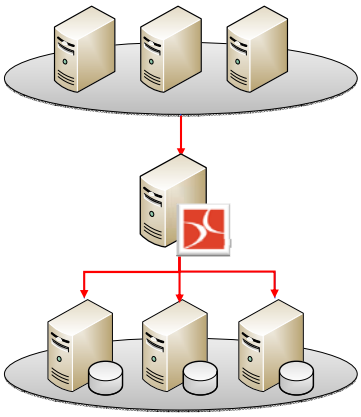
A previous night's back up can be used to add a new server or restore an existing server.

GRIDSCALE will automatically re-sync the server from the point the back up was restored to.

Adding a new server is easy, just restore the database from a backup.

There is a table inside the database that allows GRIDSCALE to locate the database server in it's recovery log.

Database Maintenance – Central

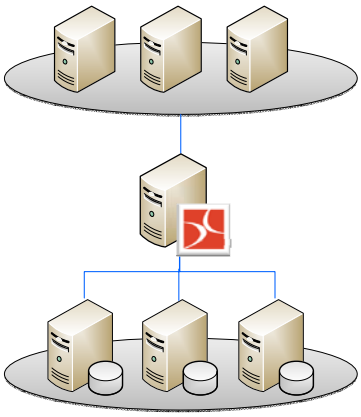


- Maintenance can be performed from the top, like any other application.
- GRIDSCALE applies the maintenance to all database servers at the same time.

You now have TWO options for Maintenance.

DDL are handled just like any other write.

Database Maintenance – Rolling



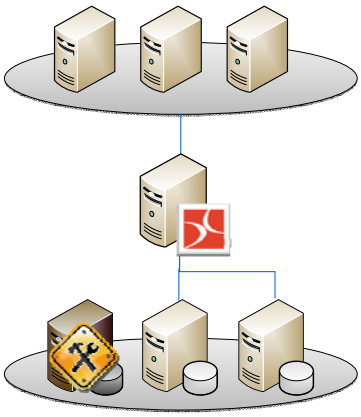
- Maintenance can also be accomplished in a rolling fashion
- Applications can remain up while maintenance task are completed.
- The maintenance can be applied to each database server one at a time.

You can always go direct, and make changes one at a time.

Anything that the application cannot see – Can be different.

You can even tune databases differently…

Database Maintenance – Rolling



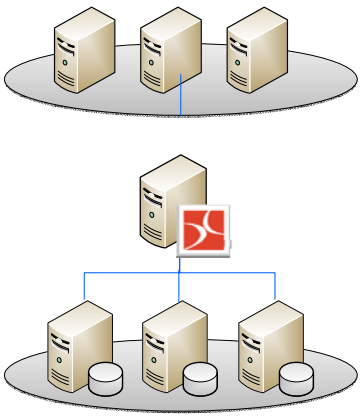
- Maintenance can also be accomplished in a rolling fashion
- Applications can remain up while maintenance task are completed.
- The maintenance can be applied to each database server one at a time.

You can always go direct, and make changes one at a time.

Anything that the application cannot see – Can be different.

You can even tune databases differently…

Database Maintenance – Rolling



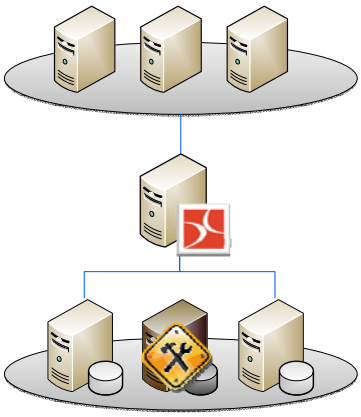
- Maintenance can also be accomplished in a rolling fashion
- Applications can remain up while maintenance task are completed.
- The maintenance can be applied to each database server one at a time.
- Mixed version is supported. Allows DBA to test-run a patch without risking the entire production cluster.

You can always go direct, and make changes one at a time.

Anything that the application cannot see – Can be different.

You can even tune databases differently…

Database Maintenance – Rolling



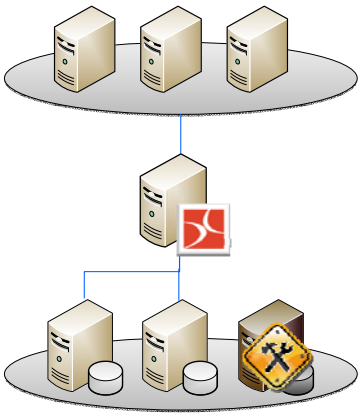
- Maintenance can also be accomplished in a rolling fashion
- Applications can remain up while maintenance task are completed.
- The maintenance can be applied to each database server one at a time.
- Mixed version is supported. Allows DBA to test-run a patch without risking the entire production cluster.

You can always go direct, and make changes one at a time.

Anything that the application cannot see – Can be different.

You can even tune databases differently…

Database Maintenance – Rolling



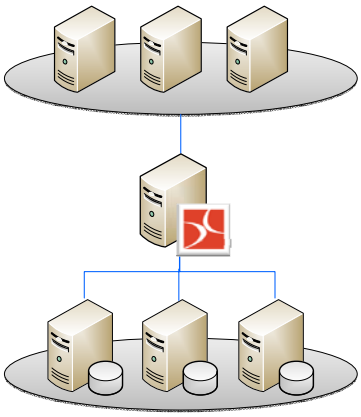
- Maintenance can also be accomplished in a rolling fashion
- Applications can remain up while maintenance task are completed.
- The maintenance can be applied to each database server one at a time.
- Mixed version is supported. Allows DBA to test-run a patch without risking the entire production cluster.

You can always go direct, and make changes one at a time.

Anything that the application cannot see – Can be different.

You can even tune databases differently…

Database Maintenance – Rolling



- Maintenance can also be accomplished in a rolling fashion
- Applications can remain up while maintenance task are completed.
- The maintenance can be applied to each database server one at a time.
- Mixed version is supported. Allows DBA to test-run a patch without risking the entire production cluster.

You can always go direct, and make changes one at a time.

Anything that the application cannot see – Can be different.

You can even tune databases differently…

GRIDSCALE's Proven Performance

GRIDSCALE Load Performance Test

- GRIDSCALE delivers **near linear performance** for every commodity database node that is added to the cluster

E-Commerce Benchmark (89% Scale)

Transaction-intensive e-commerce benchmark run at IBM's Linux Integration Center lab in Austin, TX. A simulated user is selecting items and utilizing an on-line shopping cart. 80% read-only queries and 20% read/write transactions.

DVD Store (DS2) Benchmark (80% Scale)

E-commerce benchmark developed by Dell, performed by xkoto and IBM. It models an online DVD storefront supporting 50,000,000 customers, 1,000,000 products stocked in inventory in varying amounts and a history of approximately 12,000,000 purchases.

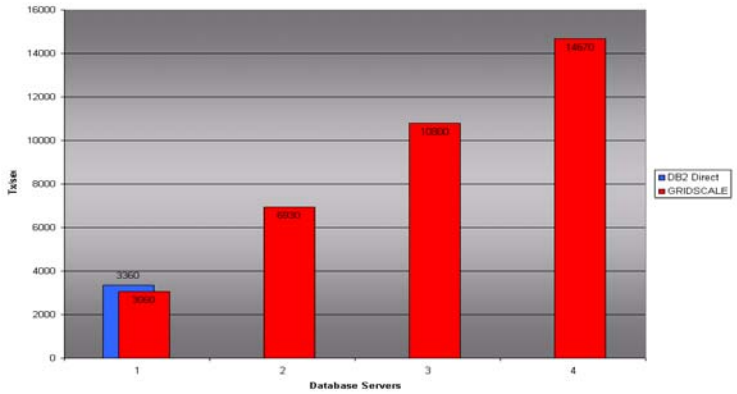
IBM's internal TPW style benchmark in Austin DB2 performance lab.

Scalablity follows the Read/Write mix. 80% read/write will mean about 80% scale.

100% Read Mix - Tx/sec on Typical Hardware

GRIDSCALE Server
- 2 CPU, 2 Core Intel 3.0 Ghz
- 99 MB used/2 GB RAM
- 14670 tx/sec Peak

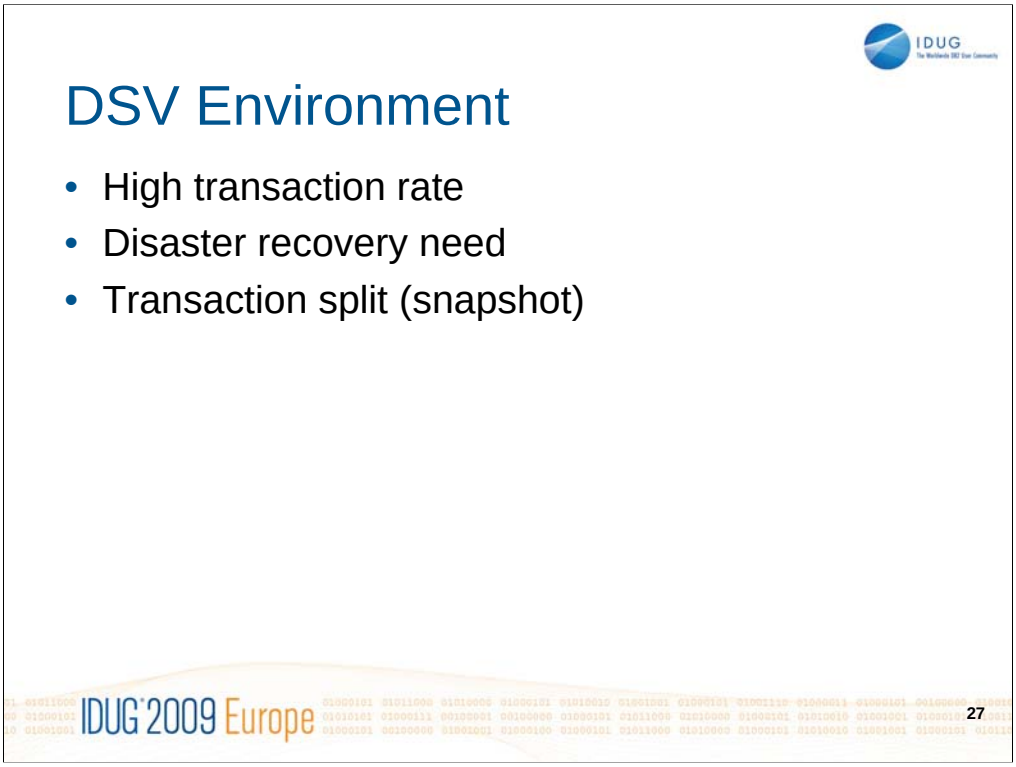
Database Server
- 4 CPU – 2 Core IBM 3950
- Database in Memory



Performance will vary based on application work load.

This is from a GRIDSCALE customer.

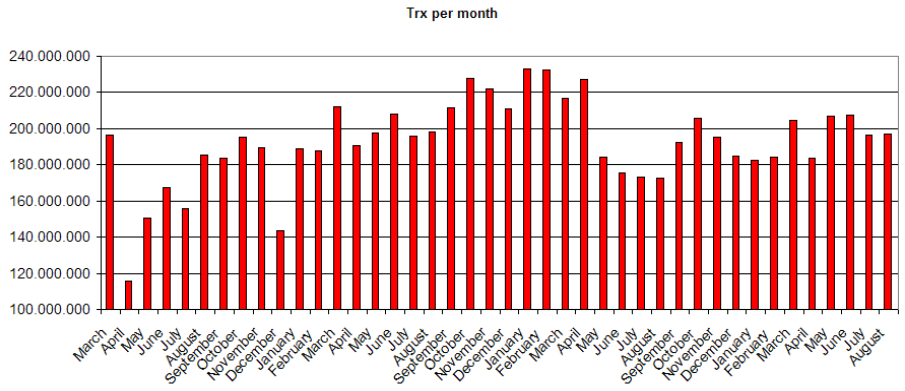
15K statements per second!



A transaction is a committed or a rolled back transaction.

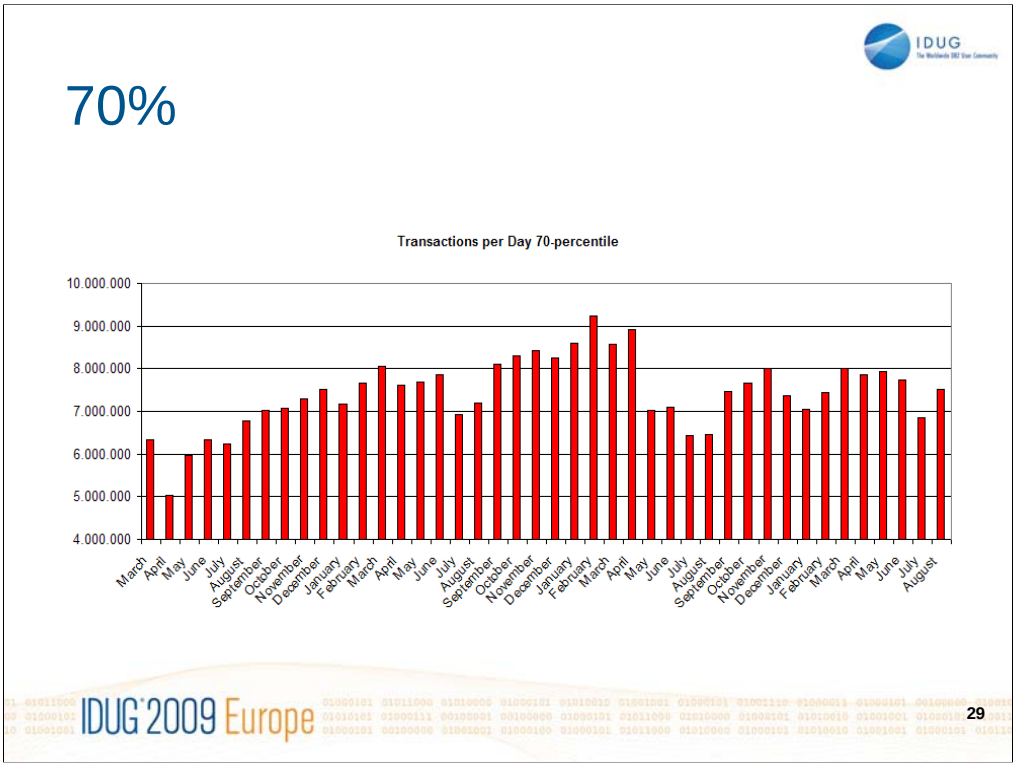
We are peaking out at 30-40K stmts per second. App. 250-300 trx/seconds.

Transactions per Month



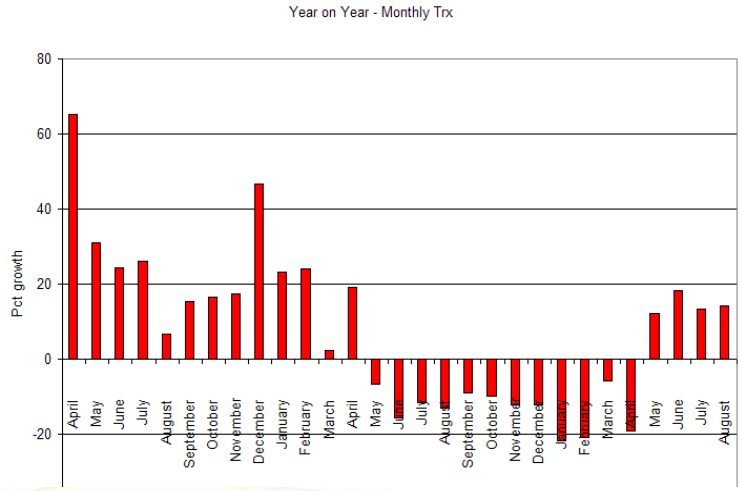
We do monitor the number of transactions per month. Simple – but solid measurement.

We had to start somewhere.



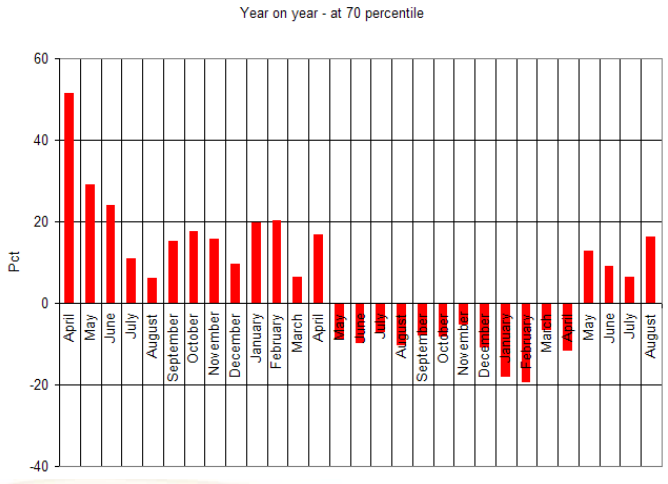
At the 70 percentile. Why 70%? Because we filter out monthends (peak days) and weekends gets filtered as well.

YearOnYear - Monthly



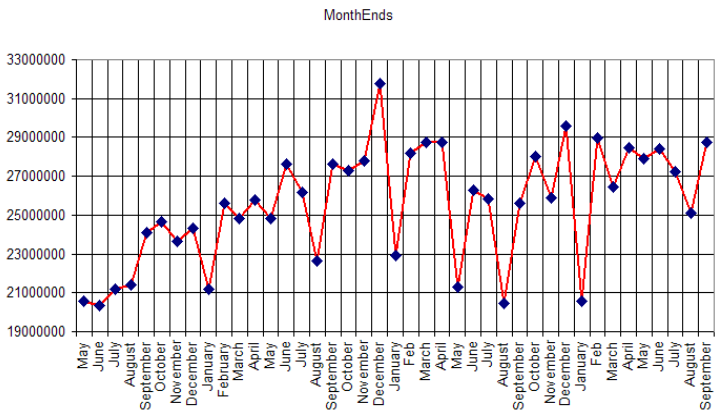
The year on year – growth is back.

YearOnYear -70%



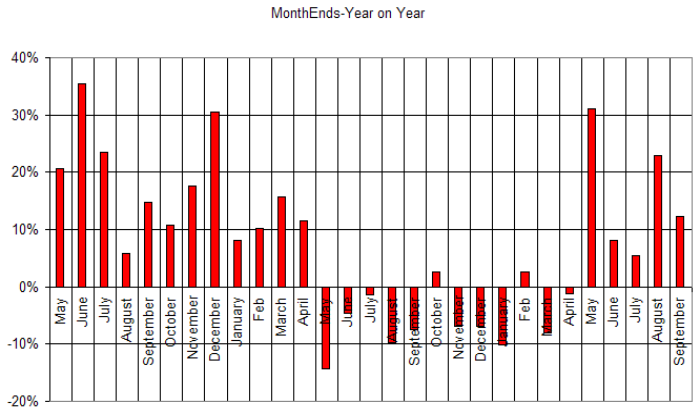
Same for year-on-year

MonthEnds



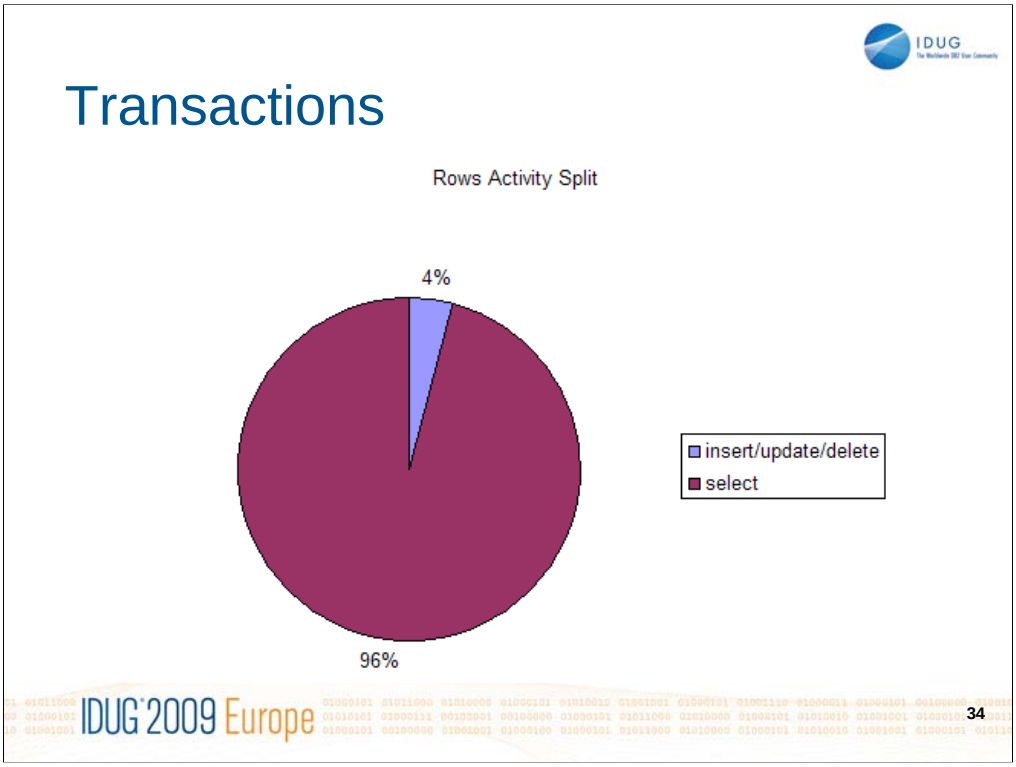
The real workload is monthends. Invoicing, etc.

MonthEnds - YearOnYear

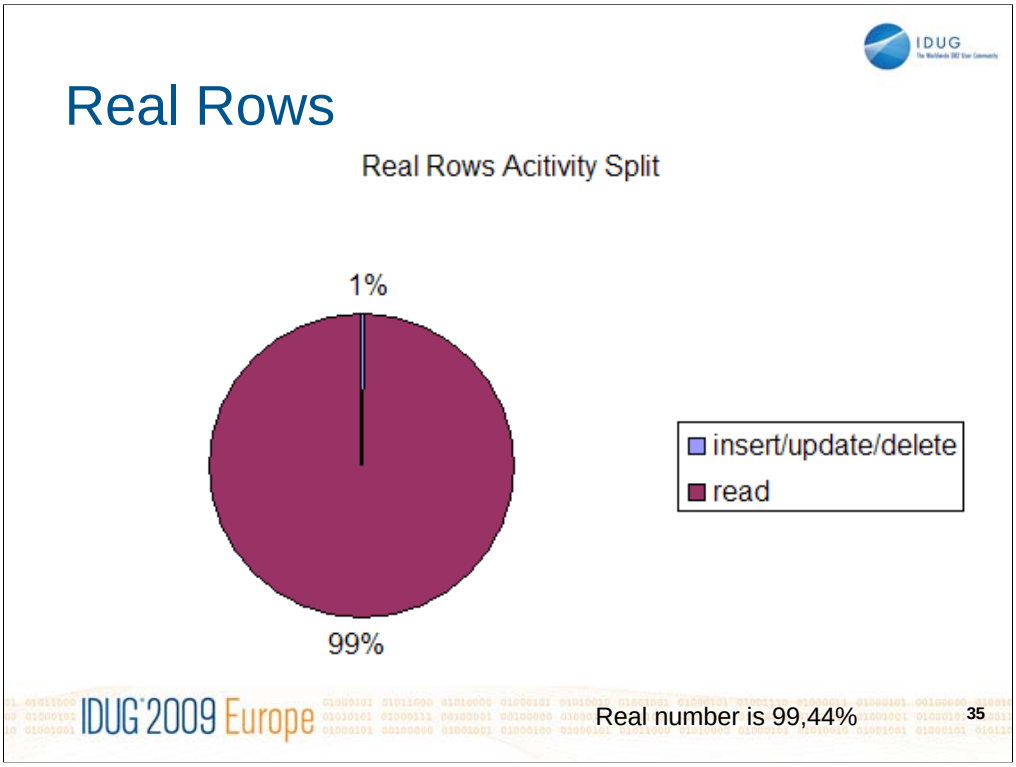


Growing… again.

There has been some concerns about the financial situation of ”the World”.



This is WHY we are a good fit with gridscale. Remember that reads get decreased by the number of servers you deploy?



And if we are talking real row activity?... Even better…

DSV Requirements

- Disaster/Recovery
- Scalability
- Upgrade
- Failover
- Maintenance (reorg/runstat/etc)

Challenges

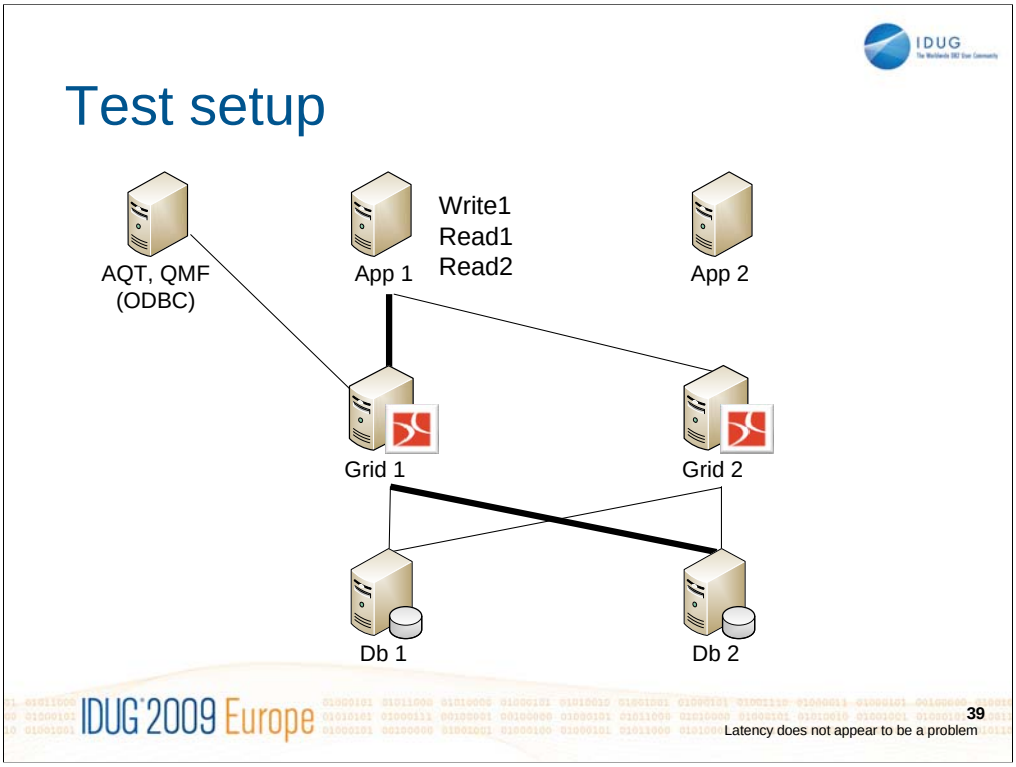
- Running CICS
- Using XA

- NO recompile of programs
- NO relinking of programs
  - 2009 individual packages
  - 66219 in total across schemas
  - 3.298.436 entries in sysibm.syssection
  - 30049 tables
  - 3,7 TB
- ”Seamless” integration
- Basically – it should be fully virtualized

This were the challenges that we were facing when starting out. Basically – we should just replace the driver….

Current Setup

- Test setup
- Versions
- GRIDSCALE II



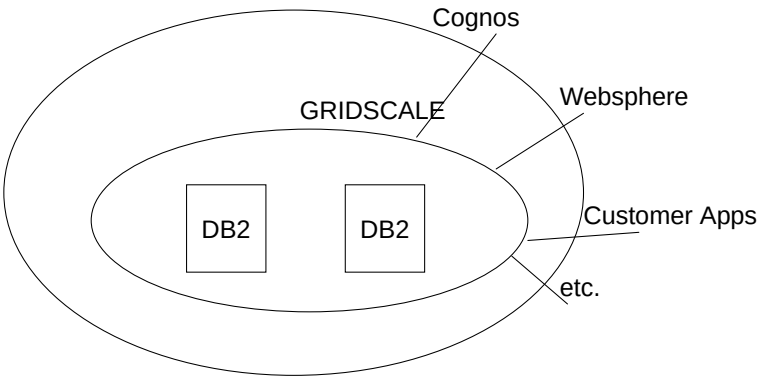
Two application servers

Two GRIDSCALE servers

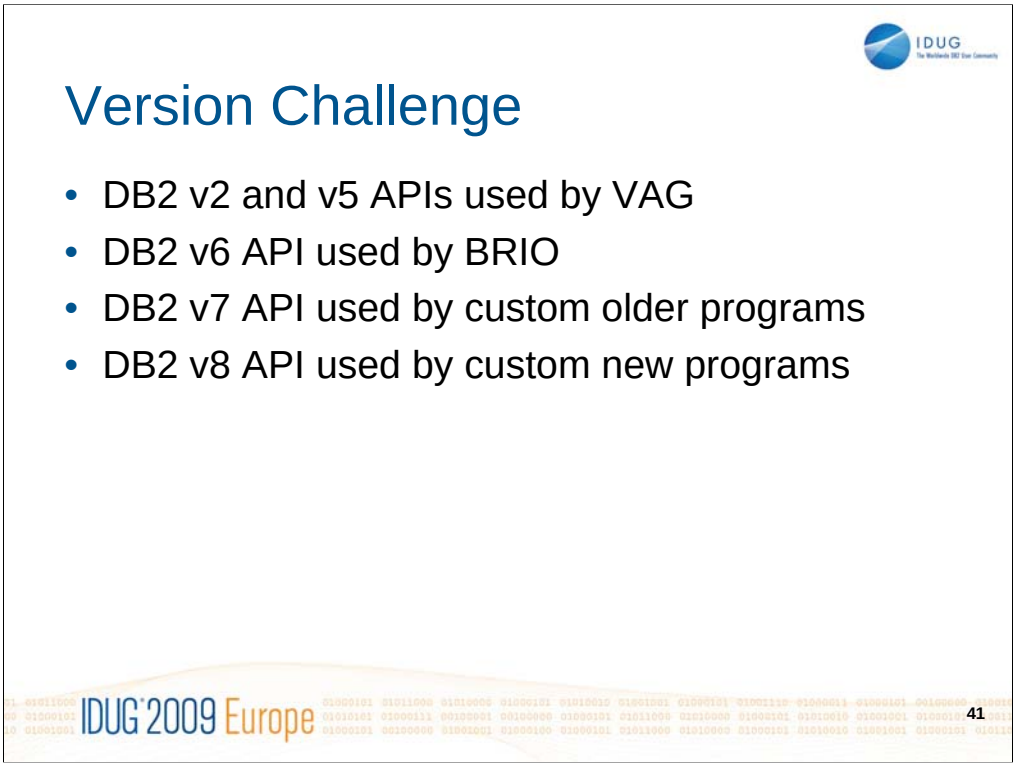
Two Database Servers

And one client on the side….

GRIDSCALE

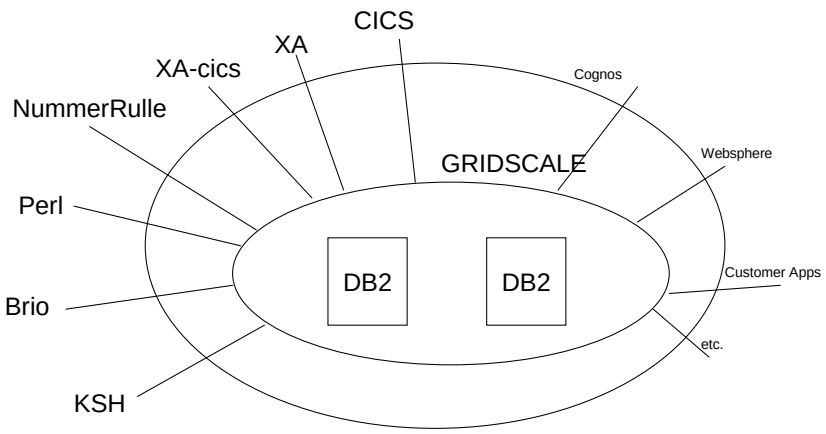


This is my normal view of GRIDSCALE



Well – the ”we support V8 and V9” statement – was not really good enough.

GRIDSCALE II

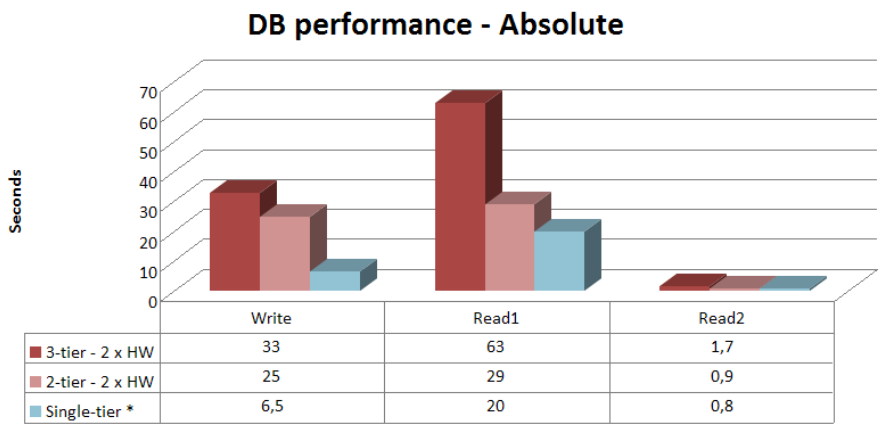


We wound up with something looking like…

Evaluation

- Initial test
- Estimation
- Further Impact Analysis
- Results

Initial Tests



	Write	Read1	Read2
3-tier - 2 x HW	33	63	1,7
2-tier - 2 x HW	25	29	0,9
Single-tier *	6,5	20	0,8

xkoto tuning 1,7-app. 1,3 *

We sort of mimic the workload – using a test program.

Write – is a simple ”write 10.000 rows to a table”

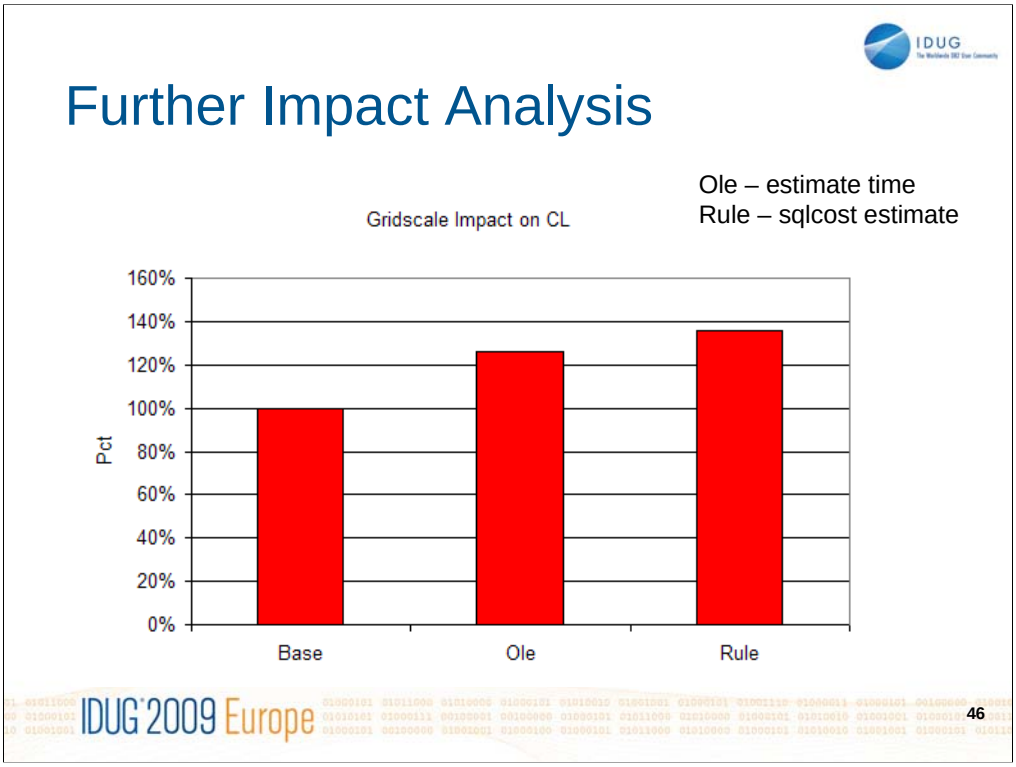
Read1 – is read 1 row 10.000 times.

Read2 – is read 10000 rows once.

Estimation

sqlcost	Access	Relative performance
50	index access only	200
75	index access with rid lookup	45
125	Other	30

Went back – did an analysis of the Read2 case.



If I had index access ONLY – there is “double the response time”. For a index access, RID lookup – we are talking 45% increase. For the “normal query” we are talking 30% increase.

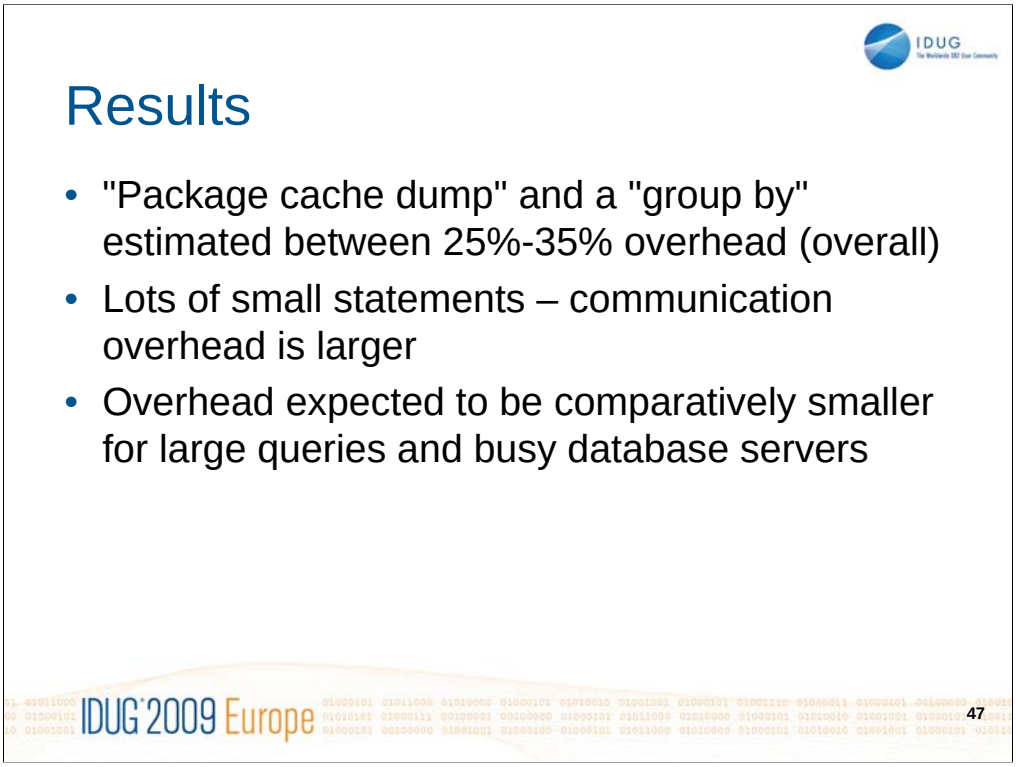
So – I decided to dump the packagecache, group by the various sections, packages and do a “bit of simple math”. I then went through EACH of the 50 statements, and explained these – retrieved the sqlcost and analyzed the access path.

I JUST looked at the 50% (49% to be accurate) of the workload in CL (ranked by the sql statements used the most across all schemas – ie. DB2 performs equally across schemas). That is ranked in Appendix B. That is 50% of all statements in the package cache based on sqlcost. Just for the reference: There is a total of >405000 entries all together….

I then to EACH of these and explained them.

In general there has been 309 million references to the 50 sections. That covers app 49% of all entries. I then roughly estimated by looking at the access path – what the impact would be. I came up with a factor of 1,2585 – roughly 25%. When just applying the measured impact and the sqlcosts – that is some 1,3558 – roughly 35%.

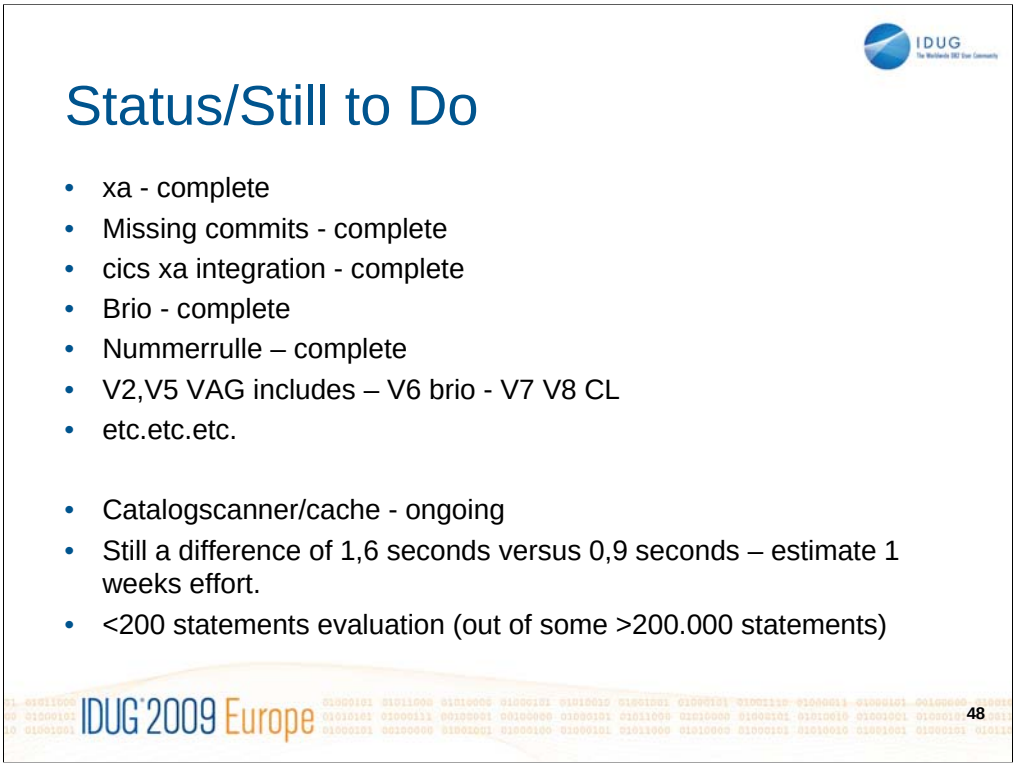
I was thinking A LOT about the method: Actually – the sqlcost is not really what is uplifted. But it is the execution time. Or rather – the experienced execution time measured on the app server (the execution time might actually get shorter under load on the database server). But basically I use the sqlcost for categorization and determining the request types, and then just estimates the impact upon deployment of GRIDSCALE.



The conclusions so far.

Please NOTICE that an application using tons of small statements is the worst-case scenario for scaling:

If the UOW increases, the GRIDSCALE and network overhead diminishes.



Still items to close.

1) Due to the size of the catalog (due to the number of packages) – there was some product issues around the GRIDSCALE catalogscanner.

2) The 1.6 seconds could be brought down to 1.3 seconds. However, that had a dramatic impact on the Read1 testcase. But there are parameters available in GRIDSCALE to tune performance.

3) There are app. 200 statements that has been identified during the GRIDSCALE preliminary evaluation phase, that should be analyzed. Most of these are expected to have NO impact.

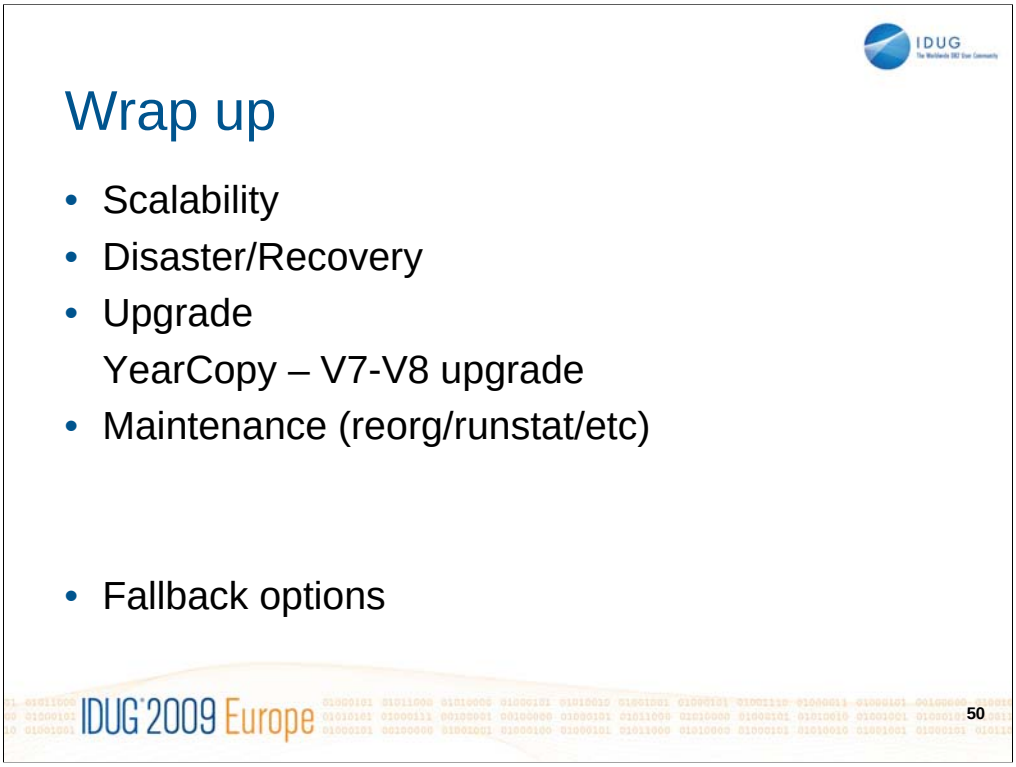
Future Considerations

DB2 V9.7

- Package cache (525E3 -> 2E9) (450K)
- Threaded model
- WLM
- Scan sharing
- Logfiles
- Currently committed
- XML
- Range partitioning (local indexes)

P6 pricing/performance

- PVU +20%



This is what we were looking for.

There is a small story about the YearCopy program: A program that crashed the V8 FP 12 upgrade we did.

How often do you run this (once a year?) ☺ It brought the instance down!

3 weeks before going into production we had to go back and retest V8 FP14.

I would like to have something like GRIDSCALE in place to help perform these critical upgrades!

Breaking the logchain/altering the logfile format is not REALLY an option when doing these upgrades.

Session C08

24x7 Using GRIDSCALE In Theory and Practice

Ole Holmskov
HITCON
ole.holmskov@hitcon.biz

David Tung
Xkoto
david.tung@xkoto.com