DAQ Monitoring and Auto Recovery at DØ

Monitor Design

• The simple monitor system is easy to write
  - Perhaps not very interesting
  - Often gets left to last
  - Crucial for long term debugging.

• Only a few basic design questions
  - Push/Pull
  - Direct connect to data sources or server
  - Request Granularity
  - Data Format
Ours Was No Different

- The DØ DAQ Monitor
  - Wasn’t written until it was clear we needed it
- General design principles were based on Run 1 experience
  - Too many displays affected system performance
  - Display maintained connections to every source
  - Data requested was only a large binary block.

Run 2 Environment

- Many more data sources
  - Of order 200
- Possibly many more displays
- Online System Firewall
  - Due to DOE security requirements.
- Primary debugging tool for shifters and remote experts
Design Principles

• Use a Monitor Server Architecture
  - Only connection to the data sources
  - Protocol is based on the pull

• Data Format is XML
  - Standard format that is easily extensible
  - Easily readable by any programming language.

• Bother monitor sources of data as infrequently as possible
  - Fine grained monitor data request
    • Some data may almost never be asked, but take significant resources to generate
  - Cache data in the monitor server for all data items.

• Be robust against crashes and other spurious effects
  - Auto reconnect built into clients and data sources

• Be able to handle a large number of data sources and displays
  - Heavily multithreaded

• Serve Real-time data
  - Cached data has a time stamp
  - Display data requests have an associated maximum age
    • Minimum time is one second
Monitor Server Implementation

- Written in C++
- Communication is TCP/IP, based on the ACE RT framework
- Data Sources connect to it
- Displays connect to it
- Caches data
  - One for every data source
- Highly multithreaded
  - Prevents blocking due to long latency displays

**Performance**

- Takes 10% of a 2 CPU machine under full Monitoring Load
  - PIII Xenon, 1 GHz.
- 4% of ½ gig memory
- Serves about ½ meg per second of monitor data
  - Cache Hit Rate is about 50%
  - Could setup second layer of a caching server if so desired
- Typical time to satisfy query is 50ms
- **Relay** to serve requests from outside online system
  - An access point through security firewall.
- 162 Data Sources, 35 displays
Wire Protocol

- All component connections are TCP/IP
- All run a common format
  - Message length word followed by the message text.
- XML format is simple!

Example Request To Client

```xml
<luminosity>
  <d0_luminosity/> <b0_luminosity/>
  <beam_energy/> <pbar_stack_size/>
  <outside_temperature/>
  <tev_protons/> <tev_pbars/>
  <tev_beam/>
  <store_number/>
</luminosity>

<luminosity>
  <d0_luminosity>13.28</d0_luminosity>
  <b0_luminosity>24.58</b0_luminosity>
  <beam_energy>987.43</beam_energy>
  <pbar_stack_size>24.58</pbar_stack_size>
  <outside_temperature>41.92</outside_temperature>
  <tev_protons>5869.87</tev_protons>
  <tev_pbars>602.7</tev_pbars>
  <tev_beam>6.8</tev_beam>
  <store_number>1949</store_number>
</luminosity>
```

This simplicity is perhaps the single biggest reason for this system's success.
Complex Monitor Data

- What is sent between monitor item tags is up to data source
  - Can be arbitrarily complex.
  - Matches best with rest of system if in XML
  - But this is not a requirement!
Smorgasbord...

Translate from other monitor systems

Reformats data from the MS

- SBC's
- Nodes
- Channel 13
- Node Collator
- RM
- Super
- Online Monitor Server
- COOR NS
- TFW L1L2
- DL Rates
- Store #
- Muon FE Monitor
- Monitor Server
Current Displays...

- L3 Monitor Server
  - uMon
  - fuMon
  - I3xQT
  - Web Pages
  - Windows Systray
  - daqAI
  - l3history

- Node Collator
Coding...

**C++ Data Source Code**

```cpp
l3_monitor_util_reporter_ACE monitor_callback ("tester");

L3_monitor_connector<l3_monitor_util_reporter_ACE>
ms_connection ("d013mon.fnal.gov",
DEFAULT_L3MONITOR_CLIENT_PORT,
&monitor_callback);

ms_connection.connect_to_server (true);

l3_monitor_object_op<int> counter ("count");

while (1) {
    counter++;
    std::cout << "Doing iteration " << counter <<
    std::endl;
    ACE_OS::sleep(1);
}

return 0;
```

**Python Display Code**

```python
import time
import l3xmonitor_util_module
disp = l3xmonitor_util_module.monitor_display
test_item = disp.get_item("tester", "count")

while 1:
    disp.query_monitor_server()
    print "Count value is %s" % test_item[0]
    time.sleep(1)
```

*Code frameworks in Python, C++, Java, and C#*
Complex Displays

Strip Charts

- Maximal amount of information
- Prevent 4am phone calls

Connections between FEC & Node

120 kb every 5 seconds
History Logger

- Keep track of data over long periods of time
  - Data Storage in Root
  - Data Storage in Oracle
- XML Format
  - Automatically parsed into ntuple like forms
- Change in data automatically reflected in data stored in DB
  - Web interface to change what is logged
- Make Plots in Real time using web interface
- Written in .NET

Monitor Server

Web Interface

Web Plotter

DB
Lessons and Out Look

- **Problems:**
  - Didn’t understand how to build strings efficiently in C++
  - Was generating a monolithic block of data in a single monitor item
  - Multi-threaded code that stays up for weeks at a time is hard

- **Text vs Binary**
  - Only once have we had to encode information in Text for compression reasons

- **The code is stand alone**
  - We rely on ACE for the monitor server
  - Display code frequently relies on Xerces and also ACE.
  - Data Source code relies on ACE and Xerces

- **Wire protocol is very simple**
  - Could write directly to it and skip dependencies.

Simple To Use  Everyone Uses It
Complex Systems...

- Experiment has many 1000’s of computer controlled components.
- Problems fixable
  - Long term
    - Visa problem!
  - Work around
  - Decrease data taking efficiency.
- Can this be addressed?

Called “daqAI” ← Bad Name!
New problems often require the addition of new data to monitor system

- All of detector is computer controlled
- Control is decentralized in DØ
- Safety/Political allowing computer to send commands

Hard!
Recognizing The Problem

• Must be dynamic
  - Each new problem should not require major update to code
  - Problems change
  - What problems symptoms indicate change

• Must be safe
  - Feedback loops are bad!
  - Should not cause harm or increase dead time!

Rule Based Recognizer

Monitor Data -> CLIPS -> Detector Commands
Monitor Data Converted to facts
(l3xetg-rate_events 55)
(coormon-store_number 2326)

Rules...

(defrule rate_is_low "See if the daq rate is less than 30 Hz"
  (l3xetg-rate_events ?num&:(< ?num 30))
  =>
  (assert (b_daq_rate_is_low ?num)))

(defrule rate_low "Rate is too low. Fix with reset"
  (b_daq_rate_is_low ?r)
  (coormon-store_number ?s&:(> ?s 0))
  =>
  (log_reason "Rate is low")
  (talk "Rate is low. SCL Init issued")
  (issue_coor_request "scl_init"))
Produces Error Reports

- Had to produce a report of what it had done.
- All problem detection is associated with a particular problem.
- Shift Problem Reports
  - Can see how often something goes wrong and how much downtime it causes.

```
# daqAI Shift report covering the period 2003-03-24-11:16:
1 times 'Bad A/O BOT Rate in term(193): muon' took 00:00:30 (255 secs on)
2 times 'Crate 0x61 is FEB' took 00:10:56 (328 secs on)
1 times 'Crate 0x64 is FEB' took 00:00:24 (24 secs on)
3 times 'Crate 0x6b is FEB' took 00:19:21 (387 secs on)
1 times 'L3 RM is asserting its disable' took 00:00:03 (3 secs on)
1 times 'Muon Crate 0x18 has fatal error' took 00:00:03 (3 secs on)
1 times 'Muon Crate 0x30 has fatal error' took 00:00:03 (3 secs on)
21 times 'Other' took 00:43:37 (124.619 secs on avg)

# Timer Status:
Timer in_store: 04:39:04 (off)
  86.0726% of in_store
Timer 13_configured: 04:00:12 (off)
  74.2594% of in_store
  86.2753% of 13_configured
Timer good_data_flow_to_13: 03:11:38 (off)
  68.6694% of in_store
  79.7807% of 13_configured
  92.4723% of daq_configured
Timer 13_disable: 00:00:16 (off)
  0.0955566% of in_store
```
Lessons...

- **Successful**
  - In the sense it dramatically decreased the downtime for common problems

- **Not Successful**
  - Requires constant up keep by myself
  - Must convince people this is the right thing to do
  - Adding new conditions is not as easy as it should be
    - Rule based is easier, but there must be a better way!
    - Have to be careful of feedback and unintended consequences
Conclusions

• The Monitor Server has been wildly successful
  - XML format along with simple wire protocol were key to this
  - A easily usable and unified monitoring system allows projects like the auto detection to be easily written.

• The daqAI Project was successful
  - Big help reducing and cataloging DØ’s down time.

• Future
  - Monitor Server is stable and not under active development.
  - Looking for a smarter way to implement the problem recognizer for daqAI.
The Outside World

Online ACL Firewall

MS

d0l3mon2

www-d0l3mon
d0l3mon2 only hole

MS Relay

www-d0l3mon

Display

No command information is passed back and forth, just request and query

Only displays can run outside...

Adding your data...

Simplified road map...

1. Choose a name for your data source
   “sbc” “daqAI” “luminosity”

2. Choose the language you’ll source the data
   C/C++  Python

3. Write Code!

4. Run behind online ACL
   No configuration changes required in MS
```cpp
#include "l3monitor_utils/client_objects/l3_monitor_object_op.hpp"
#include "l3monitor_utils/client_objects/l3_monitor_util_reporter_ACE.hpp"

int main()
{
    l3_monitor_util_reporter_ACE monitor_callback ("tester");
    L3_monitor_connector<l3_monitor_util_reporter_ACE> ms_connection ("d0l3mon.fnal.gov",
    DEFAULT_L3MONITOR_CLIENT_PORT, &monitor_callback);
    ms_connection.connect_to_server (true);
    l3_monitor_object_op<int> counter ("count");

    while (1) {
        counter++;
        std::cout << "Doing iteration " << counter << std::endl;
        ACE_OS::sleep(1);
    }

    return 0;
}
```

Comments removed!!

- **Client Name**: `l3_monitor_util_reporter_ACE`
- **Connect to MS**: `ms_connection.connect_to_server (true);`
- **Declare Integer to be monitored**: `l3_monitor_object_op<int> counter ("count");`
- **Change it once per second for fun using usual C++ syntax**: `counter++;`
import time
import l3xmonitor_util_module
ms = l3xmonitor_util_module.monitor_client("tester")
count=0
while 1:
    ms.set("count", count)
    time.sleep(1)
count += 1
import time
import l3xmonitor_util_module

disp = l3xmonitor_util_module.monitor_display_relay()

test_item = disp.get_item("tester", "count")

while 1:
    disp.query_monitor_server()
    if len(test_item) == 0:
        print "No data from the MS"
    else:
        print "Count value is %s" % test_item[0]
    time.sleep(1)