Control System Studio archiver with PostgreSQL back-end

Optimizing performance and reliability for a production environment*

M. Konrad†, C. Burandt, J. Enders, N. Pietralla

Institut für Kernphysik, Technische Universität Darmstadt, Germany

TECHNISCHE UNIVERSITÄT DARMSTADT

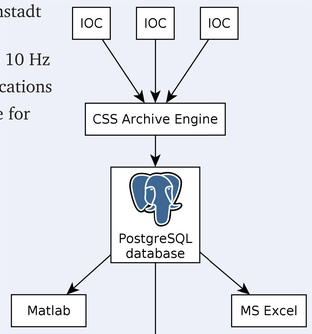
Requirements and Hardware

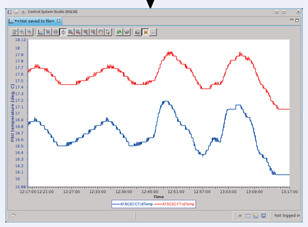
- archiving system for the Superconducting Darmstadt Linear Accelerator (S-DALINAC)
- tens of thousands of EPICS channels with up to 10 Hz
- should provide read access from different applications

⇒ based on Control System Studio Archive Engine for relational database (RDB)

⇒ PostgreSQL 9.1 on Debian Linux as back-end

⇒ CSS Data Browser as control room front-end



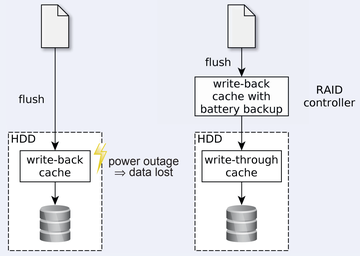


Main-board	Supermicro X9DRi-F
CPU	2x Intel Xeon E5-2643 @3.3 GHz
Main memory	128 GB DDR3 reg. ECC
RAID controller	Adaptec 6805 SAS2
Disks	36x SAS 300 GB

Disk performance and reliability

Reliable writes

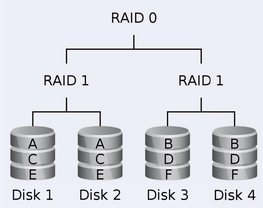
- RDBs flush data to disk after each commit
- data can be lost in case of power-outages if disk write-back caches are used
  - ⇒ use RAID controller with non-volatile write-back cache
  - ⇒ run RDB back-end on physical machine (no virtual machine)



Performance

- heavy random access workload during queries (index search)
  ⇒ optimize for random access performance
- use RAID 10 for database, RAID 1 for write ahead log
- separate disks for database, write ahead log and operating system

Volume	RAID level	Number of disks
operating system	1	2
write ahead log	1	2
database	10	30
hot spare		2

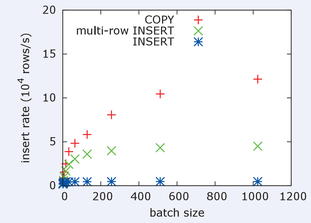


Performance measurements

Write performance

- use Archive Engine tests to measure performance (same code as used during operation)
- write rate depends on database configuration (see table)
- benchmarks written in Perl suggest that performance can be improved by using more efficient commands to write the data (e. g. `COPY`)

Foreign key constraints	Partitioning	Rows/s
no	no	18289
yes	no	6075
yes	yes	3865

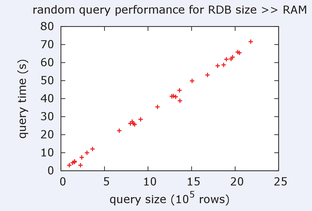


Read performance

- tuning operating system read ahead can improve read performance drastically
- benchmarking RDB read performance is complex
- caching is very important
- up to now: measuring execution time of single queries (90% of all query times below 250 ms)

⇒ partitioning increases performance

- multi-threaded test issuing parallel queries is under development

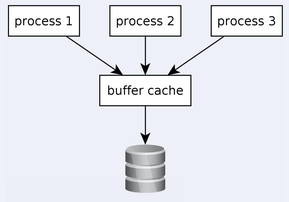


Memory configuration

- most queries ask for data from the last days

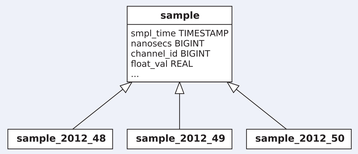
⇒ **RAM should be large enough to hold data of the last days including relevant indexes**

- PostgreSQL spawns a dedicated process for each client
- read and write operations are performed against a common memory region (“buffer cache”)
- **increase the size of the buffer cache to 25% of the RAM** for optimal performance (default is three orders of magnitude lower!)
- on Linux the “maximum shared memory segment size” of the kernel has to be increased accordingly



Database partitioning

- if `sample` table becomes much bigger than RAM query times can escalate and maintenance becomes difficult
- **improve performance by splitting table into smaller pieces**
- weekly partitioning for S-DALINAC
- PostgreSQL does not provide built-in functions for partitioning



⇒ implemented partitioning using server-side programming features
- use table inheritance to combine weekly subtables into one table
- trigger function redirects INSERTs into appropriate partition
- new partitions are created automatically

⇒ included in Control System Studio distribution

Summary

To achieve high performance while at the same time ensuring reliability

- use battery backup for write cache
- use a enough main memory to cache important data
- tune operating system carefully
- configure RDB back-end carefully
- partition big tables
- if very high write performance is needed remove foreign key constraints
- in the future the performance of the Archive Engine might be improved by using more efficient write commands

*Work supported by DFG through CRC 634

†konrad@ikp.tu-darmstadt.de

DFG

SFB 634