COOL for Atlas Prompt Reconstruction

Further Performance Studies

Andrea Valassi

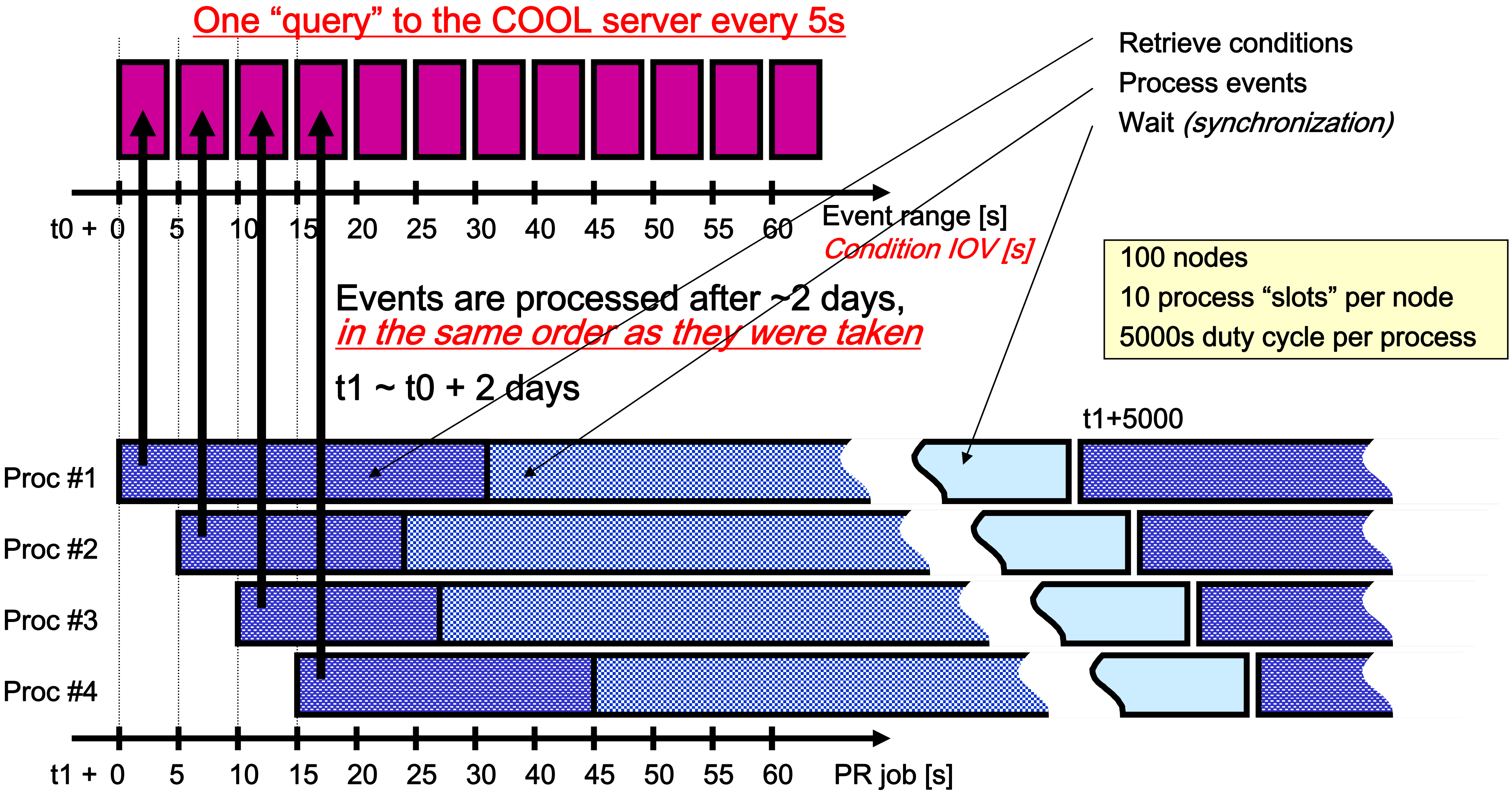
(CERN IT-ADC)

Motivation

- **Performance problems observed by David in his tests**
  - Where is the bottleneck in David's tests?
  - Is it a software problem or a server configuration problem?
  - Is there an area of parameter space with no performance problems?

- **Parallel study using a complementary approach**
  - Try to reproduce closely the timing of the Atlas prompt reconstruction
  - Start with simplified scenario to "calibrate" the tests, add complications one by one

Model for Atlas prompt reconstruction

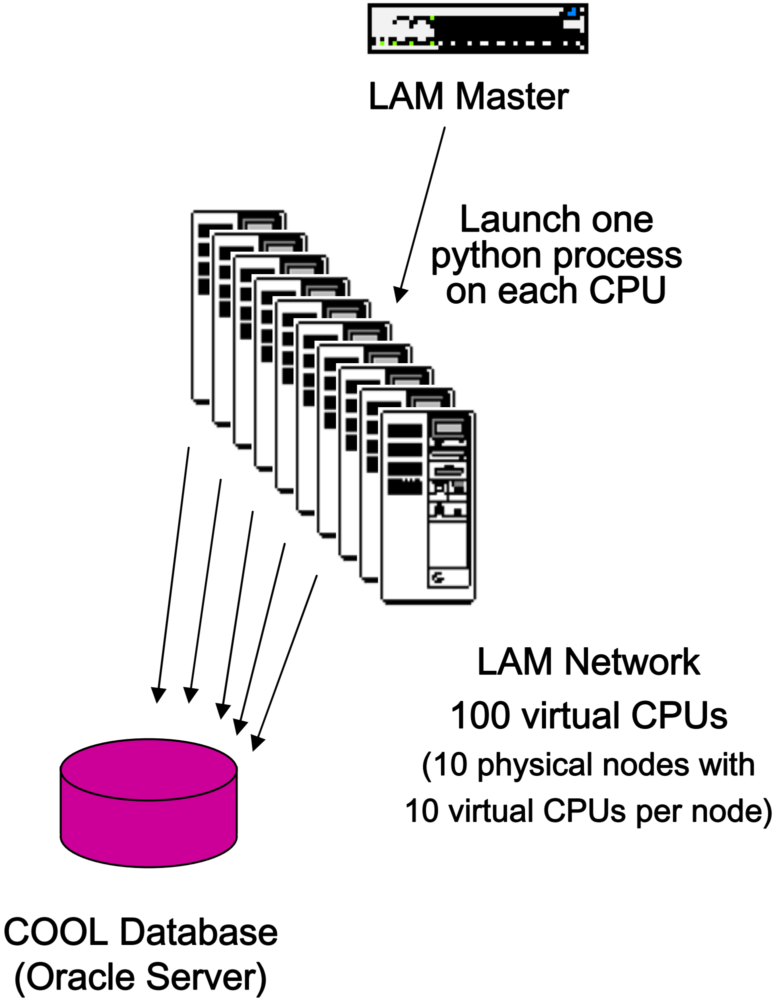


100 nodes

10 process "slots" per node

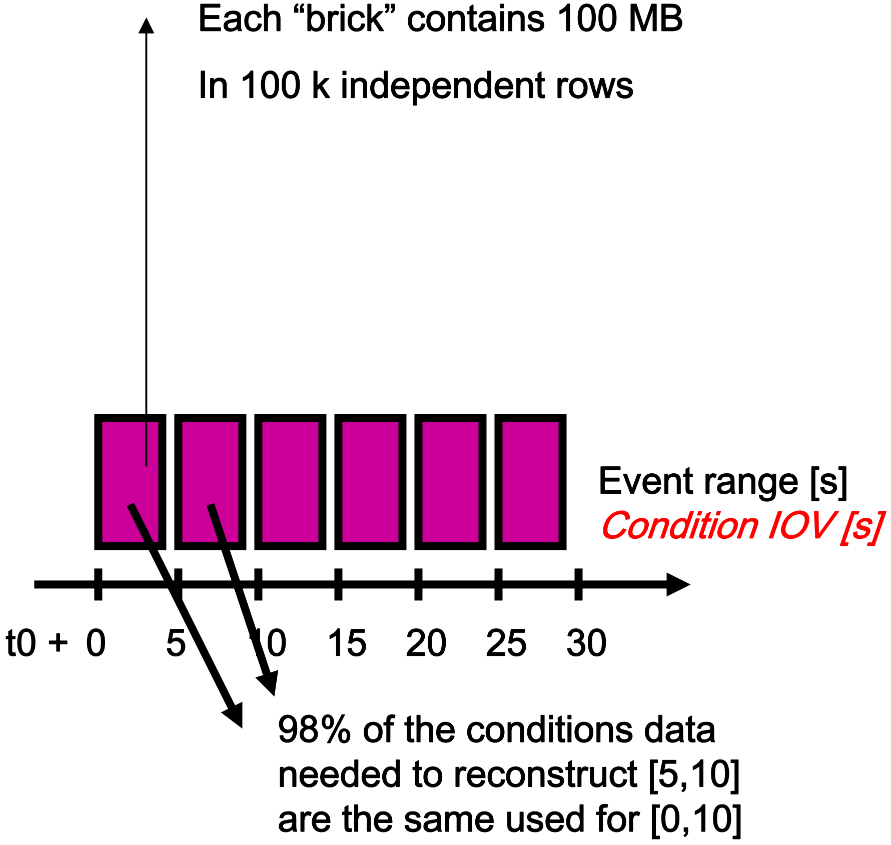
5000s duty cycle per process

Test setup


LAM Master
Launch one python process on each CPU
LAM Network
100 virtual CPUs
(10 physical nodes with 10 virtual CPUs per node)
COOL Database (Oracle Server)

- **Multi-client setup using LAM/MPI**
  - Technology of Harp/Compass migration
- **100 simultaneous python processes**
  - One on each of 100 virtual CPUs
- **Python process is a "scheduler"**
  - Every 500s it opens a new connection to the COOL database via PyCool and retrieves the conditions for the 5s of event data it pretends to be processing
  - Processes on 100 different virtual CPUs start at 5s intervals from each other
  - Real Atlas case foresees 1000 processes, but the 5000s available are for event processing!
- **Main outcome: efficiency**
  - Essentially, how many queries in < 500s
  - If the server cannot handle all requests, some would take >500s (neg. feedback)
  - Standalone query takes << 500s

Simulated conditions data


Each "brick" contains 100 MB
In 100 k independent rows
Event range [s]
Condition IOV [s]
t0 + 0
5
10
15
20
25
30
98% of the conditions data
needed to reconstruct [5,10]
are the same used for [0,10]

- Full "snapshot" description of Atlas
  - 100 MB in 100k *independent* channels

    (by the way: is it really true that 100k items have independent IOVs?)

- Server delivers one snapshot every 5s
  - ***Database server must deliver 20 MB/s and 20k table rows/s***

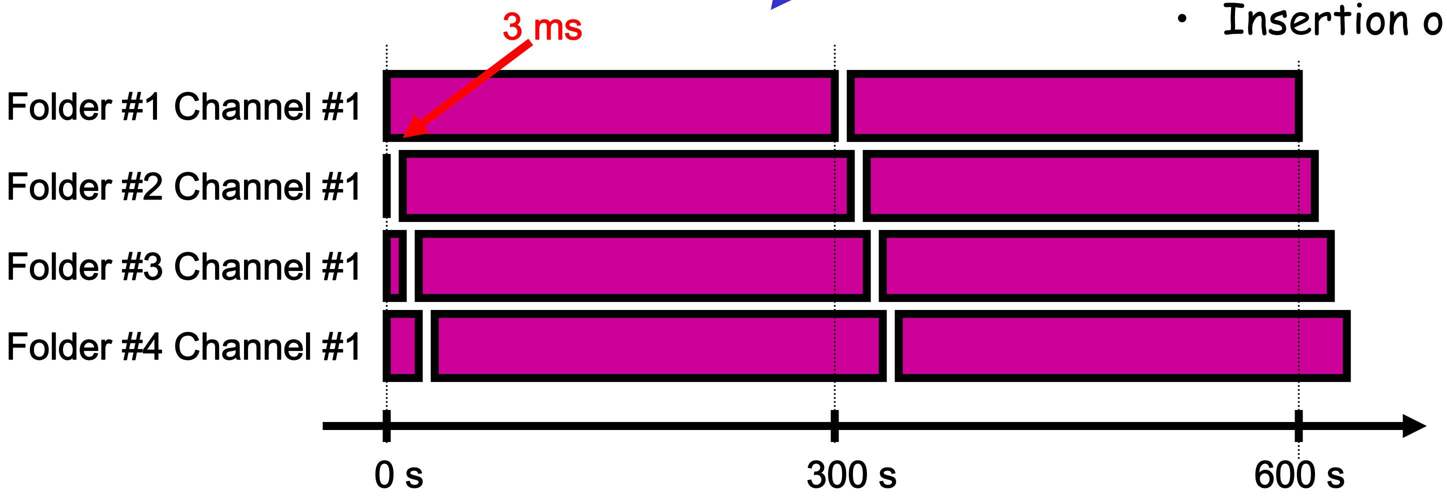
- Average IOV ~ 5 minutes
  - Only 1/60 of conditions data change from [0s,5s] to [5s,10s]
  - *If data are cached in the database server memory, server I/O must read only 300 kB/s and 300 table rows/s?*

Conditions data samples

- Payload per IOV ~ 1kB
  - A 1000-character *(random)* string

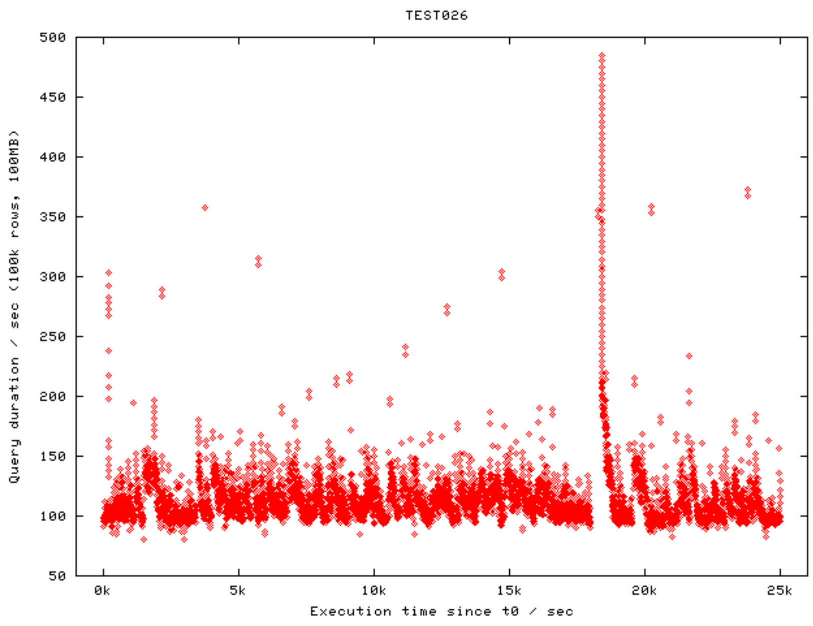
- Control (initial) samples
  - All jobs retrieve same 100k rows
  - "Browse" 1 folder with 1 channel
    - 100k 1s-IOVs (i.e. 100k rows)
  - Total data in the database: 100MB

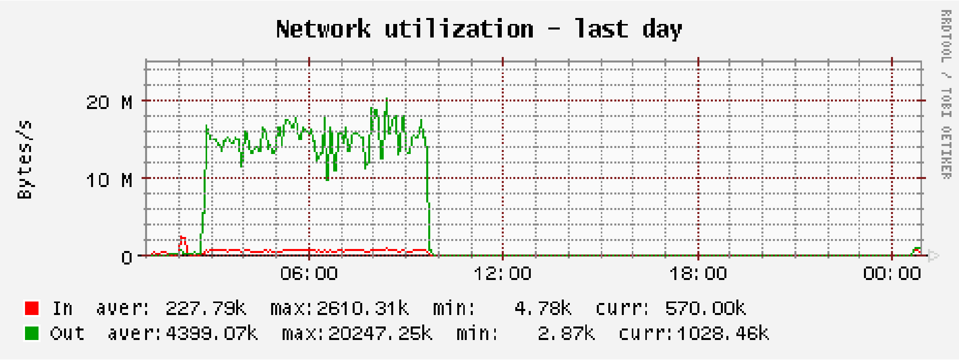
- Realistic (final) samples
  - 100 folders with 1k channels each
    - COOL multi-channel bulk retrieval: 1k rows from each of 100 tables
  - All IOVs are exactly 5 min (300s)
    - *IOVs are 3ms apart from one other*
  - Total data in the database: 6 GB
    - 5 hours of conditions data
    - Insertion order: by channel, since

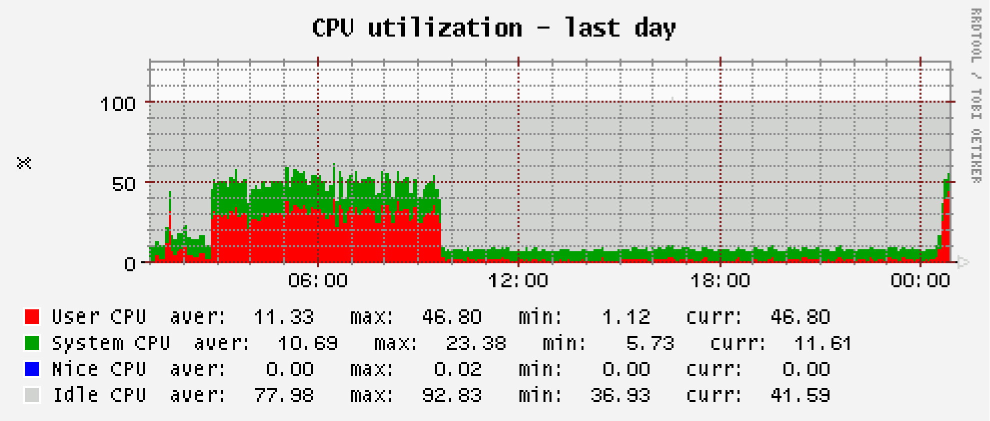


The outcome: SUCCESS!

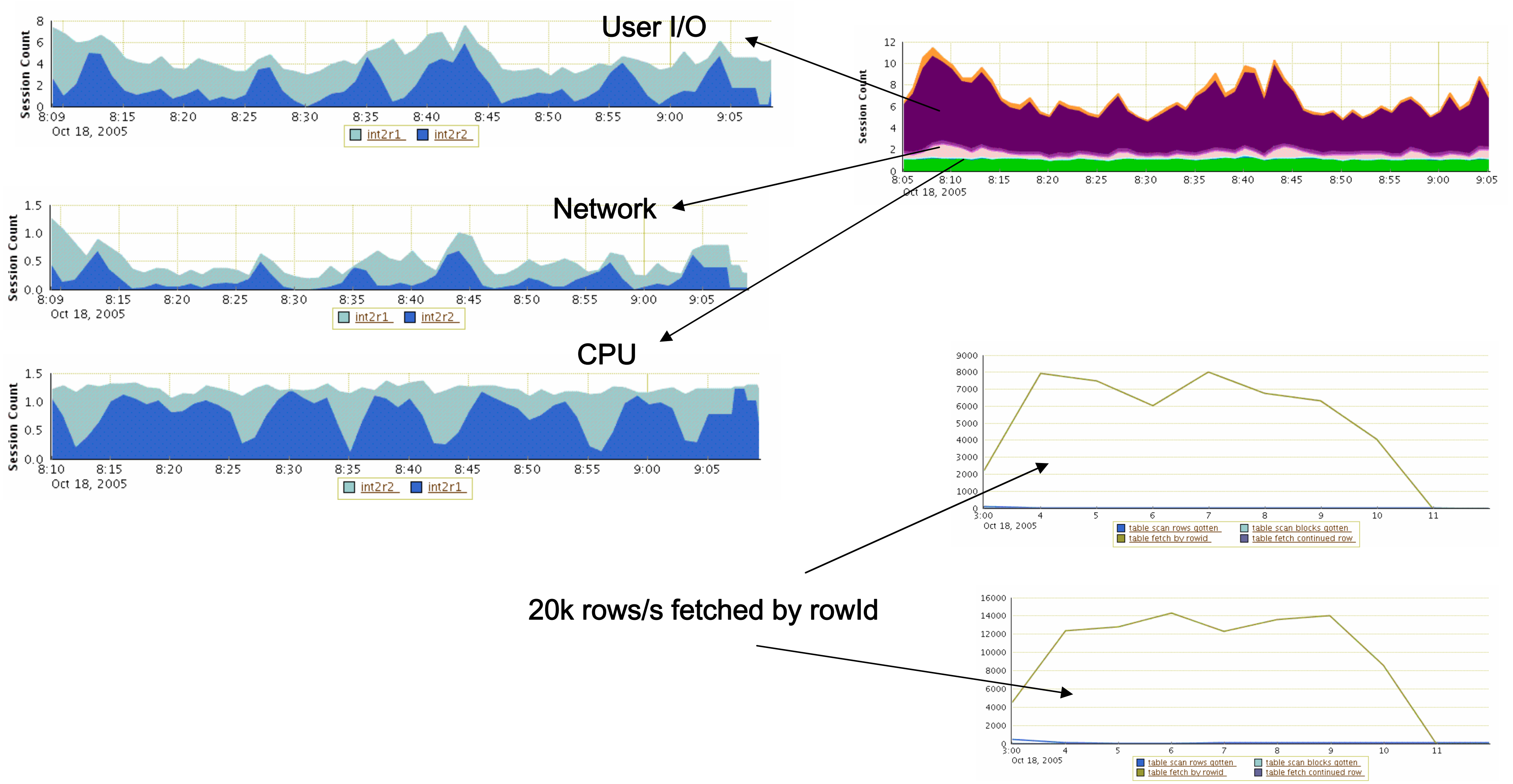
- **Latest test on "Integration RAC"**
  - 1.2 GB buffer cache in memory
- **5000/5000 jobs successful**
  - Typical query time ~ 80-150s
  - 80s: typical query time for a standalone node with data already in memory
    - 500-1000s after FLUSH BUFFER CACHE
- **RAC cluster handles the load well**
  - 50% CPU load on each node
  - Sustained network rate ~ 12 + 9 MB/s







Steady-state Atlas PR - server



User I/O

Network

CPU

20k rows/s fetched by rowId

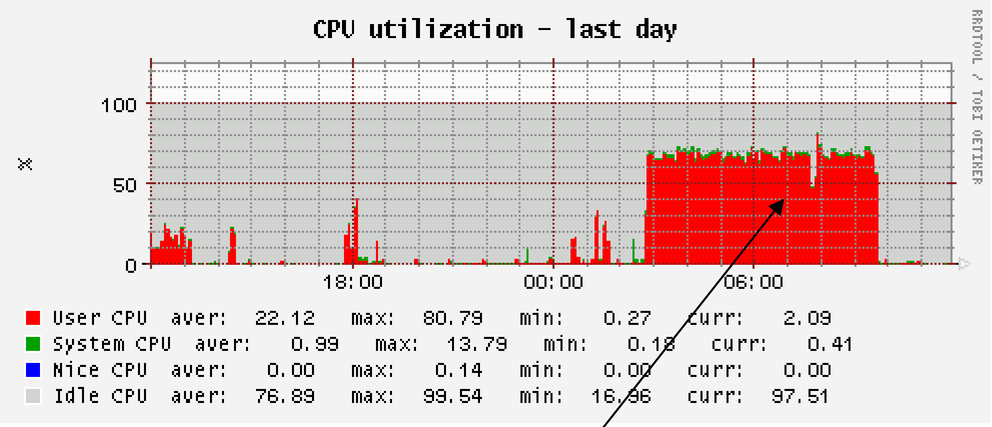
Steady-state Atlas PR - client

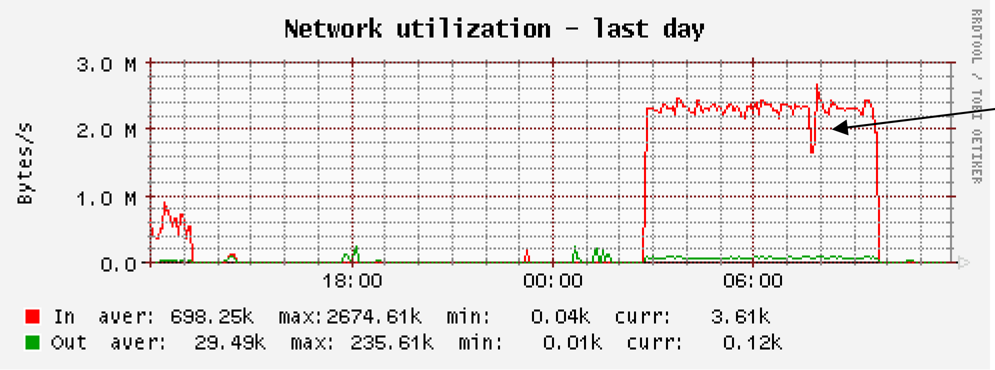
Typical client

10 simultaneous python processes

Data retrieved ~ 2.1 MB / s as expected

Constant CPU load: only ~ 70%



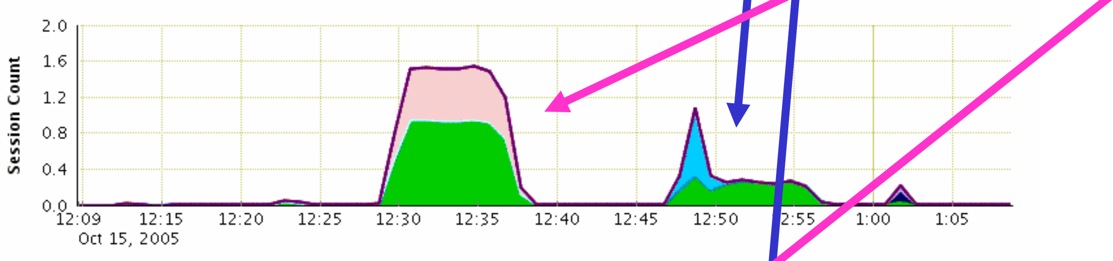


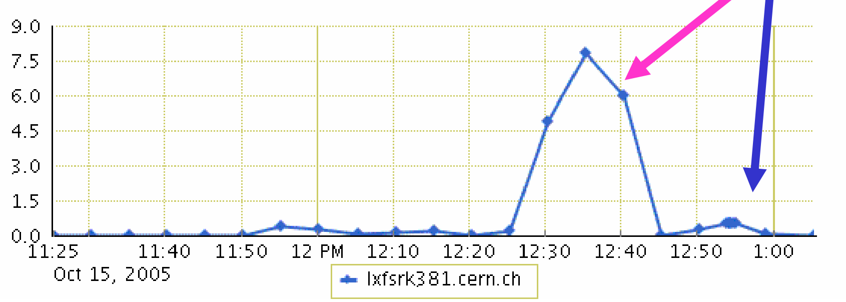
Database response problem at 7am

Observe it also on the plot on the previous slide (query time ~ 400-500s)!

To be understood… (1)

- Network data rates: Oracle data compression?
  - Tests using two almost identical control samples: ~ 10MB/s for random payload strings vs. only 0.5 MB/s for “000….000” strings!
  - Confirmed by preliminary study of SQLnet trace (thanks to Luca!)
  - *In practice: make sure you use random (or at least non identical?) payloads for any performance tests!*



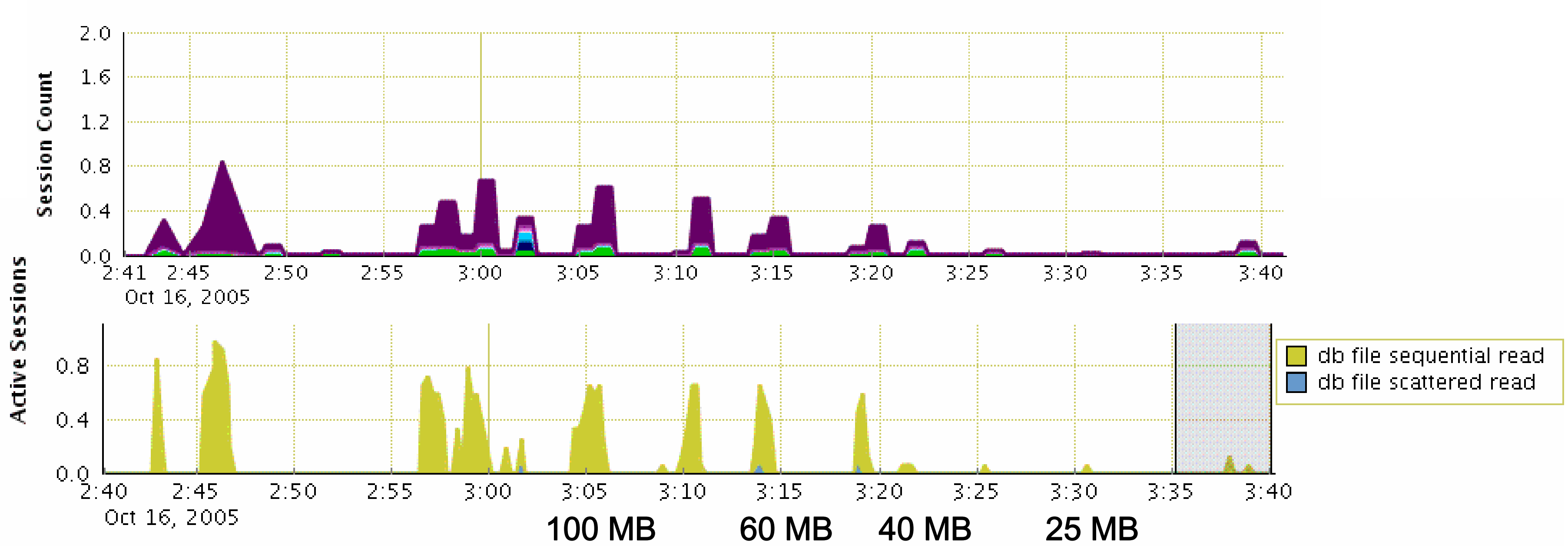


To be understood… (2)

- **Buffer cache: how much non-relevant data in there?**
  - Threshold effect observed in 40MB -> 80MB transition for a database server with 400 MB buffer cache size using early "realistic" sample
    - "Calibration" of 40 MB maximum size: if you don't even manage to re-read the same "snapshot" without I/O, then you are in trouble…
  - 100MB data snapshot comfortably retrieved from "control" sample using exactly the same database server!

- **Effects probably caused by data distribution across blocks**
  - An 8kB block is likely to contain data from 8 or more 1kB rows
  - *Inserting "by since, channel" better than "by channel, since"?*
  - *Evaluate index-organized tables?*

Buffer cache ~ 400 MB

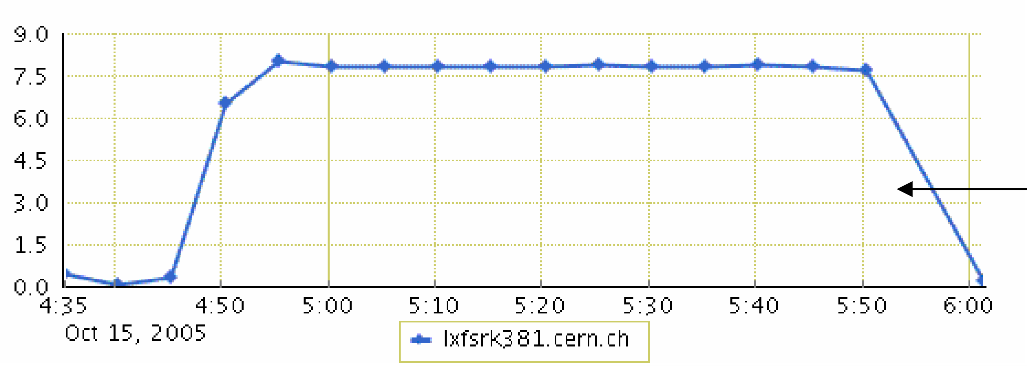


100 MB 60 MB 40 MB 25 MB

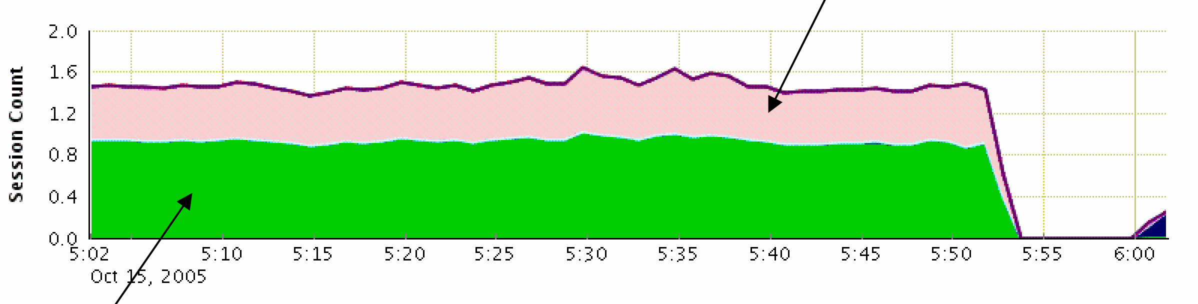
80 MB 50 MB 25 MB 40 MB

- **Reduce "snapshot" data volume until it can be re-read with no I/O**
  - For N mega bytes, read only N / 100 folders
  - 40 MB fit all in the buffer cache, 50 don't!

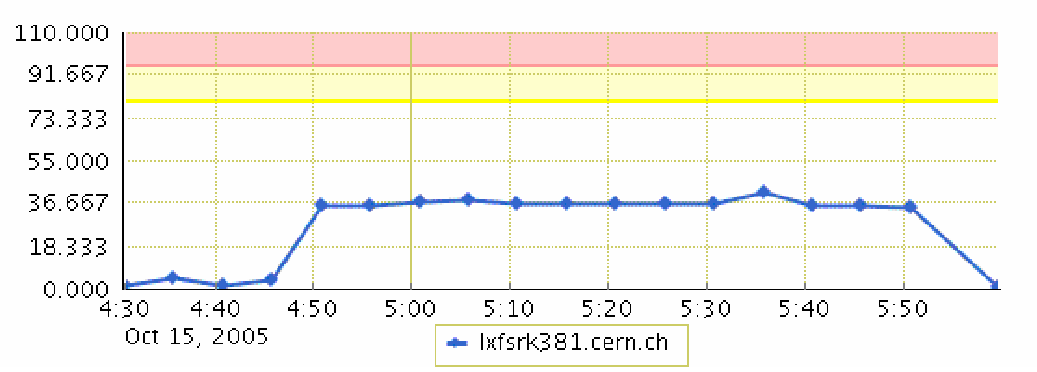
100 MB control sample - sustained retrieval



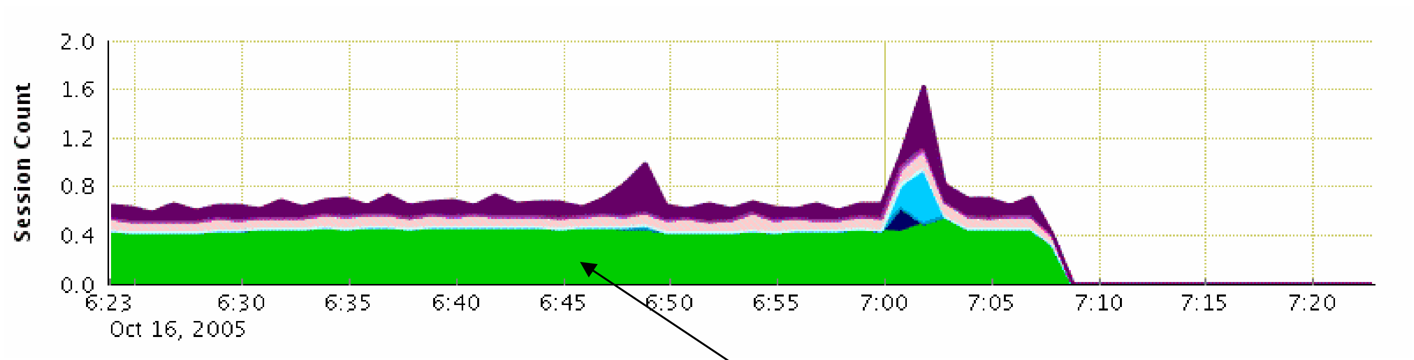
Network (why not 20 MB/s?)



Server CPU ~ only 35%!



40 MB vs 80 MB: much worse than double the query time!



80 MB snapshot volume on coolprod (400 MB cache)

Sustained stress test with only 85% efficiency

HUGE increase in I/O

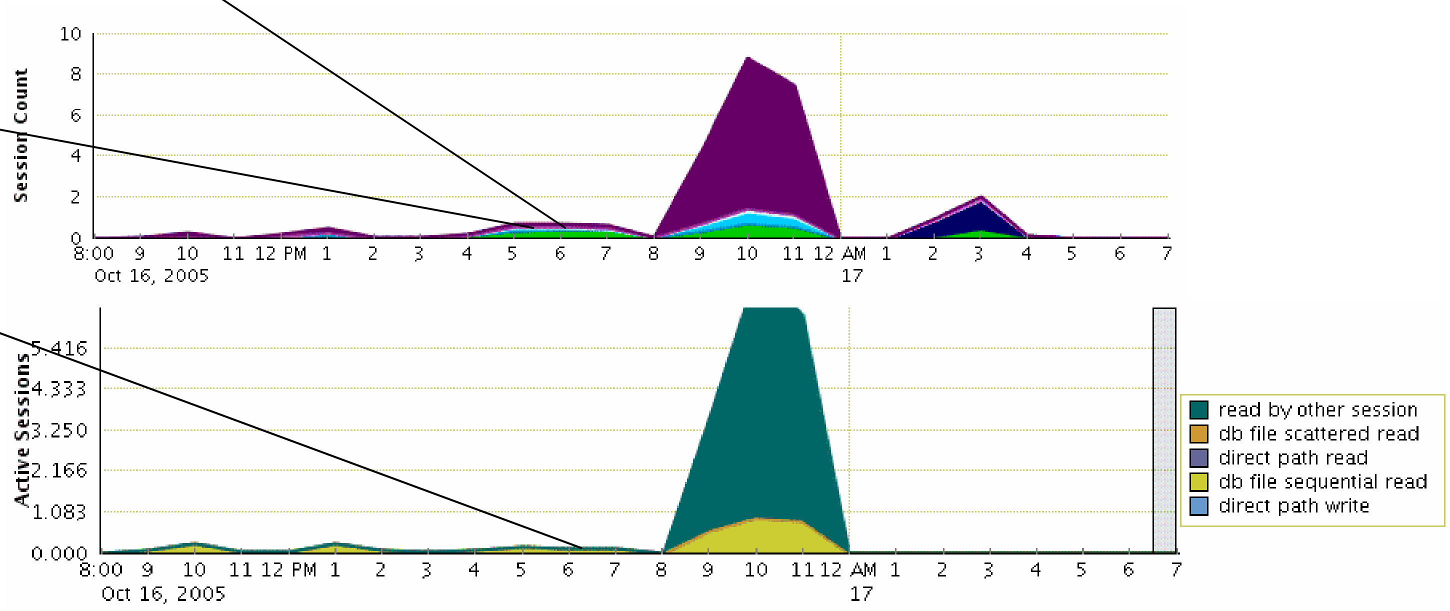
ZOOM

40 MB snapshot volume on coolprod (400 MB cache)

Sustained stress test with 99% efficiency!

Mainly CPU: some network, some I/O read

Like 100 MB on integration RAC (1.2 GB cache)



Known software limitations

- **Non-uniformity of IOV retrieval**
  - For the purpose of these tests, use tables with few IOVs
    - One of the differences with David's tests
  - Work in progress (next main priority)

- **Other sub-optimal execution plans most certainly exist**
  - A new one discovered thanks to these Atlas tests: multi-channel bulk retrieval for all channels should not use the index on channel!
  - Preliminary task (Uli?): systematic testing and trace retrieval

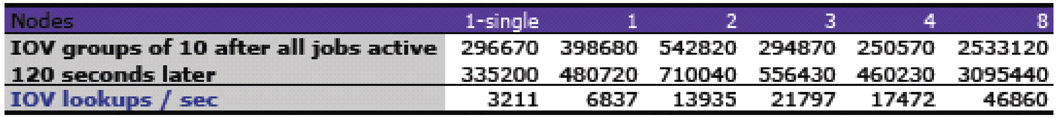
- **No multi-channel bulk insertion**
  - Design ideas waiting to be prototyped since many months
    - Waiting for bulk update/delete in RAL (already available in CORAL)
  - Populating the sample databases for the Atlas PR tests was a pain!

Some feedback for Atlas

- **Are there really 100k independent channels?**
  - On average, one condition changes every 3ms
  - Performance penalty if modeling *correlated* IOVs as *independent*

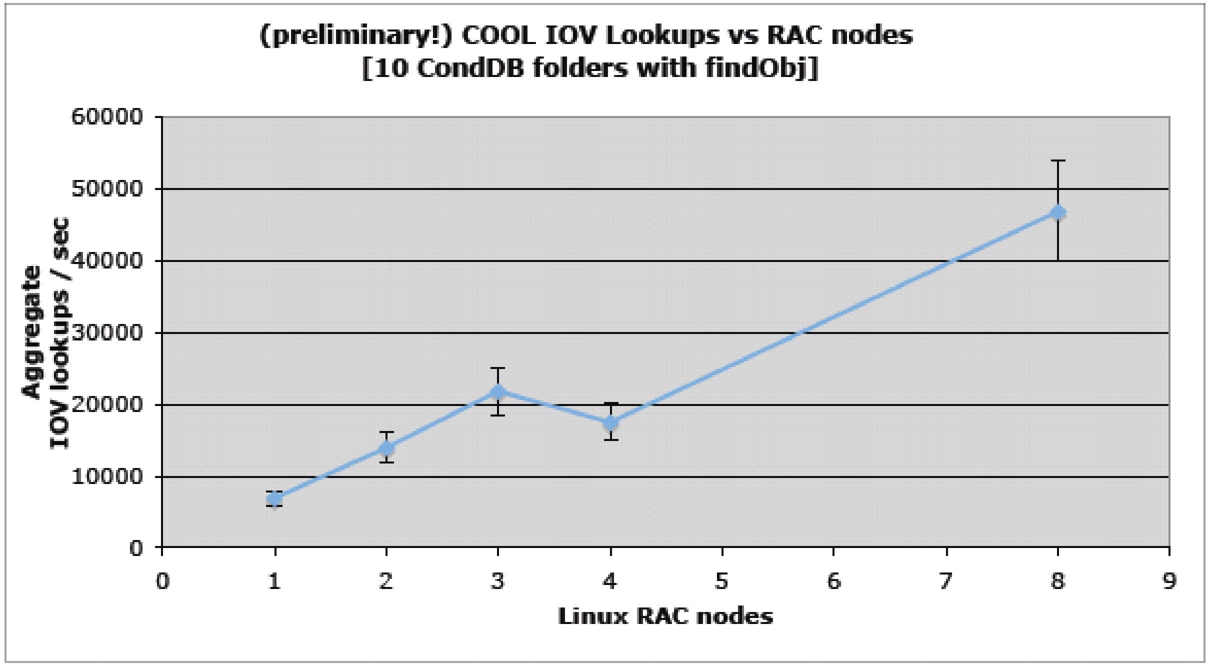
- **How stringent is constraint to process 5-second chunks?**
  - COOL database load easily reduced if processing longer chunks
  - Relying on database caching from the start may be dangerous

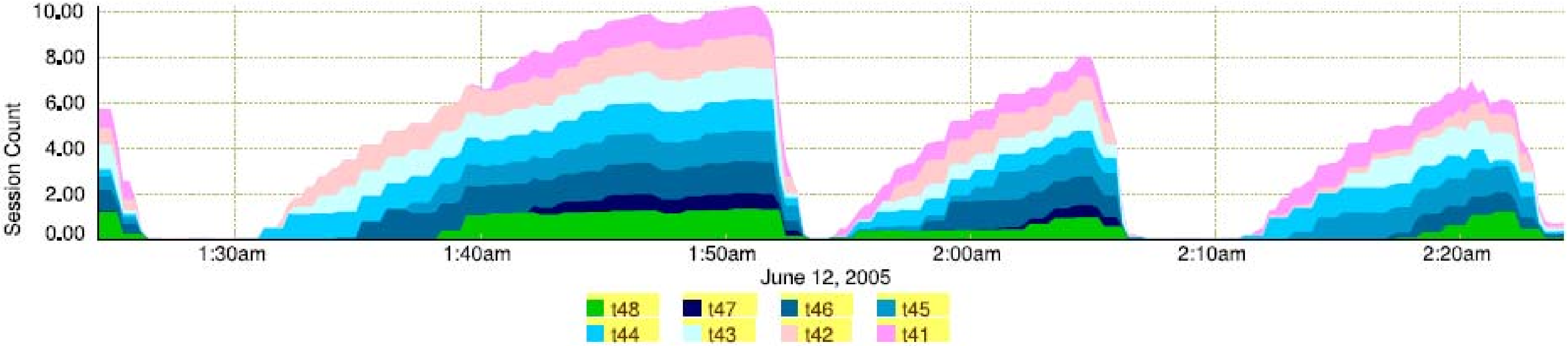
- **Database response quality is highly non-linear**
  - Doubling conditions data volume from 40 MB to 80 MB results in much worse than just double the query time…
  - ***Estimating precise requirements may well be difficult, but is crucial!***



Nodes	1-single	1	2	3	4	8
IOV groups of 10 after all jobs active	296670	398680	542820	294870	250570	2533120
120 seconds later	335200	480720	710040	556430	460230	3095440
IOV lookups / sec	3211	6837	13935	21797	17472	46860

In parallel: RAC scaling tests (Dirk)





Summary

- ***Under well-defined conditions, Atlas use case is validated!***
  - 100% efficiency on 5000 client jobs
  - "Integration RAC" handles the load rather comfortably
- **Database response is highly sensitive to many parameters**
  - David's initial tests were performed in "problem areas"
  - ***Detailed realistic prediction of user requirements is crucial***
- **Performance testing and application validation is essential**
  - Learn that your assumptions were wrong
  - This applies more generally than just to COOL alone
- **COOL is a software component with a limited and precise scope**
  - Understanding the performance of "only" that is already a big task!!
- **Outcome of these tests may suggest changes to the experiment model**
  - Feedback from Atlas about these tests will be useful! ☺