CONTROL SYSTEM STUDIO ARCHIVER WITH PostgreSQL BACK-END: OPTIMIZING PERFORMANCE AND RELIABILITY FOR A PRODUCTION ENVIRONMENT*

M. Konrad†, C. Burandt, J. Enders, N. Pietralla
TU Darmstadt, Darmstadt, Germany

Abstract

Archiving systems based on relational databases provide a higher flexibility with regard to data retrieval and analysis than the traditional EPICS Channel Archiver. On the other hand they can suffer from poor performance compared to the Channel Archiver for simple linear data retrieval operations. However, careful tuning of the database management system's configuration can lead to major performance improvements. Special care must be taken to ensure data integrity following power outages or hardware failures.

This contribution describes the hardware and software configuration of an archiving system used in the production environment at the S-DALINAC. It covers performance and reliability aspects of the hardware as well as tuning of the Linux operating system and the PostgreSQL server.

INTRODUCTION

Archiving systems that collect and store data from an accelerator control system play an important role in modern control system environments. They can e. g. help to identify broken hardware, decreasing performance of components or wrong operation of systems by an operator. Thus archiving systems with high availability are demanded. Since wrong or inconsistent data from an archiving system can lead to wrong decisions by the operators it is also important to ensure data integrity. On the other hand data from the archive should be (read-only) accessible from many different applications ranging from simple spreadsheet applications to mathematical software for sophisticated data analysis. These requirements make relational databases (RDBs) a reasonable choice as a storage back-end.

The archiving system of the S-DALINAC is based on the Control System Studio (CSS) Archive Engine. This service collects data from control system channels and writes it to different RDB back-ends (MySQL, Oracle or PostgreSQL). PostgreSQL 9.1 running on Debian Linux has been chosen as a back-end because it is a powerful open-source RDB solution that comes without license fees.

Archiving systems usually have to manage a growing amount of data making read and write performance an important issue. In the following we describe how to tune an archiving system for maximum performance while ensuring data integrity.

* Work supported by DFG through CRC 634
† konrad@ikp.tu-darmstadt.de

DISK PERFORMANCE AND RELIABILITY

Disk access is the most important factor for the performance of a database system that deals with much more data than can be stored in main memory. While modern disk drives can deliver data rates of more than 100 MB/s for sequential reads and writes they can be very slow when they have to deal with random access patterns. Heavy random access workload can occur if data is read from an archive by multiple clients using indexes. Thus database disks have to be optimized for high random access performance.

Write Caching

To improve access time all recent hard disk drives use integrated write-back caches and techniques to speed up disk access by optimizing the order in which read and write commands are executed to reduce the amount of drive-head movement. To insure data integrity database management systems (DBMS) flush operating system write caches after each database commit to make sure all relevant data has been stored on disk before marking the transaction as complete. Write-back caches on the disk drive itself are not flushed and thereby can lead to data loss and corruption of the database files in case of a loss of power. To avoid this, disk write caches have to be disabled for a database server.

Unfortunately switching off this cache severely reduces write performance. Part of this performance loss can be regained by using an appropriate operating system I/O scheduler that sorts the commands before sending them to the disk. But frequent flushing limits the possibilities for optimizations. A solution that provides better performance is using a hardware RAID controller with a battery backup unit. These controllers contain a non-volatile write-back cache that is powered by a battery in case of power outages. Data still in this cache when a loss of power occurs is written to disk after power resumes. Note that disk write caches often have to be disabled explicitly using the configuration tool of the RAID controller.

In contrast to hard disks, solid-state drives provide very high random access performance, but usually their write-back cache is volatile and cannot be disabled making them a bad choice if data integrity is a concern. Write caching is also an issue if a DBMS runs in a virtual machine because reliably flushing data to disk is impossible with most of today's virtualization products. Thus a physical machine is required for the RDB back-end of an archiving system.

ISBN 978-3-95450-124-3

Table 1: Characteristics of RAIDs Consisting of N Disks with Different Organization (Assuming Pairs of Two Drives for RAID 10)

RAID level	Space efficiency (disks)	Fault tolerance (disks)	I/O impact read	I/O impact write
0	N	0	1	1
1	1	$N-1$	1	N
5	$N-1$	1	1	4
6	$N-2$	2	1	6
10	$N/2$	1 per pair	1	2

The CSS Archive Engine on the other hand can safely be run on a virtual machine.

RAID Organization

Redundant arrays of independent disks (RAID) are a way to improve disk performance and reliability [1]. Table 1 gives an overview of the characteristics of the most common RAID levels. RAID 0 increases performance by striping data over multiple disks. It is unsuited for most archiving purposes because data is lost if a single disk fails. RAID 1 mirrors data to multiple disks to improve reliability. The size of the array is thereby limited by the size of a single disk which is insufficient for big archives. RAID levels 5 and 6 store additional parity information to make the array tolerant against failures of one or two disks, respectively, while still providing high space efficiency. The disadvantage of these RAID levels is that modifying a single block leads to 4 or 6 I/O operations, respectively. This can significantly reduce performance of archiving systems that constantly archive thousands of channels at high data rates. RAID 10 stripes data over pairs of mirrored disks and thereby provides high reliability along with good read and write performance, even if a drive has failed. Therefore RAID 10 has been chosen for the S-DALINAC's archiving system. If space efficiency is more important than write speed (e. g. for long-term archiving systems) RAID 6 might also be an option. Regardless of the RAID level, modern RAID controllers distribute random reads over all N disks resulting in an N times higher random-read performance than a single disk can deliver.

The performance of a RAID can be improved by using faster spinning disks or a higher number of spindles. For the S-DALINAC archiving system a higher number of slower disks (10,000 rpm) turned out to be more economic. Direct-attached storage has been favored over network-attached storage for its lower latency. For systems that need more storage than can be directly attached to a single machine storage area network technology might be an option.

Separate RAID volumes are used for the database itself, the write ahead log (WAL) of the database, and the operating system (see Table 2). This makes sure that operating system activity does not slow down database access. Separate disks are used for the WAL because this file is written very heavily while data is inserted into the database. Since the write pattern of the WAL is sequential this almost completely avoids disk seek times. In addition to that separate volumes simplify identification of bottlenecks as well as continuous load monitoring.

Table 2: Disk Organization of the S-DALINAC RDB Server

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

Operating system configuration can also have a severe impact on the performance of an archiving back-end. On machines using a hardware RAID the `noop` operating system I/O scheduler should be used to avoid false optimization. In addition to that increasing read-ahead size can be necessary to exploit full sequential read data rates.

MEMORY CONFIGURATION

The database back-end can answer queries a lot faster if it does not need to read the data from disk. With 128 GB the size of the system main memory has been chosen to be large enough to hold the data of the last days including the relevant indexes.

PostgreSQL spawns a dedicated process for each database connection. All these processes perform read and write operations against a common memory region called buffer cache. Improper configuration of this memory can severely reduce read and write performance. Note that this parameter is configured for maximum compatibility instead of maximum performance by default. For optimal performance we had to increase the buffer cache size by three orders of magnitude to roughly 25% of the main memory. This still leaves the operating system enough RAM for caching reads. On Linux systems it is necessary to increase the maximum shared memory segment size of the kernel accordingly.

PARTITIONING

If the size of the `sample` table grows much larger than physical memory and even its index stops fitting into memory query times can escalate. One way to improve performance is to split data into several partitions that each fit into main memory. At the S-DALINAC most database queries ask for data from the last three or four days making partitioning over time with a partition size of a week a reasonable choice. When executing queries the DBMS can skip partitions that are outside the requested range. For most queries only one or two partitions have to be considered.

In contrast to enterprise databases like Oracle or MS SQL, PostgreSQL does not provide built-in functions for partitioning. Using PostgreSQL's extensive server-side

ISBN 978-3-95450-124-3

Table 3: Write Rates Obtained with Different Configuration of the Archiver Database Table

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

Table 4: Hardware of the S-DALINAC PostgreSQL Server

Main-board	Supermicro X9DRi-F
CPU	2×Intel Xeon E5-2643 @3.3 GHz
Main memory	128 GB DDR3 registered ECC
RAID controller	Adaptec 6805 SAS2
Disks	36×Toshiba MBF230LRC 300 GB

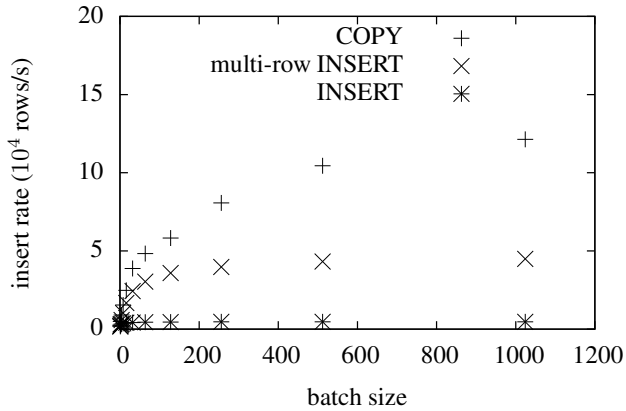

Figure 1: Write performance obtained with different write schemes and batch sizes.

programming features a partitioning solution tailored to the particular needs of the CSS archiver has been developed. It uses table inheritance to combine the data of weekly subtables into one table. A trigger function redirects `INSERT`s to the appropriate partition. New partitions are added automatically. The partitioning feature only affects the database side of the archiver and has been included into the Control System Studio distribution.

While partitioning can significantly speed up read access it introduces checking of additional constraints during `INSERT`s. This results in higher CPU load and can limit maximum write rates.

PERFORMANCE MEASUREMENTS

Write Performance

Before the archiving system has been put into operation, write performance has been measured using the `RDBArchiveWriterTest.testWriteSpeedDouble()` test included in the CSS Archive Engine code. This test is very close to real archiving load because it uses the same write scheme and the same code on the client side. It has been used to verify the success of the tuning steps described in this paper. Typical write rates obtained with this test are presented in Table 3. An overview of the hardware used for this benchmark is given in Table 4.

Write performance strongly depends on the configuration of the database. Adding foreign keys between tables to ensure referential integrity forces the DBMS to check each row against all foreign keys before an insert operation can be performed.

Performance of database writes also strongly depends on the command that is used to write the data. The current implementation of CSS Archive Engine improves performance by submitting `INSERT`s to the database server in batches before committing the data. Benchmarks implemented in Perl confirm this performance gain but they also show that other write techniques can provide even better performance. Figure 1 clearly shows that much higher write rates can be obtained by using multi-row `INSERT`s or the PostgreSQL-specific `COPY` command. Although performance might depend on the library used for database access, results suggest that performance might be improved in the future by using more efficient write commands.

Read Performance

Benchmarking read performance is more complex than benchmarking write access since synthetic benchmarks can lead to unrealistic caching of relevant data. In contrast to the write case multiple clients can issue queries at the same time. Up to now read performance has only been determined by measuring execution times of single queries. These results confirm that partitioning improves read performance. A more realistic benchmark is under development.

SUMMARY

A battery backed up write cache as well as careful configuration of the PostgreSQL back-end are important to achieve high performance while at the same time ensuring data integrity. Write performance can be improved by removing foreign key constraints while at the same time losing referential integrity checks. Another way to improve performance in the future might be to use more efficient write patterns like multi-row `INSERT`s or the `COPY` command. Read performance can be significantly improved by partitioning the `sample` table. As soon as solid-state drives are reliable enough they can also help to further improve performance.

ACKNOWLEDGEMENTS

We thank Kay Kasemir (ORNL) and Lana Abadie (ITER) for fruitful discussions and their valuable help in understanding the implementation of the Archive Engine.

REFERENCES

[1] D. A. Patterson et. al., “A case for redundant arrays of inexpensive disks (RAID)”. Proceedings of SIGMOD 1988, Chicago, IL, USA, p. 109–116.

ISBN 978-3-95450-124-3