10th ICALEPCS Int. Conf. on Accelerator & Large Expt. Physics Control Systems. Geneva, 10 - 14 Oct 2005, PO1.056-6 (2005)

SCALING UP PVSS

P.C.Burkimsher
CERN, Geneva, Switzerland

ABSTRACT

A typical large Industrial Control System may have 5,000 data points. Very large systems sometimes have 50,000 data points. The four experiments at the Large Hadron Collider (LHC) will have between 500,000 and 5,000,000 data points. These experiments have jointly opted to use an industrial SCADA controls toolkit, PVSS, with which to build their experiment control systems. This paper summarises the issues involved in building very large control systems and outlines how PVSS was put through its paces and even modified by ETM, the manufacturer, in order to scale up and meet the requirements of the LHC era. It describes the areas investigated, outlines the results obtained and shows how PVSS is evolving into a High Energy Physics tool *par excellence*.

INTRODUCTION

The High Energy Physics (HEP) experiments at the CERN Large Hadron Collider [1] under the auspices of the Joint Controls Project JCOP [2] conducted an in-depth evaluation of SCADA systems [3] and made a unanimous decision to use the PVSS SCADA toolkit from ETM [4] for building their detector control systems. This decision was taken, in part, due to PVSS' evident scaling capabilities. Nevertheless, JCOP instigated a series of tests entitled "Scaling Up PVSS" to confirm PVSS' capacity to build very large systems.

PVSS ARCHITECTURE

Control systems built with PVSS comprise a collection of communicating independent processes (tasks) known as managers. Figure 1 shows the principal managers in a system.

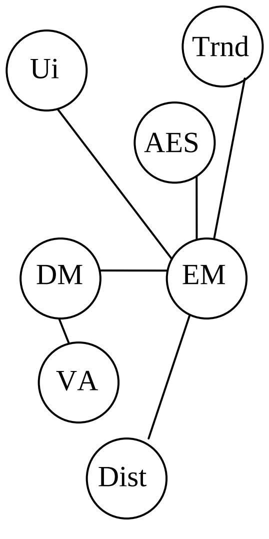

Figure 1. PVSS is Manager based

The Event Manager (EM) is central to PVSS. It holds all data values in volatile memory, responds to notification of data changes and calculates alarm conditions. Values are held in structures called Data Points (DPs).

The Data Manager (DM) is responsible for managing the database where non-volatile data such as system definitions, alert occurrences and latest values are stored.

The Value Archive Manager (VA) looks after long term storage of data point values.

The User Interface (Ui) managers exist in multiple instances for display of the synoptic panels, Alarm Screen (AES), Trend graphs (value against time) etc.

The Distribution Manager (Dist) is responsible for interconnecting an individual PVSS system with other individual systems to build one united, distributed system.

Many other managers are available for performing specific functions (e.g. executing user code) or for interfacing to specific hardware.

WHAT MAKES A SYSTEM SCALABLE?

The ability to master complexity is crucial to a successful large system but needs support from any underlying software package. PVSS offers this support at many levels:

1) PVSS is a device-oriented system. An additional piece of external hardware can be declared at a stroke by creating a new instance of a given type.
2) PVSS is an open system. This means that:

   a. It is possible to build layers of abstract software to provide additional functionality.
   b. Large, complex systems can be generated automatically and accurately from an external database definition.
   c. PVSS can be interfaced with external tools such as the CERN Finite State Machine [5], itself a crucial component for mastering the complexity of running a large physics detector.
3) PVSS is manager based. This means that:
   a. The load of a PVSS application can be easily scattered across many processors and/or many computers.
   b. PVSS is capable of monitoring itself and taking self-healing action.
4) PVSS supports distributed systems meaning that individual PVSS systems of an easily manageable size can be developed. These systems then cooperate as a single, very large, distributed system at run-time, where all of the data in any one system is available to all of the other connected systems in a transparent way.

WHAT MAKES A SYSTEM NOT SCALABLE?

An unprofessional implementation can ruin a good system design. PVSS was tested to search for inbuilt implementation restrictions on string length, number of Data Point Types (DPTs) and Data Point Elements (DPEs), number of managers, number of inter-manager connections etc. At CERN's explicit request, ETM made a fundamental design change to the PVSS distributed system feature, better enabling their implementation to support large systems by removing the need for synchronisation between systems and by supporting dynamic system inclusion.

It can be difficult to establish a single "view" of a large system, especially when there are communications difficulties with other parts of the system! PVSS offers programming hooks to be able to signal such difficulties on-screen. PVSS' default behaviour under adverse network conditions is discussed later in this paper.

A multi-computer system contains many time-of-day clocks which can drift relative to each other. We confirmed that PVSS contains explicit algorithms for coping with data that arrive in the future (!) as well as in the past.

Cooperating systems are difficult to debug because of the irreproducibility of time (order) dependent effects. PVSS contains debugging trace tools but the output from these facilities is complicated to interpret and is intended for expert use only.

CERN identified a security weakness in early tests which was fixed some years ago. Denial of service is a general threat and PVSS must be protected adequately from external, deliberate network perturbation through the appropriate use of firewalls. The current security imperative is to run all production control systems on a standalone network.

Management of large systems is inherently more complicated than management of small systems. Appropriate methods and policies for managing large PVSS systems are still being elaborated. PVSS is able to support transition to new software versions without service interruption through its redundant system feature, though this is unlikely to be used in the CERN experiment control systems.

SCALABILITY TESTS

At the request of the LHC experiments, via JCOP, a wide range of scalability tests and measurements were undertaken covering both functionality and performance.

Connectivity test

The theoretical limit within PVSS allows support of up to 254 component systems in a distributed system. The LHC experiments expect to require of the order of 100 component systems. For testing, we built a hierarchy of PVSS systems containing 5 layers, where each parent system had 3 child systems. It was decided to extend this configuration to 130 systems by adding a partial 6th layer to ensure that there was no hidden limit at 128 systems!

Figure 2 portrays the first 3 levels of the hierarchy where each system was connected directly to its parent and to each of its ancestors all the way up the tree. The cross-platform test included Windows 2000, XP and Red Hat Linux machines. Representative data points were declared in each system: five High-Voltage Crates, each containing 16 boards with 16 channels of 30 float values each, made a total of 38,400 DPEs per system. This figure of ~40K DPEs per system is representative of the experiments' expected system size. Approximately 5 million datapoints (130x40K) were seen to be accessible in the system.

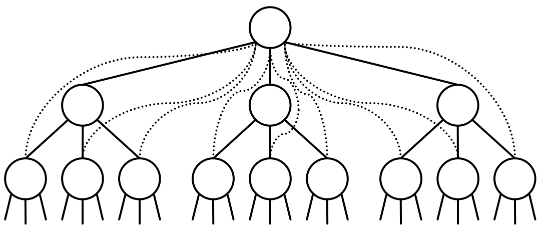
Figure 2. Hierarchy of PVSS systems for the connectivity tests.

Network Perturbations

The question was raised as to how affected PVSS would be by other, unrelated network traffic. This issue is relevant not just in terms of security (i.e. accidental or malicious denial of service) but more mundanely whenever the experiment downloads a large quantity of data. For example, the Alice [1] collaboration expects to load up to 200Mb of experiment configuration data into its PVSS machines. It is envisaged that the PCs will be connected via switches rather than hubs, so basic contention on the wire is discounted. Modern network cards support full duplex operation and hence inbound and outbound traffic are separated. Any delays will only be caused by directed traffic to (or from) a specific machine. TCP-level traffic from a generator application was not able to saturate the network link beyond 73%. We believe the limiting factor here was the speed of the computer clock. Nevertheless, the PVSS UI manager was not affected. A second test was carried out generating PING traffic which, being dealt with lower down in the protocol stack enabled us to achieve a network load of 99%. A scattered UI plotting task was still not affected. Other tests have shown that PVSS as a TCP application is incapable of generating traffic at any rate that might begin to cause problems.

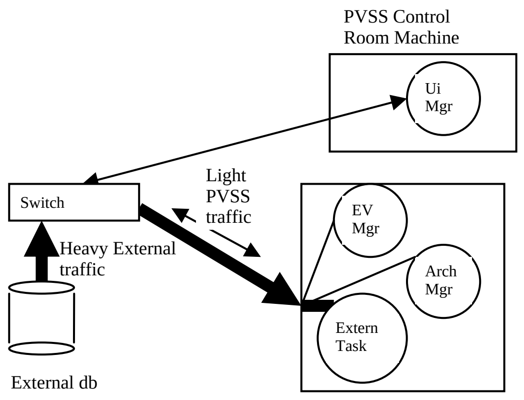

Figure 3. External network traffic scenario

A Long-Term-Test has been run but thus far has merely shown up weaknesses in the CERN environment! Aside from the regrettably unstable power supply, CERN's default Linux settings were seen to hamper PVSS' attempts to recover automatically after a network component failure. This effect has been documented [6].

Overload Handling

A series of tests were conducted to observe sustained PVSS traffic generation and processing in the large system hierarchy shown in Figure 4. We ran one PVSS system per computer, running Windows or Linux. The root node corresponds to a control room computer. The leaf nodes correspond to PVSS systems that will be connected directly to front-end hardware. As before, each system had a direct connection to all of its ancestors with no horizontal (peer) links. 40,000 DPEs with DPTs resembling those of a high-voltage crate, including alert and archiving definitions, were defined. PVSS traffic, in the form of simulated data changes, was generated in the leaf nodes.

For the first test, the PVSS traffic was displayed as a trend of values on the "control room" computer's screen, i.e. on the top node. Trend displays were added one at a time onto the top node's screen, showing data being retrieved from archives on the leaf nodes. It was observed that the additional CPU load and the additional RAM load on the top node were negligible for each new trend. After 48 trend windows had been opened simultaneously the test was stopped. The available screen real-estate was so full of (even iconised!) trends that it was meaningless to continue.

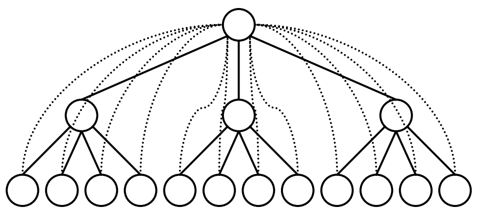
Figure 4. Hierarchy for the Overload tests

The overload test was then repeated, but instead of all the leaf nodes sending trend data to the top node, they were instructed to send their current *alert* information instead. The PVSS traffic generators on each leaf node were set to provoke and clear alerts continuously, offering the potential to create a very high rate of alert traffic. As additional leaf nodes' alerts were added to the top node's display, PVSS' capabilities for overload handling were revealed.

Overall traffic generation was ramped up until the top node could no longer absorb and process data as fast as it was coming in from one of the leaf nodes below. The leaf node eventually cut the connection to the top node. We learned that this "break the connection to a slow data consumer" is a typical PVSS "evasive action" and it offers 3 marked rewards:

5) It frees up the leaf node to continue its own processing without choking.
6) It reduces traffic on the link.
7) It frees up the top node from what is most likely an unwanted cascade of data.

Further testing confirmed that the bottleneck was occurring in the top node, not in the leaf node. The top node was, of course, connected to many leaf nodes and had to deal with *all* of their traffic. The "top node" in this test was a dual-CPU machine with hyper-threading. Its four (apparent) CPUs shared the load.

Running the alert display screen (AES) was seen to be very demanding on CPU resources. CPU load was seen to be caused by the AES display task and the system's Event Manager task - which supplies the information to the display task.

Traffic generation rates were reduced slightly, to just below the rate at which the leaf system would cut the connection to the top node, but still at a very high rate such that the top node display could not quite keep up, i.e. could not update the display quickly enough. Under these circumstances, a prominent "alert avalanche" warning message appears on the alert display screen and screen update is paused. Once the torrent of incoming alerts subsides, screen update is resumed. The benefit of this behaviour is that under avalanche conditions, the load on the top node is temporarily reduced. The top node is then able to absorb (although not display) the alerts that are being thrown at it. In turn the leaf node (the alert source) does not have to take the drastic step of cutting the connection to the top node.

The same "evasive action" behaviour was also observed under a completely different scenario. When a new PVSS system joins the distributed system cluster, it transmits its DP namespace definitions to its partners, in our case, to its ancestors. It then transmits all of its current, active alerts to any system having registered an interest (i.e. any system which has opened an alert screen and has selected to display this node.) In our test, the top node was deliberately made to be very busy before the new leaf node joined the system. The tardy nature with which the top node absorbed the new information caused the leaf node to overload and promptly cut its link to the top node. This behaviour, although somewhat surprising when it first happened was correct in that we had simulated a very busy system and then tried to make it even busier. In effect, PVSS rejected the entry of the additional system.

The test scenario revealed yet another interesting aspect of PVSS behaviour. PVSS' own monitoring task called PMON notices when a task dies. PMON is configured by default to automatically restart a dead task that it believes should be running. This behaviour is very desirable and we would normally expect a task to be configured in this way. Unfortunately when a task dies because of PVSS taking evasive action, then we certainly do *not* want PMON immediately to restart the task.

What we are seeing is a conflict of requirements:

- For resilience, the EM needs to take evasive action and kill an offending task.
- Yet for resilience, PMON is eager to restart any task that dies unexpectedly.

Clearly, if a task dies because of a crash we would want it to be restarted immediately. On the other hand if it has been killed by the EM because of an overload then an immediate restart may be way, way too soon. Only when the overload crisis has passed should the offending task be restarted. There is a problem in that in the current implementation, PMON unfortunately always attempts an *immediate* restart. We have requested ETM to make the restart interval configurable (on a per task basis).

There is a second undesirable consequence of PMON's behaviour. In order to prevent infinite looping (of the start/crash/restart/crash variety) PMON will eventually give up trying to restart a task. Superficially this is desirable as infinite loops are usually unwelcome! Unfortunately once PMON has given up, a manual restart of the offending task will be necessary. The "self-healing" nature of the system will have been completely lost. In a small system this may not matter, but in a large distributed system, it is considered to be very important. ETM have been informed and are considering the suggestion that PMON never gives up, but resumes trying to restart the offending task after some *configurable* interval.

When a link to a participating leaf system is broken, that system's alerts are removed from the top display, as one would wish. Unfortunately there is no indication on a *trend* display that a link has broken. ETM have been informed. When PVSS takes evasive action, the occurrence is recorded in the project logfile. However, PVSS' logfile viewer is unlikely to be running all the time. It is heavy in CPU consumption and in a large system, there are many such logfiles to be monitored. We therefore deem it appropriate to recommend to application designers that the top node ("Control Room machine") permanently display the interconnection status of the distributed system (even if only as an array of red and green pixels!) and/or should raise an alert when a connection is lost.

The Evasive Action feature of the Event Manager was seen to protect recorded data from random gaps. A loquacious data source would be disconnected completely before any data were intermittently lost.

The top node coped with slightly more than 200 new alert transitions (came or went) every second. Further information regarding exact traffic rates and other details are contained in [7].

Load Balancing

A PVSS system is made up of a set of co-operating processes (tasks) which run with simulated concurrency on a machine with a single processor. We investigated how explicit allocation of tasks ("scattering of processes") to explicit machines affected performance as defined by the sustained number of datapoint changes per second that the system could support. We investigated how PVSS is able to profit from dual-CPU machines and run with real (as opposed to simulated) concurrency. Performance was measured under a variety of different simulated run-time scenarios, with and without alert traffic, display and archiving. Details of the individual tests are contained in [8]. Here we present the major conclusions, when compared to running the entire system on just one, single processor machine.

First, we observed that "whole system on one machine" configuration is CPU-bound. The Event Manager (EM) is a heavy user of CPU. Scattering the EM to another machine improves performance considerably, in our test by 75%. This result was not entirely expected, but it was clear that the additional CPU benefit really did outweigh the additional communications and network overhead of scattering the EM.

Scattering another important task, the Data Manager (DM) on its own is also beneficial, but less so than scattering the EM alone. Leaving the EM behind on the CPU-bound machine only permitted an overall improvement of 25%. Scattering both the EM and DM to the same "other" machine caused that other machine to become CPU-bound. Improvement stayed at 25%.

Scattering the DM and the value archiving tasks together to the "other" machine but leaving the EM on the main machine, showed a healthy performance improvement of 56%. Scattering just the archive tasks (i.e. making them separate from the DM) reduced the observed improvement back to 25%.

The system could sustain a higher average rate of changes if the arrival pattern of the changes was steady rather than if the changes arrived in bursts. Sudden bursts of changes fill up task queues and provoke evasive action behaviour.

The Alert Event display screen (AES) can use up to 50% of a machine's available CPU. Scattering the AES to a quiet CPU helps, though each additional AES does still place an extra load on the EM. We recommend either pressing the AES "stop" button when screen update is not needed or closing the AES window. Running many AES' at once should be discouraged.

We attempted to investigate PVSS' ability to take advantage of the dual-CPU capability of some machines by first scattering the EM (only) to such a machine. Our machine supported Intel Hyperthreading and Windows XP believed that the machine actually had 4 processors. XP allows you to indicate which CPUs a process is allowed to run on (the process's "affinity"). We first observed that overall system performance was the same irrespective of whether the machine was hard-booted into single CPU mode or affinity was selected for just one CPU in 4-CPU mode. With affinity set to two, the EM did use both processors but neither CPU registered 100% usage and the system throughput remained the same at about 4,000 datapoint changes per second without alerts and 600 with alerts. With very smooth traffic arrival rates we managed 5,000 sustained changes per second. Other results confirmed that the system was no longer CPU-bound and that the overall system throughput was limited by I/O. Although we tried to overload the scattered EM, this was not possible as it took evasive action every time. This is a very strong feature of the software design!

To recap, scattering the EM is usually beneficial because this task is CPU hungry. We observed that an individual EM itself can in principle use about two CPUs, but in practice our example quickly became disk bound. Nevertheless, if a complete PVSS system must run in a single machine, multiple CPUs are of benefit as the different tasks can be spread out. Working solely on our "fast" machine, we observed that single CPU affinity could support 600 changes per second (with alerts) whereas all 4 CPUs working together supported 900 changes per second, with CPU cycles to spare.

SUMMARY AND CONCLUSIONS

The Scaling UP PVSS Project has investigated many more functionality scenarios and performance tests than have been described here. Results are available on the web [9] and we have learned much valuable information concerning emergent behaviour in this environment. As a direct result of feedback from these tests, ETM have made their already very resilient product even more so. PVSS is showing itself to be an exceptionally stable, capable tool, ideally suited to building very large systems.

REFERENCES

[1] Large Hadron Collider and Experiments at CERN: http://cern.ch/lhc, http://atlas.ch, http://cms.cern.ch, http://lhcb.cern.ch, http://aliceinfo.cern.ch
[2] Joint Controls Project: http://cern.ch/itco/Projects-Services/JCOP/
[3] SCADA Product Evaluation, A. Daneels, W. Salter, CERN, CH-1211 Geneva 23.
[4] ETM Professional Control AG : http://www.etm.at/index_e.asp
[5] JCOP Framework Project, Finite State Machine Component, in http://cern.ch/itcobe/Projects/Framework/Documentation/JointPVSSandJCOPFwCourseScript.pdf
[6] Linux Hostname Caching trap: http://cern.ch/itcobe/Services/Pvss/FAQ/Questions/SupportMaterial/050816LinuxHostnameCacheingTrap.pdf
[7] Test 35. Investigate behaviour when top node displays alerts from many child nodes: http://cern.ch/itcobe/Projects/ScalingUpPVSS/Documents/test35ResultsWriteUp.pdf
[8] Test 45. Task Allocation Investigations: http://cern.ch/itcobe/Projects/ScalingUpPVSS/Documents/test45writeup.pdf
[9] Scaling Up PVSS Project Home Page: http://cern.ch/itcobe/Projects/ScalingUpPVSS/welcome.html