Scaling CMS data transfer system for LHC start-up

Lassi A. Tuura

Northeastern University

With B. Bockelman, D. Bonacorsi, R. Egeland, D. Feichtinger, S. Metson and J. Rehn on behalf of the CMS experiment

CHEP 2007

Victoria, BC, Canada

2-7 September 2007

Cheat sheet for data transfers math

12 MB/s	≈	1 TB/day	≈	500 files/day	≈	30 TB/month
1.2 GB/s	≈	100 TB/day	≈	50’000 files/day	≈	3 PB/month

Average CMS file size is likely to be ~2 GB

Related content at this conference

#352—The CMS data and workflow management system

#369—CMS experiences with computing, software and analysis challenges

#280—The CMS LoadTest 2007: an infrastructure to exercise CMS transfer routes among WLCG Tiers

#277—CMS CSA06 experience at INFN

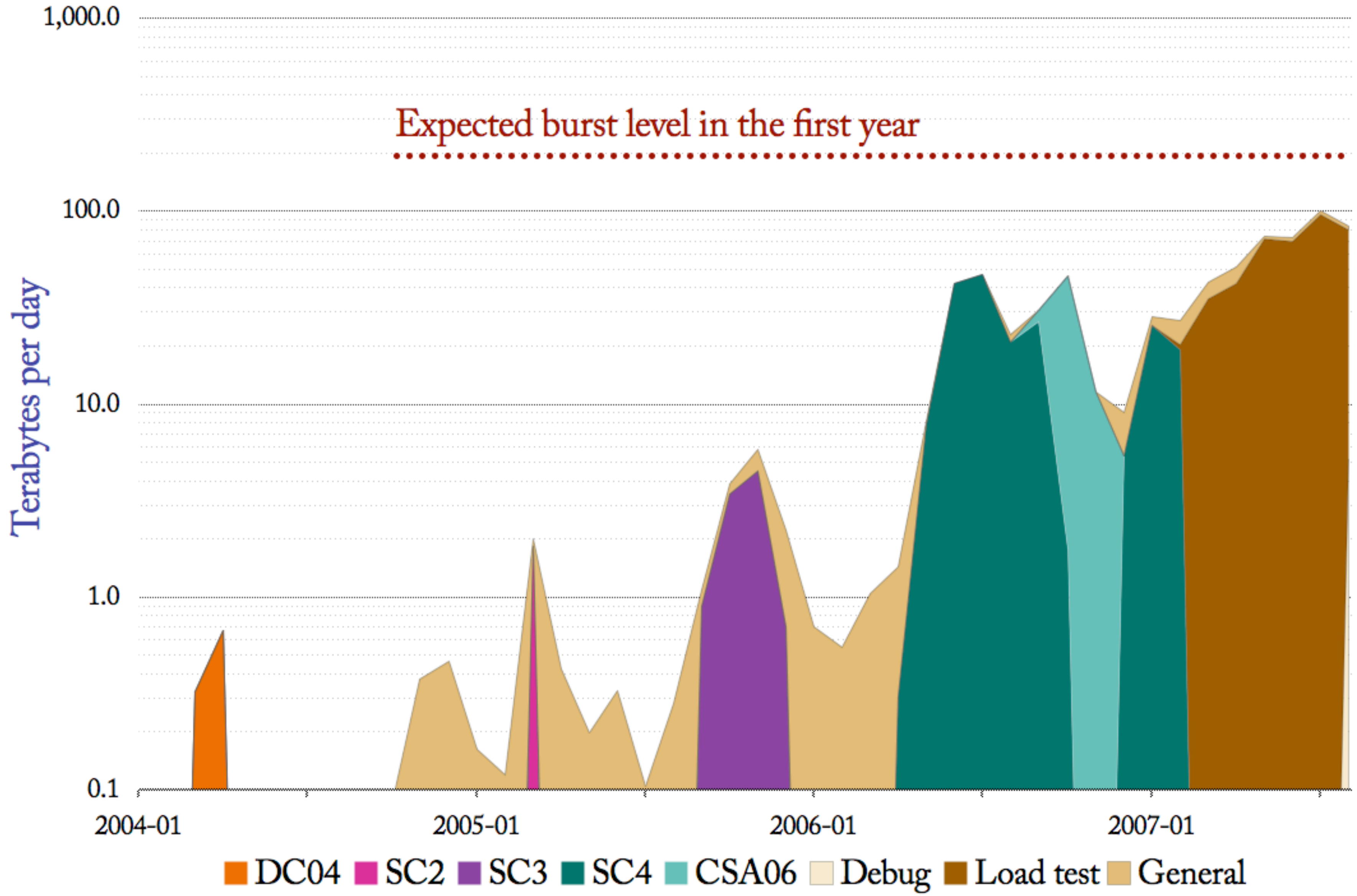
#281—Computing operations at CMS facilities

#288—Exercising CMS dataflows and workflows in computing challenges at the Spanish Tier-1 and Tier-2 sites

#325—The CMS dataset bookkeeping service

CMS PhEDEx data transfers

Total data volume month by month



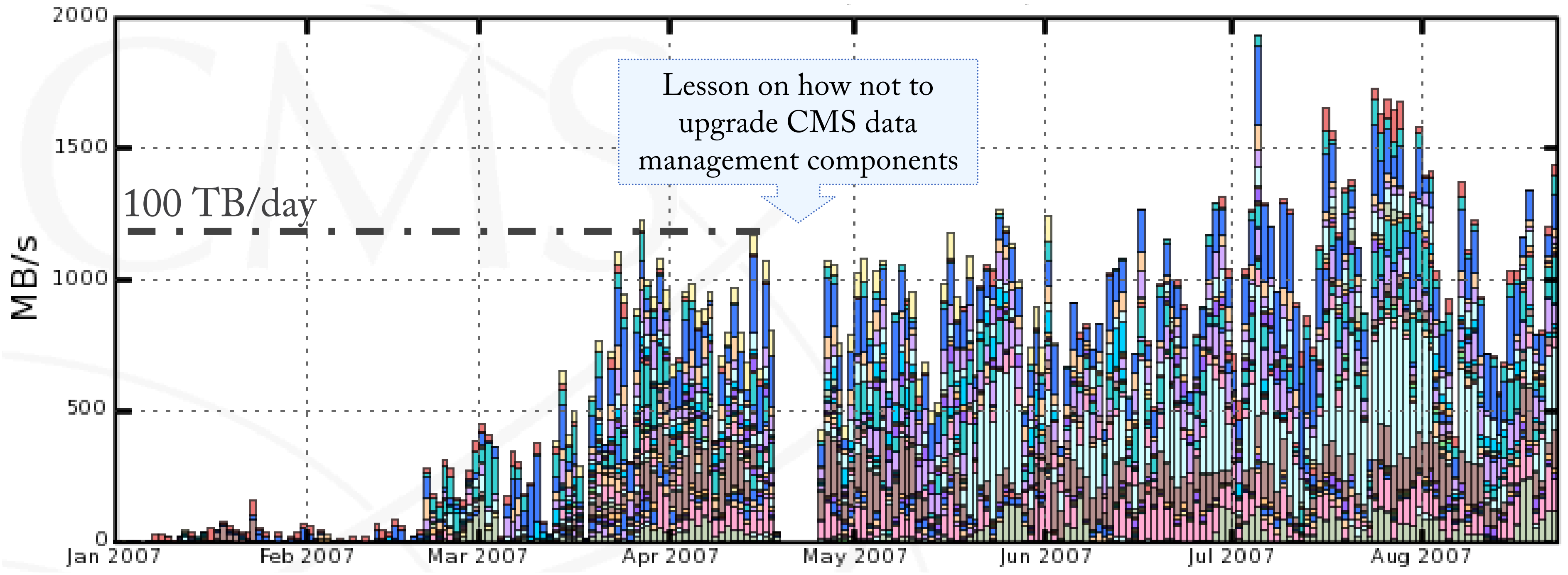
CMS has exercised the data placement and transfer system PhEDEx continuously since 2004 in preparation for LHC start-up. With one year to start-up, we approach the expected “real” transfers in scale, but not yet the full complexity.

We currently transfer at **1.2 GB/s ≈ 100 TB/day** global average rate. Next year the we expect the busiest days to reach a global aggregate of **200 TB ≈ 100’000 files a day** between CERN and the Tier-1 and Tier-2 centres. In addition come transfers to all the 170 institutes and some 3’000 personal computers.

At present **7 Tier-1s, 49 Tier-2s and 12 Tier-3s** are involved, representing ~40% of the institutes involved in CMS.

CMS PhEDEx data transfers

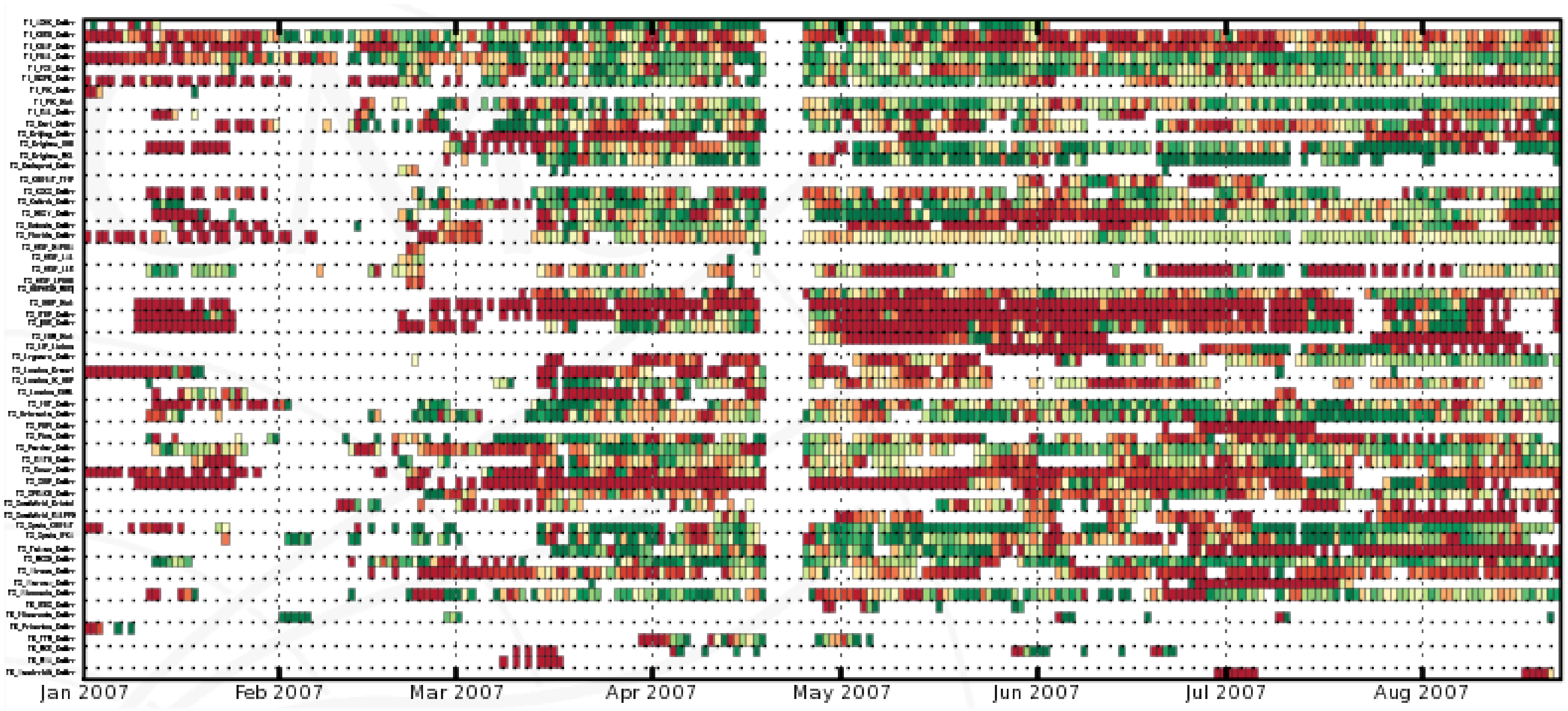
Year 2007 day by day average rate



We operate the data transfer system as if the experiment was already running. We have had two service outages exceeding 24 hours in the last two years. Another CMS data management component caused both by initiating a poorly rehearsed upgrade.

CMS PhEDEx data transfers

Year 2007 day by day average quality



The main sources for our progress have been a robust, scalable and lock-free data transfer system combined with a remarkable manpower investment to commission the data transfer service. This presentation covers the former. The plot above, showing the ratio of successful file transfer attempts, illustrates why the latter has been, and continues to be, indispensable.

➻ #280 ➻ #369

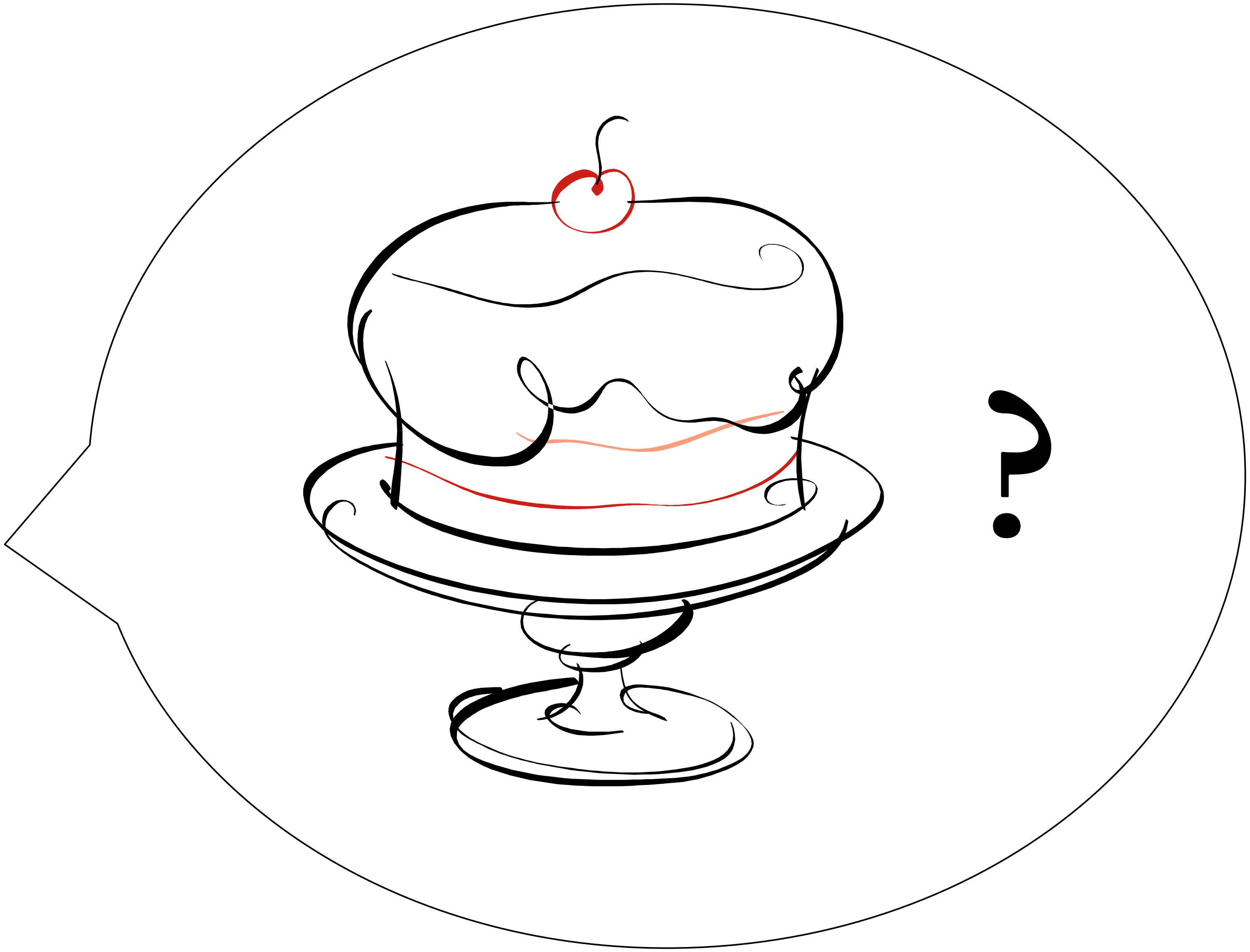
So how do I make a distributed data transfer system scalable and robust?

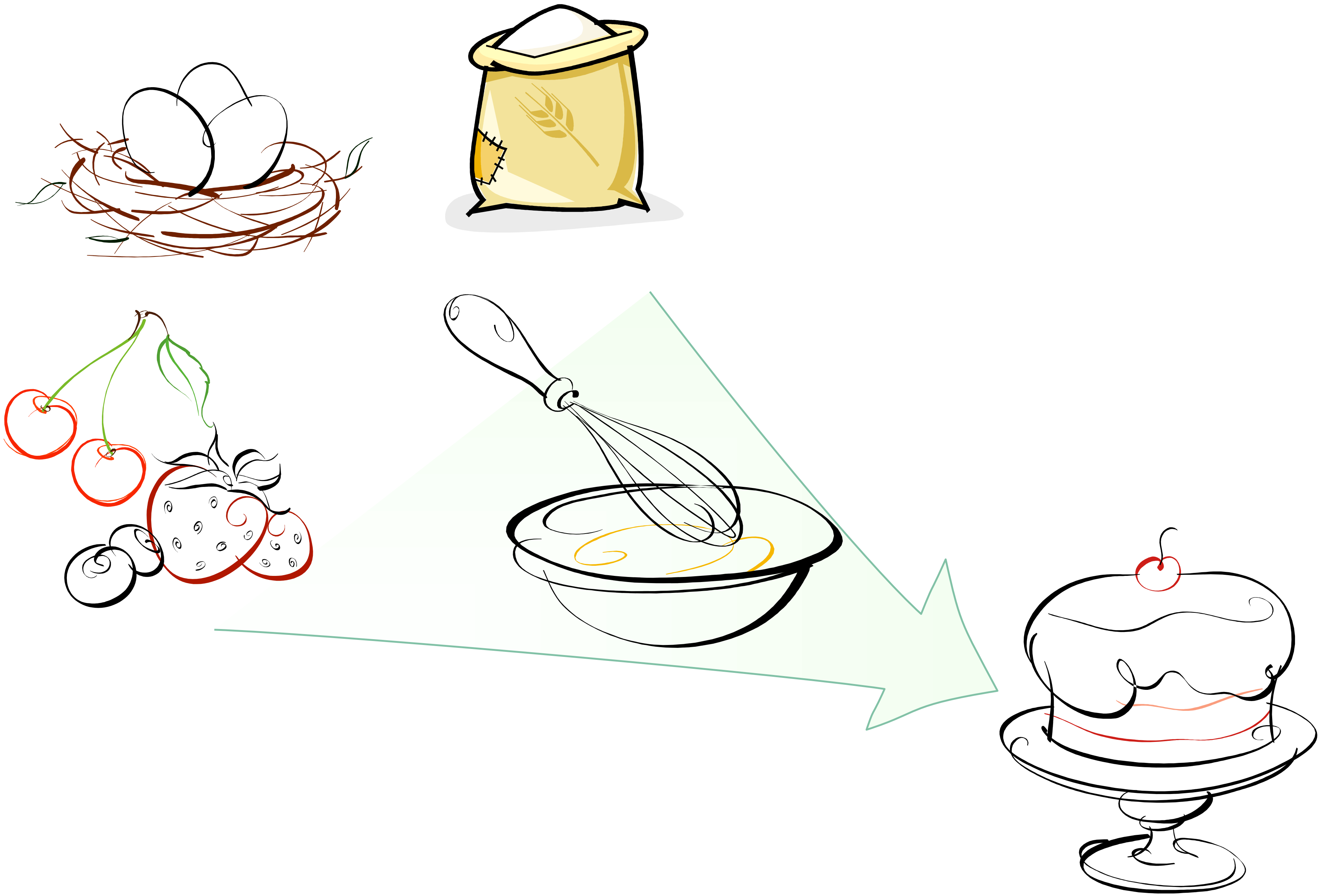
Solution A

Hire a group of clever developers.
$3M later you have a well-working system.

Solution B

The rest of this talk might save you some of the $3M…





Key ingredients used in CMS

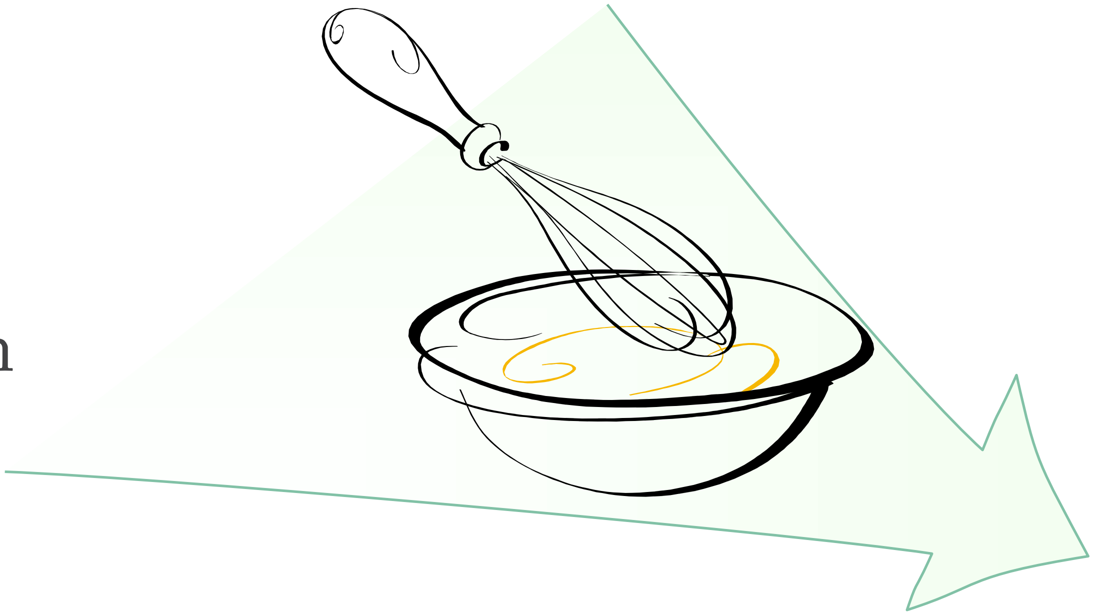
Essential factors for scalable distributed data transfer system

Technical architecture

System architecture

Database engineering

Validation processes



System architecture determines the performance envelope. It is critical to address the core scalability and reliability concerns at the architecture level. The top three most important transfer related issues resolved by CMS are listed below; please refer to the conference paper for further detail and other issues.

Restricting the problem scale. The data transfer system only tracks data actually in transfer. We divide datasets into 5-10 TB file blocks to reduce the complexity of data placement decisions by a factor of ~1'000.

Keeping local information local to the site. Transfers are operated at each participating site, giving control where it belongs and improving operations and support efficiency. CMS uses only *trivial file catalogues;* the only on-site service contacted by transfers and jobs is the storage system, and we worry about consistency with one less database.

Isolating problems. Because data transfers are critical and their operation is spectacularly labor-intensive, we aggressively protect working transfers from failure regions and actively prevent errors from slowing down the system as a whole. Problem *diagnosis and remediation* is left to humans as grid middleware rather thoroughly obfuscates what actually happened.

The **technical architecture** choices depend more on personal experience. It is entirely reasonable for others to derive a sound design by making other choices. We feel the following were important.

High-availability well-tuned central "blackboard" database at CERN. Our transfer system is agent-based. The agents communicate solely via the database. We have been very pleased with the service provided by CERN.

Level-triggered asynchronous state manipulation. We use no direct agent-to-agent communication. Hundreds of concurrently running agents store their current state in the database, effectively objects in different stages of state machine execution. Database is updated only when a task is completed successfully. Work is exclusively defined by the difference of desired and current state of the objects – we never queue task messages or send back responses. This is error-resilient, immune to losing work requests, and most importantly, lends itself to autonomous and self-healing computing. However the (big) challenge is to make state and operations on it cheap enough at the scales we need.

Defensive error handling. (Apparently) unlike most grid services, we assume errors happen all the time, everywhere. Our fault handling strategy significantly increases the experiment's effectiveness. We squelch unnecessary noise, swallow uninteresting transient failures and retry later, let error-prone systems cool off by backing off, hibernate backlog to curb system load, pace backlog handling to avoid recovery surges, and probe system stability before committing new work after sustained failures.

Hierarchical monitoring. We find effective transfer operation depends on the availability of extensive monitoring data on both current and historical performance conditions, applied to various user groups. A sizeable fraction of PhEDEx is dedicated to a hierarchical monitoring system and a web site providing a view of everything going on. *The key to making monitoring data harvesting and access cheap was the division of the system into monitoring zones.* In each zone a "well informed" agent captures a state snapshot at a time it knows to be economical. The snapshots propagate from the "hot" zones up a monitoring hierarchy so the web site and system internal decision making can make use of it. A variable-resolution time series summarises the data for frequent use. Time series access is optimised, it is as reasonable to query performance data spanning years as it is for the last 24 hours.

Database engineering begins when you begin translating your sound design into an actual product. Here are a few techniques that really helped us along. A few more can be found in the paper.

Do do the basics. Use bind variables and big enough row arrays for query fetches and uploads; remove all but required indices until measurements show one is needed; use the database for work – don't implement the database engine in your application; commit at intervals most economical for I/O including redo. Schema design stops at entity modeling only in classes; include physical storage and data organisation parameters in yours. Evaluate the query execution engines in your database, and make sure they use the right parametres.

Measure, analyse and understand. Ignorance is a poor base for a design. It is also inexcusable. Use the existing tools to *find out what is going on* and adapt your design accordingly. Excellent Oracle resources: Tom Kyte's books, Automatic Workload Repository reports, Enterprise Manager.

Address locking, row contention and cluster cache coherence traffic. *Table and index partitioning* helps reduce number of rows touched by a query. This can help with both lock contention and cluster coherence traffic as different clients will be pushed to access physically distinct storage. *We also define a data ownership model* which defines for every row in every table at all times exactly which agent owns that row. Only the row owner is allowed to update the data. In our "state machine" schema this was easily done and reduced row lock contention and cache coherence traffic noticeably.

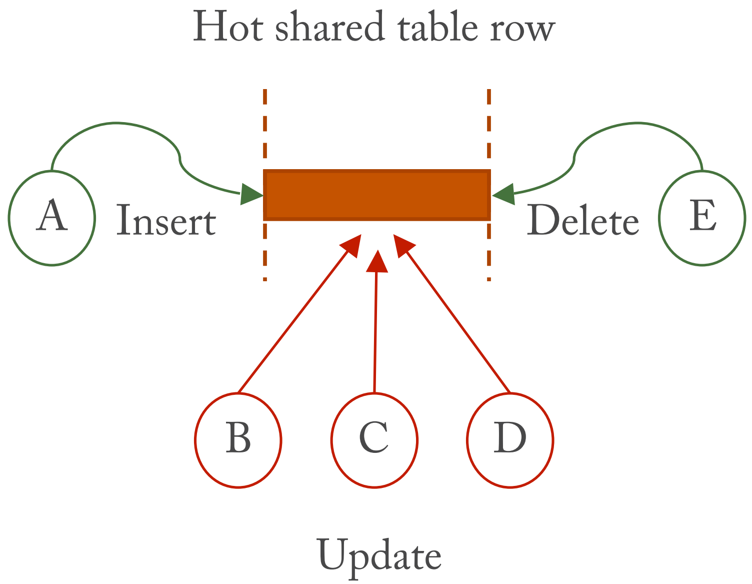
An unusual twist to table design…

Our hottest table is unusually organised. Instead of having an object state column, we have *a main table defining the objects and a small auxiliary index-organised table per possible state*. Presence of rows in the auxiliary tables indicate the state the object has reached. *Creating a "state row" passes the baton to the next agent and the previous agent is no longer allowed to access the row in any way*. Once the baton exits the hot path, the last agent cleans up in an optimal manner as it knows the rows will never be accessed again.

This yields a high throughput for several reasons. Firstly, it capitalises on Oracle's good insert performance and avoids weaker update performance. Secondly, it generates little redo and implicitly guarantees perfect read consistency. Thirdly the row ownership rules eliminate row lock contention and most cache coherence traffic.

Hot table split

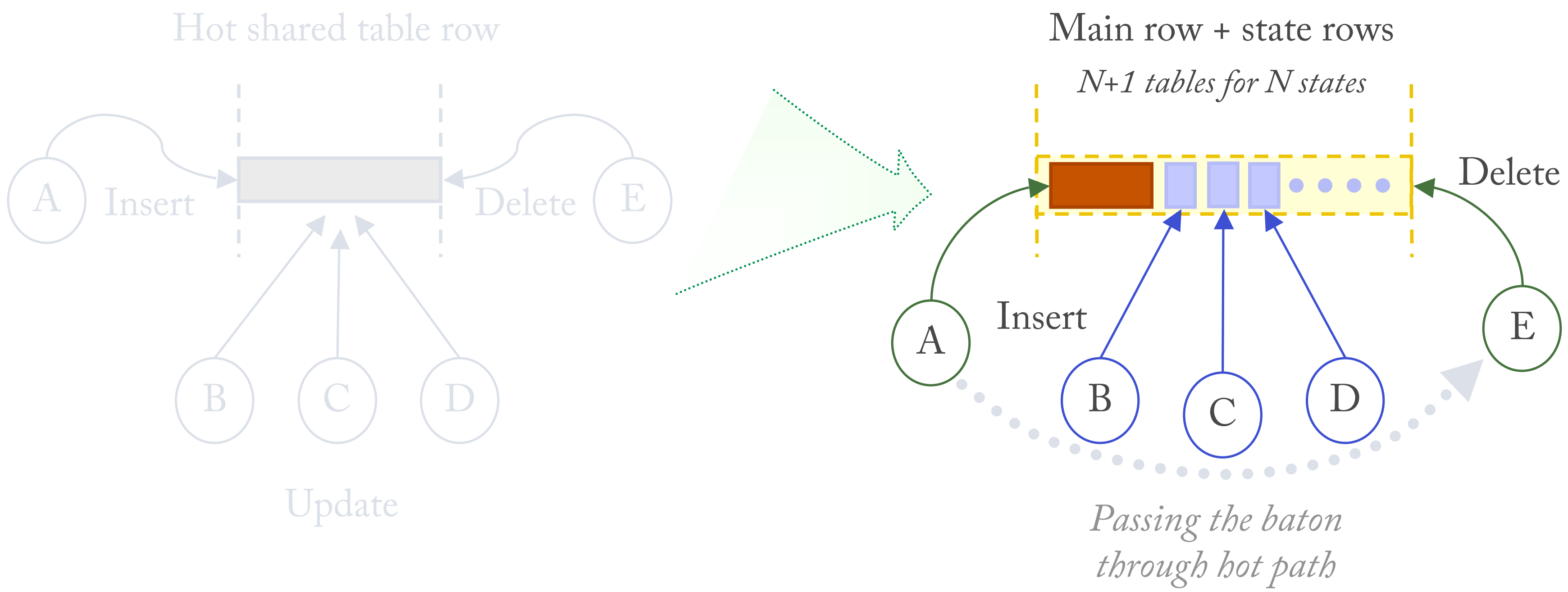
Avoiding row sharing



- Row lock contention for row state update
- False row sharing with mass row operations
- Rows ping-pong in the database cluster cache on update if agents connected to different servers
- Vulnerable to limited row update performance

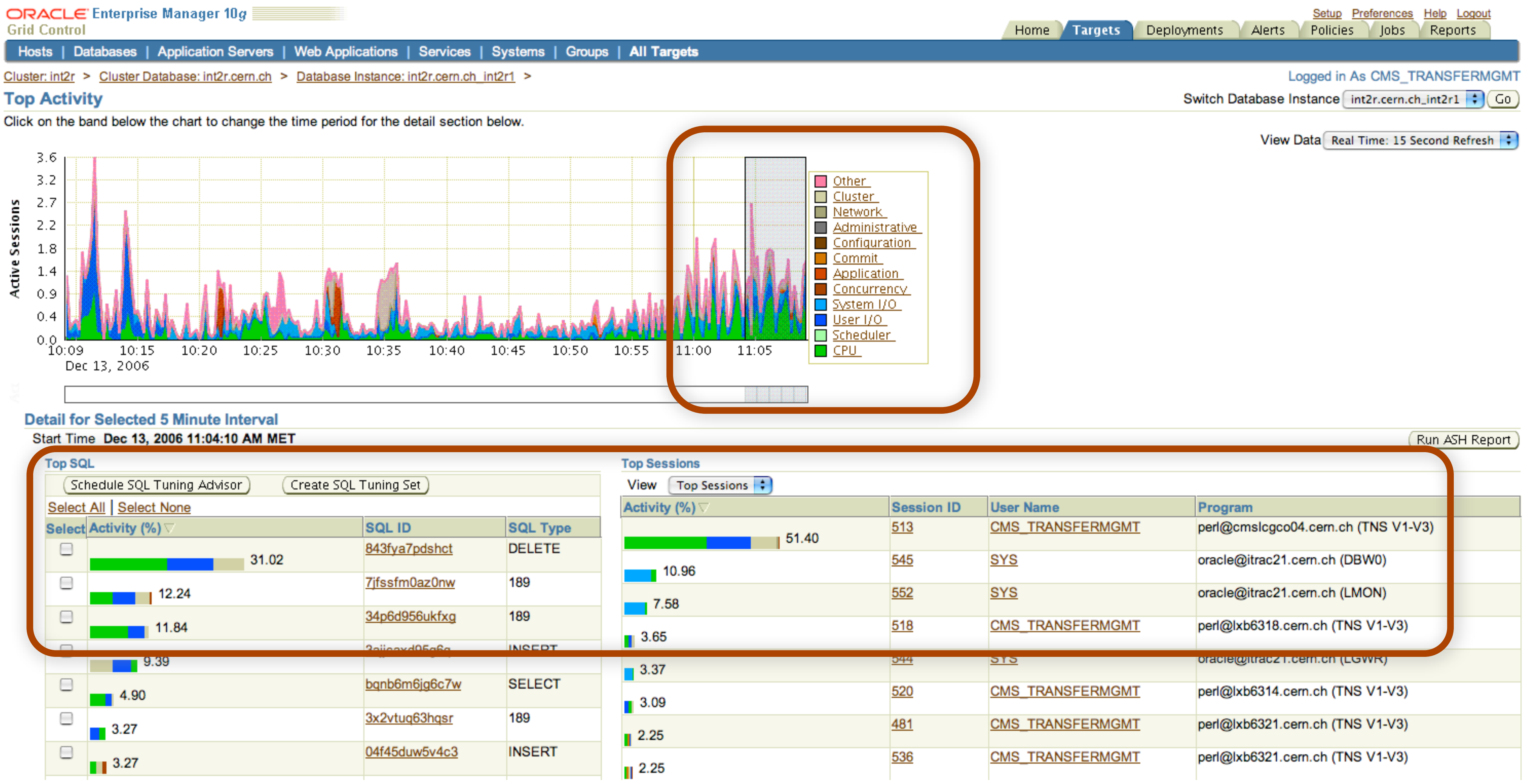
Hot table split

Avoiding row sharing



+ Cheaper locking due to read-only rows
+ No mass updates, no false row sharing
+ Rows stay put in database cluster cache
  or are shared read-only to all nodes
+ Capitalises on insert performance

Performance analysis using Oracle Enterprise Manager



Cluster: int2r > Cluster Database: int2r.cern.ch > Database Instance: int2r.cern.ch_int2r1 >

Logged in As CMS_TRANSFERMGMT

Top Activity

Switch Database Instance: int2r.cern.ch_int2r1

Click on the band below the chart to change the time period for the detail section below.

View Data: Real Time: 15 Second Refresh

Detail for Selected 5 Minute Interval

Start Time **Dec 13, 2006 11:04:10 AM MET**

Top SQL

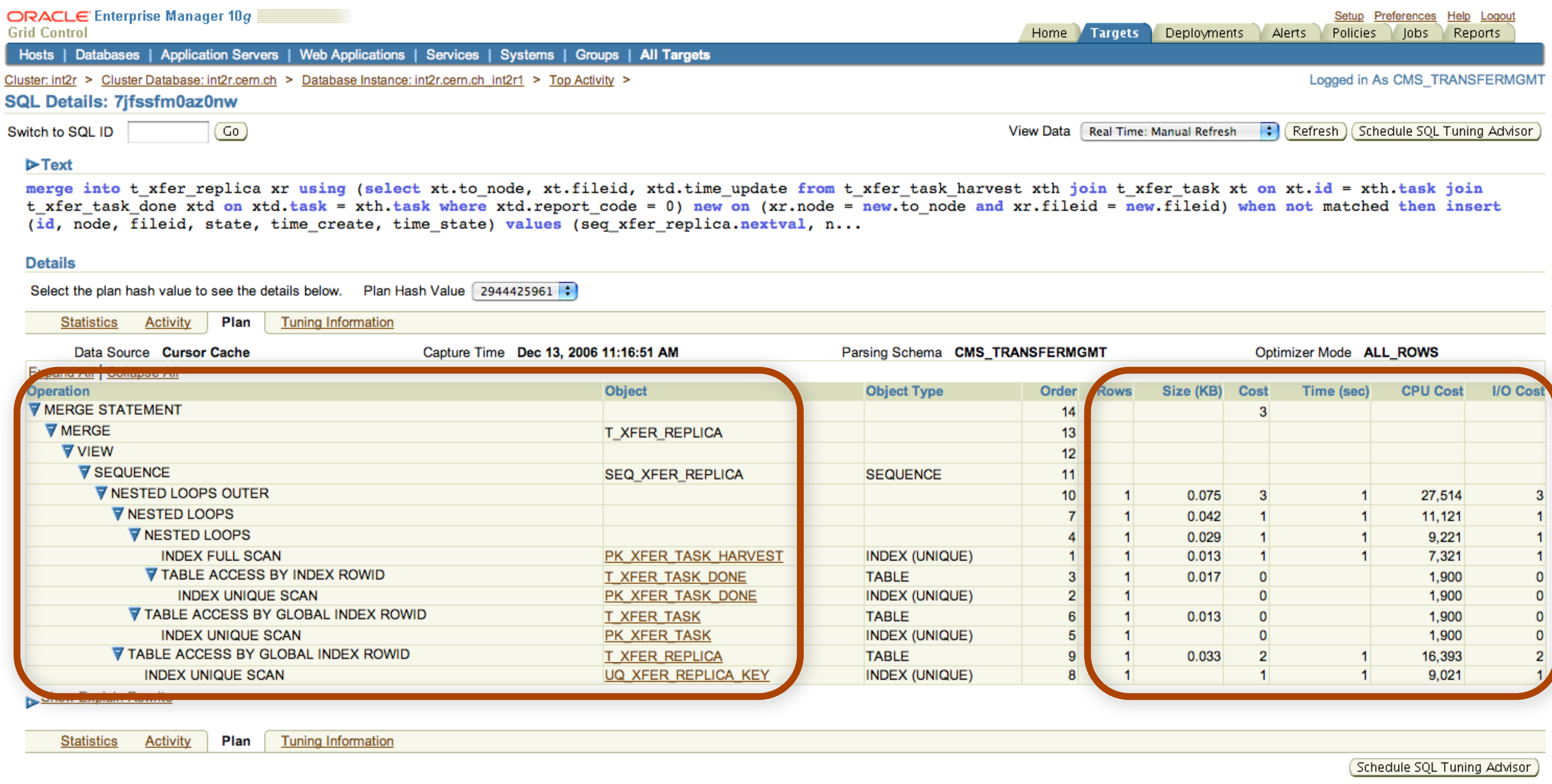
Select	Activity (%)	SQL ID	SQL Type
☐	31.02	843fya7pdshct	DELETE
☐	12.24	7jfssfm0az0nw	189
☐	11.84	34p6d956ukfxg	189
☐	9.39	3ajicaxd95g6q	INSERT
☐	4.90	bqnb6m6jg6c7w	SELECT
☐	3.27	3x2vtuq63hqsr	189
☐	3.27	04f45duw5v4c3	INSERT

Top Sessions

View: Top Sessions

Activity (%)	Session ID	User Name	Program
51.40	513	CMS_TRANSFERMGMT	perl@cmslcgco04.cern.ch (TNS V1-V3)
10.96	545	SYS	oracle@itrac21.cern.ch (DBW0)
7.58	552	SYS	oracle@itrac21.cern.ch (LMON)
3.65	518	CMS_TRANSFERMGMT	perl@lxb6318.cern.ch (TNS V1-V3)
3.37	544	SYS	oracle@itrac21.cern.ch (LGWR)
3.09	520	CMS_TRANSFERMGMT	perl@lxb6314.cern.ch (TNS V1-V3)
2.25	481	CMS_TRANSFERMGMT	perl@lxb6321.cern.ch (TNS V1-V3)
2.25	536	CMS_TRANSFERMGMT	perl@lxb6321.cern.ch (TNS V1-V3)

Performance analysis using Oracle Enterprise Manager



ORACLE Enterprise Manager 10g
Grid Control
Setup Preferences Help Logout
Home | Targets | Deployments | Alerts | Policies | Jobs | Reports
Hosts | Databases | Application Servers | Web Applications | Services | Systems | Groups | All Targets

Cluster: int2r > Cluster Database: int2r.cern.ch > Database Instance: int2r.cern.ch_int2r1 > Top Activity >

Logged in As CMS_TRANSFERMGMT

SQL Details: 7jfssfm0az0nw

Switch to SQL ID [ ] Go
View Data: Real Time: Manual Refresh | Refresh | Schedule SQL Tuning Advisor

Text

```
merge into t_xfer_replica xr using (select xt.to_node, xt.fileid, xtd.time_update from t_xfer_task_harvest xth join t_xfer_task xt on xt.id = xth.task join
t_xfer_task_done xtd on xtd.task = xth.task where xtd.report_code = 0) new on (xr.node = new.to_node and xr.fileid = new.fileid) when not matched then insert
(id, node, fileid, state, time_create, time_state) values (seq_xfer_replica.nextval, n...
```

Details

Select the plan hash value to see the details below. Plan Hash Value: 2944425961

Data Source **Cursor Cache** Capture Time **Dec 13, 2006 11:16:51 AM** Parsing Schema **CMS_TRANSFERMGMT** Optimizer Mode **ALL_ROWS**

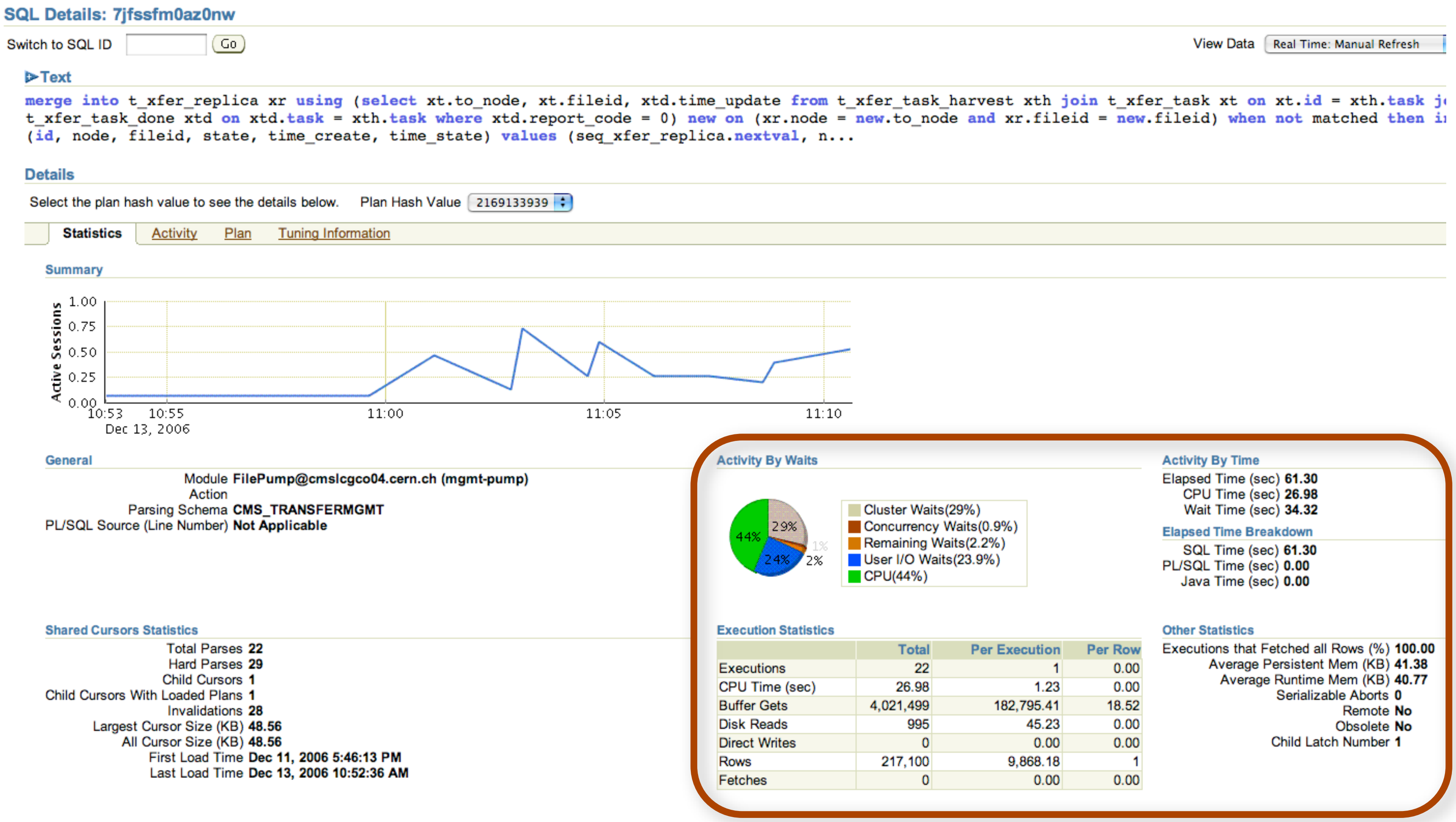
Expand All | Collapse All

Operation	Object	Object Type	Order	Rows	Size (KB)	Cost	Time (sec)	CPU Cost	I/O Cost
MERGE STATEMENT			14			3			
MERGE	T_XFER_REPLICA		13						
VIEW			12						
SEQUENCE	SEQ_XFER_REPLICA	SEQUENCE	11						
NESTED LOOPS OUTER			10	1	0.075	3	1	27,514	3
NESTED LOOPS			7	1	0.042	1	1	11,121	1
NESTED LOOPS			4	1	0.029	1	1	9,221	1
INDEX FULL SCAN	PK_XFER_TASK_HARVEST	INDEX (UNIQUE)	1	1	0.013	1	1	7,321	1
TABLE ACCESS BY INDEX ROWID	T_XFER_TASK_DONE	TABLE	3	1	0.017	0		1,900	0
INDEX UNIQUE SCAN	PK_XFER_TASK_DONE	INDEX (UNIQUE)	2	1		0		1,900	0
TABLE ACCESS BY GLOBAL INDEX ROWID	T_XFER_TASK	TABLE	6	1	0.013	0		1,900	0
INDEX UNIQUE SCAN	PK_XFER_TASK	INDEX (UNIQUE)	5	1		0		1,900	0
TABLE ACCESS BY GLOBAL INDEX ROWID	T_XFER_REPLICA	TABLE	9	1	0.033	2	1	16,393	2
INDEX UNIQUE SCAN	UQ_XFER_REPLICA_KEY	INDEX (UNIQUE)	8	1		1	1	9,021	1

Show Explain Rewrite

Statistics | Activity | Plan | Tuning Information
Schedule SQL Tuning Advisor

Performance analysis using Oracle Enterprise Manager



SQL Details: 7jfssfm0az0nw

Switch to SQL ID [ ] Go View Data Real Time: Manual Refresh

Text

```
merge into t_xfer_replica xr using (select xt.to_node, xt.fileid, xtd.time_update from t_xfer_task_harvest xth join t_xfer_task xt on xt.id = xth.task j
t_xfer_task_done xtd on xtd.task = xth.task where xtd.report_code = 0) new on (xr.node = new.to_node and xr.fileid = new.fileid) when not matched then i
(id, node, fileid, state, time_create, time_state) values (seq_xfer_replica.nextval, n...
```

Details

Select the plan hash value to see the details below. Plan Hash Value 2169133939

Statistics Activity Plan Tuning Information

Summary

General

Module	**FilePump@cmslcgco04.cern.ch (mgmt-pump)**
Action	
Parsing Schema	**CMS_TRANSFERMGMT**
PL/SQL Source (Line Number)	**Not Applicable**

Activity By Waits

▨	Cluster Waits(29%)
🟫	Concurrency Waits(0.9%)
🟧	Remaining Waits(2.2%)
🟦	User I/O Waits(23.9%)
🟩	CPU(44%)

Activity By Time

Elapsed Time (sec)	**61.30**
CPU Time (sec)	**26.98**
Wait Time (sec)	**34.32**

Elapsed Time Breakdown

SQL Time (sec)	**61.30**
PL/SQL Time (sec)	**0.00**
Java Time (sec)	**0.00**

Shared Cursors Statistics

Total Parses	**22**
Hard Parses	**29**
Child Cursors	**1**
Child Cursors With Loaded Plans	**1**
Invalidations	**28**
Largest Cursor Size (KB)	**48.56**
All Cursor Size (KB)	**48.56**
First Load Time	**Dec 11, 2006 5:46:13 PM**
Last Load Time	**Dec 13, 2006 10:52:36 AM**

Execution Statistics

	Total	Per Execution	Per Row
Executions	22	1	0.00
CPU Time (sec)	26.98	1.23	0.00
Buffer Gets	4,021,499	182,795.41	18.52
Disk Reads	995	45.23	0.00
Direct Writes	0	0.00	0.00
Rows	217,100	9,868.18	1
Fetches	0	0.00	0.00

Other Statistics

Executions that Fetched all Rows (%)	**100.00**
Average Persistent Mem (KB)	**41.38**
Average Runtime Mem (KB)	**40.77**
Serializable Aborts	**0**
Remote	**No**
Obsolete	**No**
Child Latch Number	**1**

But wait… if you didn’t test it,

It Doesn’t Work!

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

Brian W. Kernighan

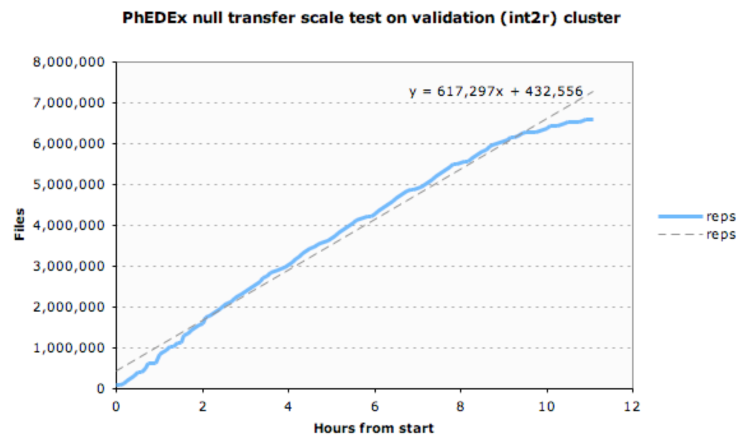
CMS operates three PhEDEx instances in parallel at all times: a developer instance, an integration and commissioning instance and the main production instance. In addition to major computing challenges approximately once a year, the developers perform a detailed design validation on average twice a year.

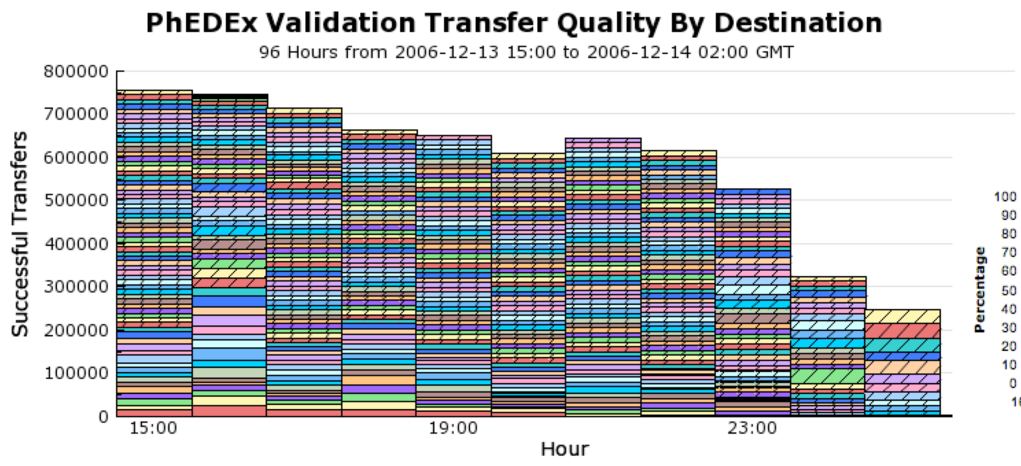
The validation is performed always before releasing a major schema upgrade and from time to time to verify accumulated patches have not introduced undesirable side effects. The validation takes place on an isolated, dedicated database cluster. Once the final results have been found repeatable, the findings are written up as a technical report. The CERN database administrators are involved and decide whether the schema may be released to the production service.

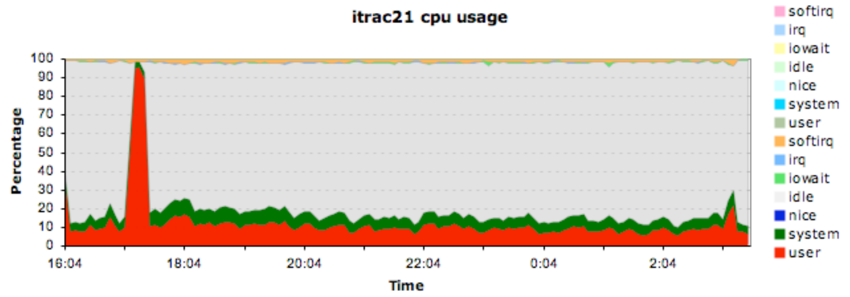
Validation tests both verify functionality and error handling, and stress test realistic behaviour with problem sizes exceeding expected peak load by a factor of about 100. The system is required to handle the load gracefully in order to pass the test.

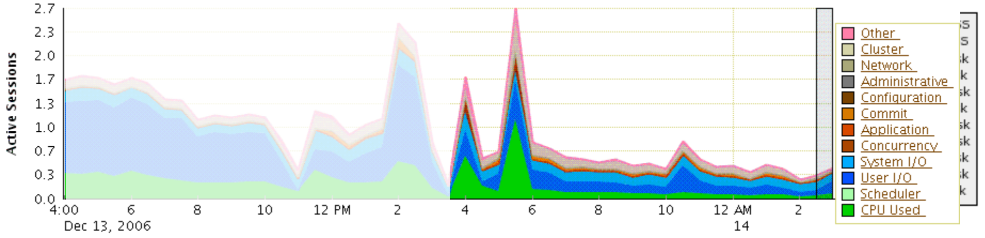
PhEDEx 2.5 validation

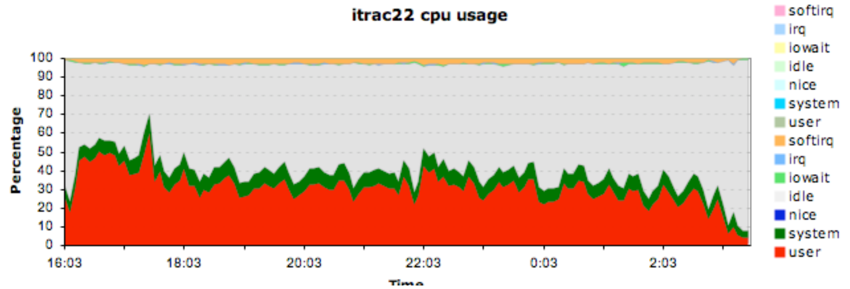
Examples of reported metrics

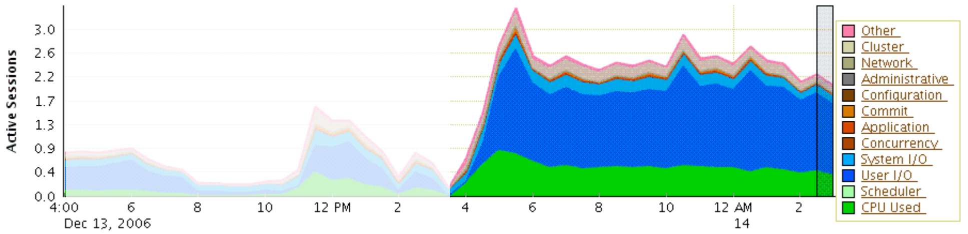


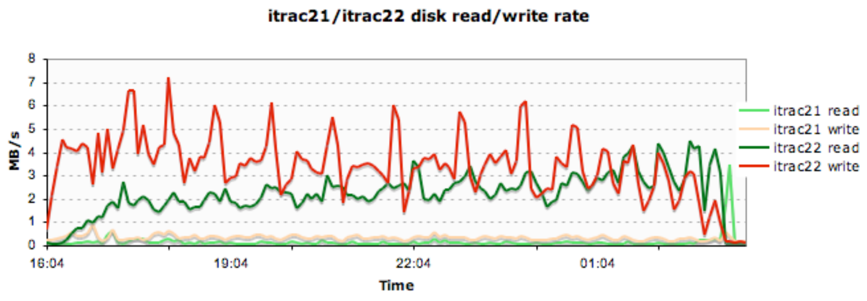


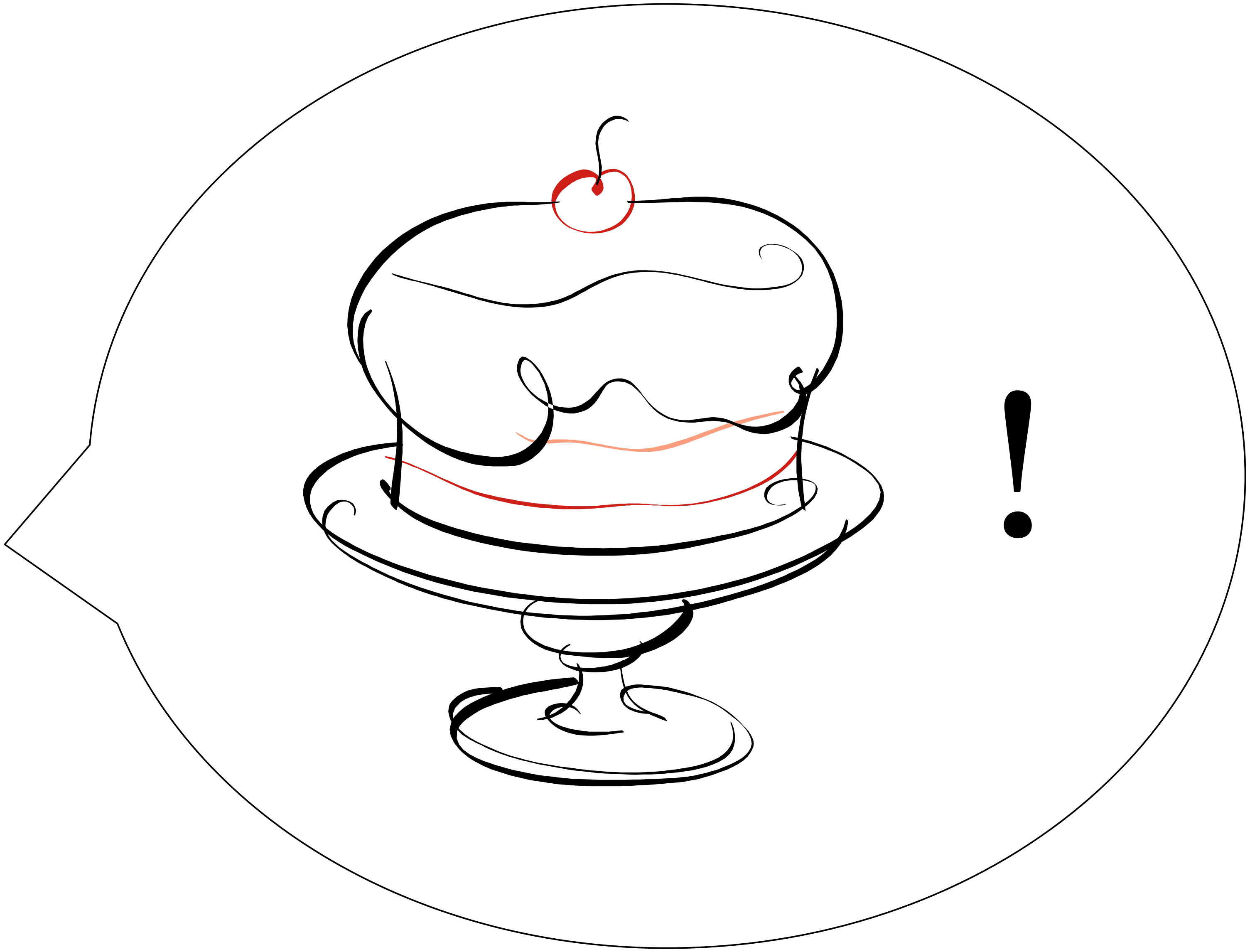












CMS has extensive experience with grid transfer technologies. We have used all grid transfer services very soon after release in large scale validation tests, including all the LCG service challenges. We have communicated back our findings promptly. How does the CMS approach to data transfers compare with those of other LHC experiments and grid projects?

PhEDEx was designed to address data placement and reliable transfers. The LHC experiments arrived together at the conclusion that currently there is no demand for a middleware product for data placement: this function is tightly coupled with the experiment computing models, policies and dataset bookkeeping systems.

Portions of EGEE's FTS were modeled after PhEDEx, a significant departure from designs in previous grid projects. FTS was expected to replace the lowest data transfer layers of PhEDEx. After a strained initial relationship, that is now largely the case at the EGEE sites.

ATLAS and CMS, apparently independently, have arrived at similar conclusions on high-level data management concepts. Both place data at sites as a deliberate policy action, not reactively in response to jobs. The data "subscription" processes are similar. Both have data units larger than files but smaller than datasets and an agent based transfer management design.

We have prepared for CMS start-up by continuously pushing our data transfer system to ever larger scales, both in technical ability and actual daily operation. Nevertheless the work continues.

The storage system and data transfer commissioning work continues to push for scale, and more importantly, for quality.

Some aspects of our computing model remain to be exercised at full scale. The major CMS challenge for 2007, CSA07, will exercise the largest ensemble of workflows and services yet. Regular transfers to smaller institutes and desktops are only starting about now.

Development will continue to improve in particular the PhEDEx web site, the main hub for data transfer operations. Many higher-level data management concepts are not yet packaged conveniently. There is increasing demand for making various administrative and operations tasks easier, and to enable ever more control of the system via the web interface.

We are beginning to draw plans for new challenges in 2009 and beyond.

My main concern in a nutshell

“What we have here is a case of too many pieces of software trying to outsmart each other and the user loses.”

Robert Lipe

GCC list, Oct 1998