July 14, 2011

Alarm System tests

Motivation

KEK is interested in investigating an alarm setup with 50000 trigger PVs, similar to the existing KEKB alarm system.

Executive Summary

It is possible to use the Best Ever Alarm System Toolkit (BEAST) with 50000 PVs. Assuming some reasonable constraints on the number of active alarms, it performs fine. When the number of active alarms or alarm system transitions goes well beyond anything that operators can handle, it degrades in a sensible way without crashing.

What is an "Alarm"?

In the CSS alarm system, each alarm is supposed to be meaningful, requiring an operator action. An alarm is no status display that operators may ignore. An alarm is no interlock with automated control system response. Each alarm *requires an operator to react* because the control system cannot automatically resolve an issue.

Each alarm must include operator guidance (informational text, links to related displays). Operators in turn are expected to react to each alarm.

The 50000 inputs of the KEKB alarm system do not necessarily represent 50000 individual alarms. How long would operators need to react to 50000 alarms?

When translating from the KEKB alarm system to BEAST, one might not directly import the 50000 PVs as alarm triggers.The top-level KEKB alarm display has only 10 indicators. In fact, one could consider the KEKB setup as a system with just 10 alarms: One for vacuum, one for RF etc., where each alarm has many possible internal triggers. For example, a vacuum excursion can cause hundreds of individual vacuum PVs to indicate an elevated pressure. This does not translate into 100 alarms. It is one vacuum alarm for some area.

Creating good alarm system triggers requires conditioning of raw signals. For example, soft IOC logic can be used to summarize individual pressure PVs by area.

Nevertheless, a setup with 50000 potential alarms directly into BEAST is an interesting performance test. It might even be reasonable to start a CSS alarm setup by importing the KEKB alarms into a suitable hierarchy. In the alarm table view, 100+ alarms from individual pressure sensors are not helpful to operators. In the alarm tree view, however, these 100 alarms could appear within a hierarchy of Vacuum/Area/Subsection, efficiently indicating to operators that there is a sizable vacuum issue in some subsection of some area. Operators would not necessarily investigate each individual vacuum pressure, but use the alarm system to determine that there is one "big" alarm: A leak in some area of the machine.

Test Infrastructure

All tests were executed on a single PC, guest-pc16.kek.jp.
CPU: 3GHz Intel Core2 Duo, 4GB RAM
OS: Windows 7, 32 bit
Relational database: PostgreSQL 9.0, with constraints enabled for the alarm tables
JMS: Apache Activemq-5.5.0
Java: Sun/Oracle SDK 1.6.0_26

Test Setup

A soft IOC executed this type of EPICS database:

```
# In total, 50000 duplicates of the following:
record(calc, "Ramp00000")
{
    field(INPA, "Ramp00000")
    field(CALC, "A<50000?A+1:0")
    field(SCAN, ".2 second")
    field(HOPR, "50000")
    field(FLNK, "Alarm00000")
}
record(calc, "Alarm00000")
{
    field(INPA, "Ramp00000")
    field(CALC, "A>=0&&A<=200")
    field(HIGH, "1")
    field(HSV , "MINOR")
}
```

A “Ramp…” record counts to 50000 in a saw-tooth fashion, and an “Alarm…” record generates an alarm. The first 200 alarm records trigger for ramp values of 0-200 like “Alarm00000” shown above. The next 200 records alarm for values of 200-400, and so on[1].

Every 40 seconds, another set of 200 alarms fires, while the previously active set of 200 alarms clears.

On average, there are thus 5 alarms per second, but they are batched as 200 alarms every 40 seconds. When also considering the 200 alarms that clear every 40 seconds, there are on average 10 alarm changes per second, but batched as 400 alarm transitions every 40 seconds.

The alarm system was configured to monitor the “Alarm…” PVs arranged into 250 areas: First area for Alarm0000 to Alarm00199, next area for Alarm00200 to Alarm0399 and so on.

[1] To reduce the number of records, a setup with a single “Ramp” record being monitored by 50000 “Alarm…” records using “…CP” inputs was attempted. It failed with errors “rngBufPut overflow in scanOnce”. A long FLNK chain from Ramp to Alarm00000 via Alarm00001 and so on to Alarm49999 caused the soft IOC to simply crash.

In addition to this batched alarm traffic, PVs “sim://sine”, “sim://ramp” and “sim://noise” were added which provide a rather constant stream of one or two alarm system transitions per second.

In total, there were 50003 inputs to the alarm system.

Original Problems

The alarm system as it has been used at the SNS with about 400 alarm inputs was unable to handle the test setup. While the alarm server was OK, the user interface was overloaded.

Using JProfiler, the overall alarm system source code was optimized in many areas. Data base access was updated to re-use more SQL statements, the GUI updates were optimized to reduce the number of screen redraws.

Alarm Tree

The Alarm “Tree” display was extremely slow. Even with a reduced alarm configuration of only 1000 alarms in groups of 50 per “Area”, opening the sub-tree for such an area took 3 seconds, freezing the CSS user interface for that time. With the full test setup, opening sub trees could take 10 seconds.

The problem was traced to a shortcoming in the SWT tree widget. When configured to support multi-element selection, it is at this time extremely slow:
https://bugs.eclipse.org/bugs/show_bug.cgi?id=259141.

By only supporting single selections, i.e. in the tree view one can now only acknowledge one PV, system or area at a time, the alarm tree display is very responsive, even with 50000 alarm entries.

In the alarm table, it is still possible to select multiple alarms at a time to acknowledge them. At least for the way the alarm system is used at SNS, the restriction to single-element selections in the alarm tree is of no consequence.

Via a preference setting, it is now possible to enable a workaround to the SWT tree widget problem. This allows multiple selections in the alarm tree with a performance that is much better, but for 50000 elements in the alarm configuration the tree GUI would still be a bit sluggish: Scrolling, opening sections of the tree can cause delays of maybe half a second, so this option was not used in the following.

Alarm Table

The alarm table displays current alarms sorted by name, time or severity. It allows selection of alarms by name or description using patterns, and provides detail about the alarm state: Current severity of the PV, value that originally caused the alarm.

The computational effort for this is roughly linear with the number of alarms. Eventually, the alarm table response to scrolling, selecting items, redraws in response to alarm system updates suffers. There is no hard cutoff, the change is subtle and individual perception will vary. On the test system 5000 alarms in the alarm table caused a noticeable delay in the user interface response.

The alarm table code now supports a configurable limit for the number of displayed alarms. For the following it was set to 2500 alarms. When the limit is exceeded, the table header will for example show that there are 40000 active alarms, but the table itself will only display 2500 alarms with a final entry indicating that there are more alarms that cannot be displayed.

In a control room, operators will not be able to inspect 2500 individual alarms anyway. Instead, they will then consult the alarm tree display to see in which area the alarms are concentrated.

Performance

Loading alarm configuration into RDB

The alarm configuration for 50003 alarms was programmatically generated as an XML file, sized about 16 MB. It takes about 5 minutes to load the XML configuration into the RDB with the AlarmConfigTool. Roughly one third of the CPU used by Java, the other 2/3 by the PostgreSQL RDB.

Soft IOC Performance

The EPICS database file for the 100000 records is about 17MB big. The soft IOC requires about 4 minutes to start. Once running, it uses 35% of the CPU.

Alarm Server Performance

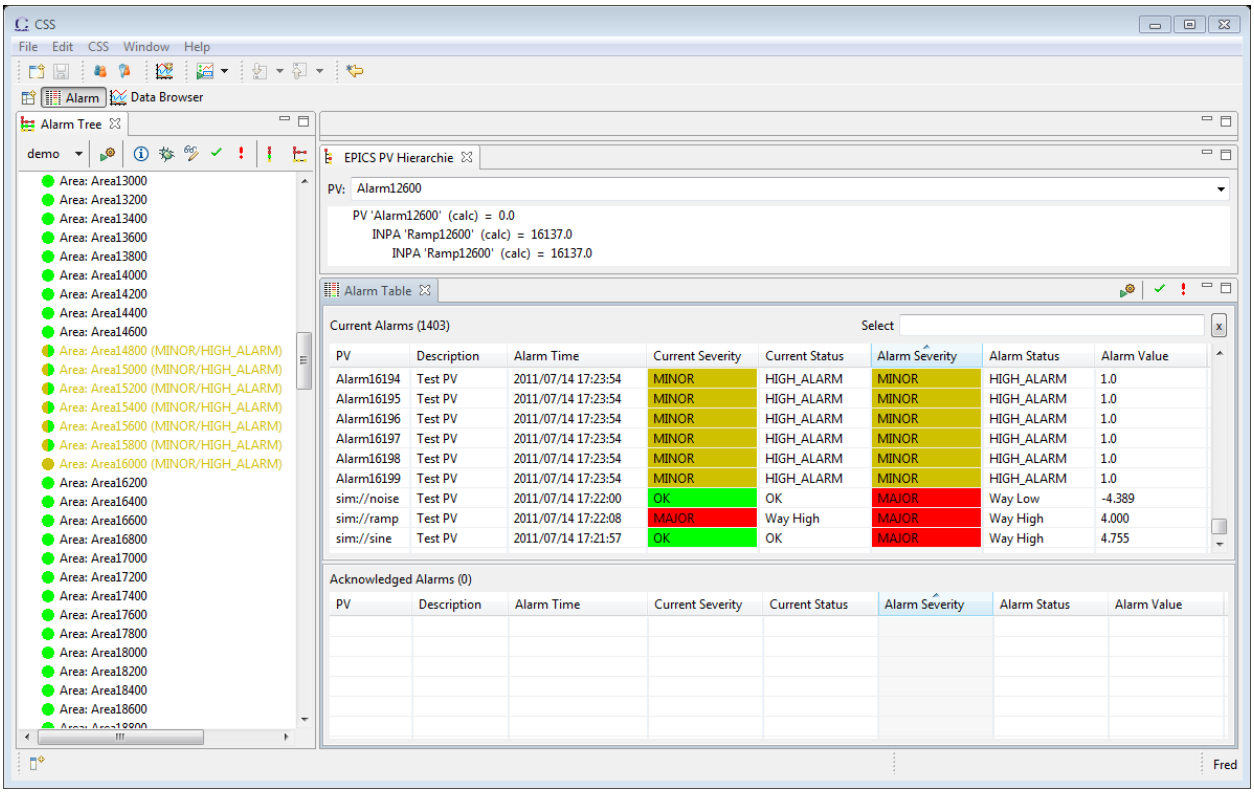
The alarm server needs only about 2 seconds to read the alarm configuration from the RDB, then about 30 seconds to connect to all PVs. Once running, it uses less than 1% of the CPU, briefly peaking to 5 or 10% every 40 seconds.

CSS Alarm User Interface Performance

In a running instance of CSS, when opening the Alarm Tree and/or Alarm Table display for the first time, it will read the alarm system configuration. This takes about 30 seconds. It is handled in a background task. The alarm GUI displays “Initializing”, the progress view displays the number of alarm PVs read so far, and CSS overall remains responsive.

Once initialized, the CPU from the CSS alarm GUI is 1 or 2% with brief peaks to 10% every 40 seconds as long as there is no user interaction.

When opening and closing sections of the alarm tree, acknowledging and un-acknowledging alarms, viewing the guidance of alarms, opening the EPICS PV Tree on an alarm etc., the CPU load goes up to 10 or 20%.



Latching Alarms

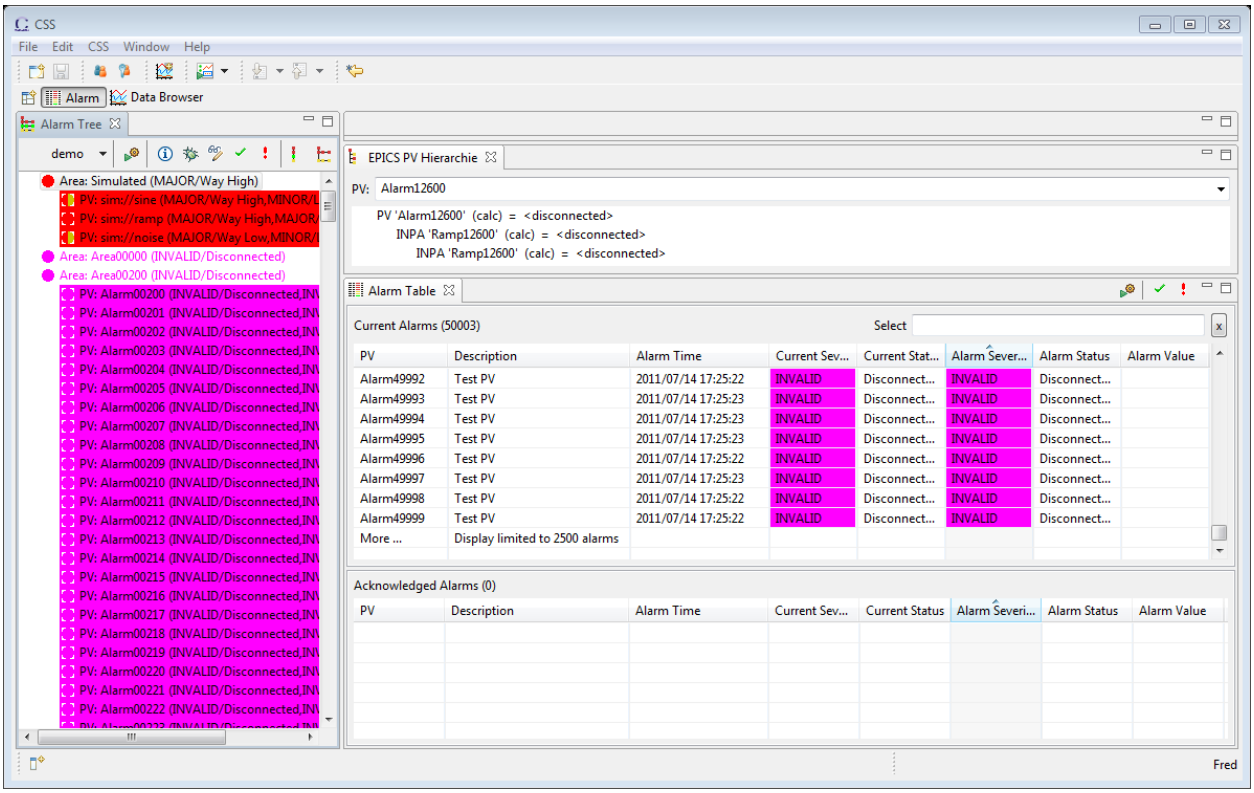
Initially, alarms were configured not to latch, so there were only about 200 alarms at a time. Alarms cleared as the associated PVs returned to an OK severity.

Next, all 50003 alarms were configured to latch via a simple SQL statement and an alarm server restart, i.e. within about 1 minute.

The alarm table would now fill with 200, 400, 600, … alarms until reaching 2500 alarms, after which further alarms are suppressed with a “More Alarm…” message.

Stopping the Soft IOC

When stopping the soft IOC, the alarm server will issue INVALID alarms for all 50000 inputs. This burst of alarm messages causes 100% CPU load for about 10 seconds. The CSS alarm GUI remained fully responsive. The alarm table indicates that there are 50003 alarms, but only displays details on 2500 of them. The alarm tree can be used to browse through all of them.



After re-starting the soft IOC (4 minutes), it takes a few minutes for all PVs to reconnect.

It can be tedious to acknowledge all 50000 individual alarms that latched the INVALID severity. The alarm system “Maintenance mode” from the alarm tree toolbar will automatically acknowledge INVALID alarms, allowing for a hands-off recovery.

Summary

The startup of the soft IOC, loading the initial alarm system configuration from XML into the RDB takes around 5 minutes.

A system with 50000 trigger PVs, generating about 7 alarms per second (constant stream of 2 per second, plus 200 every 40 seconds) performs without problems. In fact the alarm system only uses a few percent of the CPU, while the soft IOC requires 35%.

The CSS alarm system only shows up in CPU usage peaks every 40 second. The alarm user interface remains responsive except for a short period when all 50000 PVs become INVALID at once.