THE HIGH PERFORMANCE DATABASE ARCHIVER FOR THE LHC EXPERIMENTS

M. Gonzalez-Berges, CERN, Geneva, Switzerland

Abstract

Each of the Large Hadron Collider (LHC) experiments will be controlled by a large distributed system built with the Supervisory Control and Data Acquisition (SCADA) tool Prozeßvisualisierungs- und Steuerungsystem (PVSS) [1]. There will be in the order of 150 computers and one million input/output parameters per experiment. The values read from the hardware, the alarms generated and the user actions will be archived for the later physics analysis, the operation and the debugging of the control system itself. Although the original PVSS implementation of a database archiver was appropriate for standard industrial use, the performance was not sufficient for the experiments. A collaboration was setup between CERN and ETM, the company that develops PVSS. Changes in the architecture and several optimizations were made and tested in a system of a comparable size to the final ones. As a result, we have been able to improve the performance by more than one order of magnitude, and what is more important, we now have a scalable architecture based on the Oracle clustering technology (Real Application Cluster, RAC). This architecture can deal with the requirements for insertion rate, data querying and manageability of the high volume of data, e.g. an insertion rate of > 150,000 changes/s was achieved with a 6 node RAC cluster.

INTRODUCTION

The control system for an LHC experiment is composed of two parts: the Front End (FE) and the Back End (BE). The Front End is closer to the detector equipment and is responsible for tasks such as data acquisition, filtering, real time control loops and interlocks. The Back End is built on top of the FE and its main functions are alarm handling, graphical user interfaces, hierarchical operation, interface to external systems and data archiving. The BE runs on a set of around 150 Windows and/or Linux computers connected together in a distributed system. The application has been developed using the SCADA tool PVSS and the JCOP Framework [2].

The data archived from the 150 systems will go to a central database server. The experiments have set a requirement of storing simultaneously 1000 changes/s per system. This rate will not be required continuously but rather as a peak during short periods of time. However, having a system that can handle this rate continuously and that scales with the number of clients has some advantages. Firstly, future system upgrades where more systems are added or higher rates are required can be handled. Secondly, if the archiver is optimized, more resources will be available in the database server for other tasks (e.g. queries).

ARCHIVING IN PVSS

One of the main features of PVSS is that it is device oriented. A user can define a device type that can then be easily instantiated. The elements of the device type or the device instances (equivalent to the attributes in object oriented terminology) can be configured to be archived either to file or to a database. These elements can represent data read from the equipment (e.g. temperature), commands (e.g. voltage set point) sent to the hardware or any parameter internal to the application (e.g. memory left in the computer). This is shown in Figure 1.

Although PVSS works in terms of devices, the fact that the elements are stored individually makes it appropriate to use a relational database to store the changes.

It is also possible to store the alarms associated with the elements. The work done so far concentrates on the storage of the values of the elements rather than the alarms and this is presented in the rest of the paper.

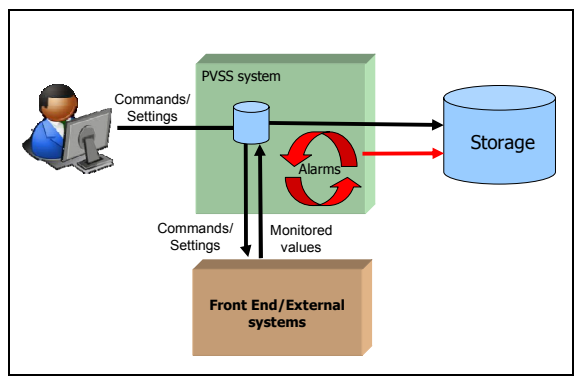

Figure 1. Overview of the PVSS archiving.

PVSS SYSTEM (CLIENT SIDE)

Initially we tested a setup with a single PVSS system (i.e. one computer) storing data to an Oracle server. A continuous rate of around 100 changes per second could be reached before the system started to buffer data. The bottleneck was in the Archive Manager (see Fig. 2) that was taking most of the CPU. This PVSS process is responsible for sending values to the database, after smoothing has been applied by the Data Manager.

We looked into the way that data is sent to the database. Each individual change in the PVSS system was sent without any grouping. In addition, a generic interface to the database was used to have a database vendor

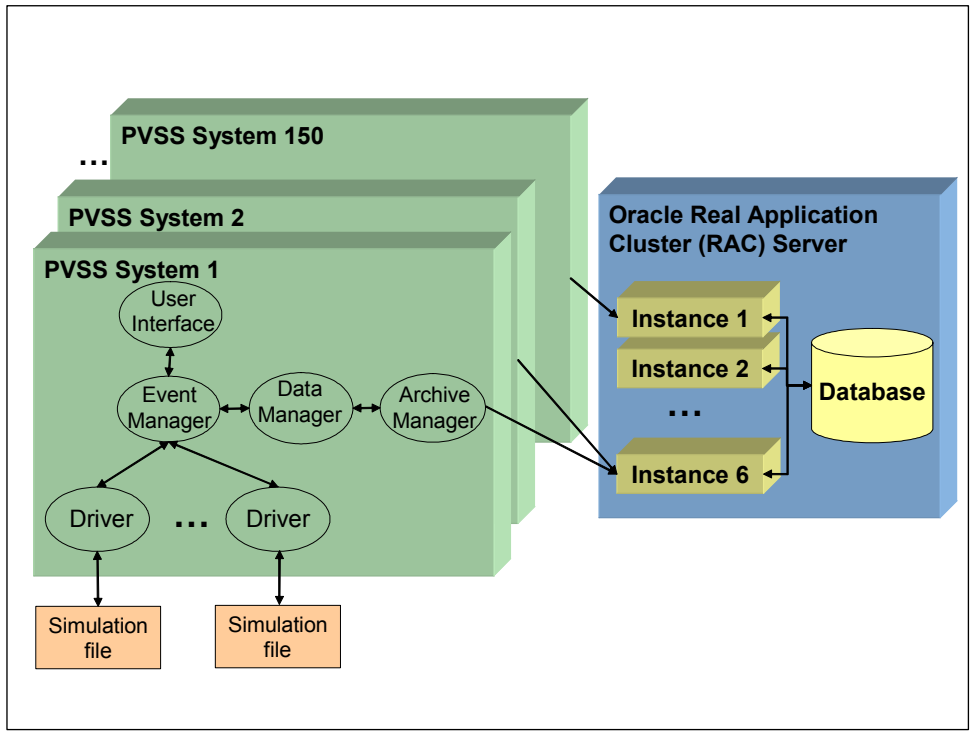

Figure 2. PVSS clients connected to the Oracle server.

independent solution. This prevented us from using many of the optimizations available in Oracle.

To improve this we chose to use the Oracle native client libraries called OCCI (Oracle C++ Call Interface) for the performance improvements. The main gain was to use bulk insertion: value changes are first stored in a block in the memory of the client and afterward transferred to the database server. The possibility to set a timeout to send blocks even if they are not full was also included. Other advantages of using OCCI were the direct use of floating point numbers and better connection handling.

Performance was measured again after the changes mentioned above were implemented. A continuous rate of 2000 changes/s, generated with a standard PC, was archived for several hours. This covered the expected peak rate for a single PVSS system, which will only be necessary for some minutes. The generic database interface was kept so that the archiver can still run with other database types, although obviously with reduced performance.

DATABASE SERVER

After data is sent to the database server, it has to be inserted into the appropriate history table. This is done with code running in the server written in PL/SQL.

After the optimizations in the client code, we tested with a group of clients each with a rate of 1000 changes/s. The server could handle around 20-30 clients depending on the configuration we used (e.g. block size for grouping value changes). This was clearly below the 150 systems required for an LHC experiment. We had two possibilities to increase the performance of the database server: buy a better server or use the clustering technology of Oracle. We chose the second solution because of the cost (PCs could be used), the redundancy (the server can continue working even if some nodes fail) and the flexibility (it is possible to upgrade the server by adding more nodes).

Real Application Cluster (RAC) [4] is the Oracle technology for building clustered servers. It was first introduced in Oracle version 9i, and it is now a mature technology. Two parts can be distinguished in a RAC server (Fig. 2). The processes and memory structures that are distributed across the nodes in the server are called Oracle instances. The data structures stored in files form the Oracle database. The instances are connected between themselves and with the database with a high speed network.

When we started to test in a server with 2 nodes we saw that although a higher insertion rate could be achieved, it was far from double the rate with a single node. Analyzing the Oracle statistics reports we could see that the nodes were taking exclusive locks in the history tables for long periods, so they were interfering with each other. A first solution was to reduce the time the locks lasted by using a technique called direct path for inserting into the Oracle tables. With this method, while insertion takes place any integrity constraints are disabled and indexes are only updated once the insertion finishes. Direct path gave a reasonable result for the 2 node server and we could handle around 50 clients with a rate of 1000 changes/s.

The next step was to move to a server that could handle the requirements set by our users. We moved to a 6 node

server and we had 150 PVSS clients with a rate of 1000 changes/s. The system was not scaling properly because the nodes had to communicate a lot to keep the coherency of the tables they were inserting into. Then we took advantage of the fact that each client inserts its own identifier in the tables to partition them according to that identifier. In this way each client inserts data in its own partition and they don't interfere with each other. This finally made the application scalable on the server side (Fig. 3). Now it is possible to add more nodes to the server if a higher performance is required. The measurements after these changes are presented in the next paragraph.

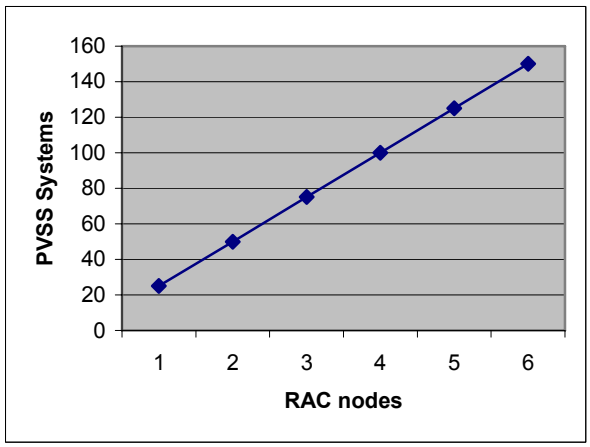

Figure 3. Scalability of the RAC server.

CURRENT PERFORMANCE

A system similar to the one in the final control system was setup to measure the performance that could be achieved with the above changes. On the one hand, there was a PVSS distributed system with 150 computers running Scientific Linux CERN 3 (SLC 3), 3GHz CPUs and 2 GBytes of memory. Each system generated a load of 1000 value changes per second. On the other hand, there was an Oracle RAC server with 6 nodes. Each node was an Intel Xeon processor with 4 GBytes of memory running Red Hat Enterprise Linux 4. The version of Oracle was 10g release 2. The server was able to handle the load. However, when the current tablespace (i.e. the files where Oracle stores data) got full, and a new one had to be created the server slowed down. This seems to be a limitation of Oracle. A very straight forward solution is to precreate the tablespaces or create them with a background job well before they are needed. We are currently implementing this solution.

NEXT STEPS

In addition to the precreation of tablespaces, we plan to include the possibility to buffer data to the local disk when the database server is not reachable. Once the server becomes available the locally stored data would be sent to it.

The work described so far in this paper covers the insertion of value changes. As explained before, the alarms are also stored by the archiver. The requirements for the alarms are much lower. Although avalanches of a few minutes have to be considered, there shouldn't be a continuous rate as it would be impossible to diagnose the source of the problem and the alarms would eventually be filtered or disabled. We plan to go through a similar optimization process as we did for the values.

CONCLUSIONS

The basic PVSS database archiving has been optimized and tailored specifically for Oracle RAC servers. This covers the requirements set by the LHC experiments. The changes made to the tool have been done in collaboration with ETM and they are now part of the standard PVSS versions. This has the advantage that CERN doesn't have to maintain it during the next years.

Work continues to solve some minor problems left with the insertion of values and to start with the optimizations of the alarms.

ACKNOWLEDGEMENTS

The work presented in this paper has been the fruit of a collaboration between several CERN groups (Experiments, Controls and Databases) and ETM, the company that produces PVSS. Mainly the following people have been involved from CERN: Eric Grancher, Luca Canali, Nilo Segura, Pior Golonka and Svetozar Kapusta; and from ETM: Ronald Putz and Ewald Sperrer. The author would like to thank also all the other people that have contributed in one way or another to this work.

REFERENCES

[1] PVSS made by ETM professional control AG, Eisenstadt, Austria, http://www.pvss.com.

[2] The Joint COntrols Project (JCOP) Framework. http://cern.ch/itcobe/Projects/Framework

[3] E. Grancher. Oracle RAC (Real Application Cluster) application scalability, experience with PVSS and methodology. Computing in High Energy Physics. Victoria BC, Canada, September 2007.

[4] Oracle Real Application Cluster (RAC) technology. http://oracle.com/technology/products/database/clustering.