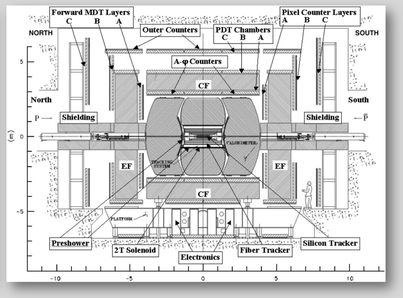
Process Watcher Application in D0 Experiment at Fermilab

V.Sirotenko, J.F.Bartlett, G.Savage
Fermi National Accelerator Laboratory

1 Introduction

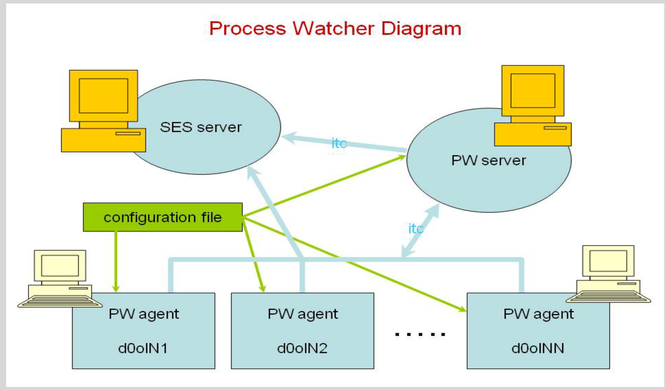
Dzero is one of two major experiments running at the Tevatron, the proton-antiproton Fermilab Collider, at luminosity up to $3x10^{32}cm^{-2}s^{-1}$ and center of mass energy 1.96TeV. The Dzero detector is a 5,500-ton multipurpose detector which is more than four stories tall and is composed of more than 1 million individual detector elements.



The sophisticated data acquisition, control and monitoring software ensures high data taking efficiency and quality. At D0 there are tens of applications running 24x7 on the online Linux cluster which consists of more than 120 nodes and many of those applications are vital for the data taking process. Therefore it is necessary to know at any moment that all such processes are running.

The idea was to write a special tool – Process Watcher (PW) – which monitors a set of selected user applications and sends alarms when it detects that an application is not running or has problems. One should note that such tool has to be quite general, it knows nothing about details of any specific application, so when we say “monitoring”, it means that only operating system (Linux in this case) resources can be used to perform this task.

2 Process Watcher General Architecture and Implementation Details



Design: a single PW server and many PW agents.
Both are written in Python.

- ❖ The server runs as daemon on a dedicated node. Its functions are:
  - monitor heartbeats from agents;
  - if an agent heartbeat was not received, send an alarm and (possibly) e-mails
- ❖ Agents run as independent cron jobs on every monitored node:
  - check metrics on selected node processes;
  - if a process check fails, send an alarm and (possible) e-mail;
  - send a heartbeat to the server;
  - receive a heartbeat confirmation from the server;
  - if a confirmation is not received, report an alarm.
- ❖ The server and agents use one central configuration file which contains information about nodes, processes and metrics on each node
- ❖ The server and agents have an automatic alarm clearance system
- ❖ Such architecture allows the dynamic addition/removal of any node or process to/from PW monitoring by editing the central configuration file

The configuration file, which is structured as a Python dictionary, is common to the PW server and all of its agents and may be modified at any time. The PW server rereads this file when it periodically checks for heartbeats from the agents and the agents, which run as cron jobs, read the file each time that they start..

Definition Block: OS commands, constants

General settings: defines some OS dependent commands used to monitor processes and sets some constants used

```
self.config = {
##################################################
'general settings':
 'cron_delta_time':180,
 'command_ps':'ps -efwwx',
 'command_top':'top b n 1 p',
 'command_service':'/usr/sbin/c/ustat',
 'command_check_disk':'df| grep %s| grep %s',
 'file_pwagent_top_prefix':'/tmp/pwagent_top_',
 'file_pwagent_epics':'/tmp/pwagent_epics.pic',
 'epics_delta_time':10800,
 'check_scratch':('df| grep scratch | grep %','95),
},
```

▪Here ‘cron_delta_time’ is the time interval in secs which is used to determine if auxiliary /tmp files left from previously running PW agent should be reset
▪And ‘epics_delta_time’ is the time interval in secs which is used to determine if EPICS IOC is updating

Process Metrics Description

For process to be watched on any given node, the following metrics block must be specified inside configuration file, see example below:

```
self.config = {
 ..........................
'd0olgw2':
[
 {
  'process':'serverAcnet.py',
  'proc_att':['python'],
  'user_id':'d0run',
  'singleton':1,
  'top_history_depth':5,
  'max_pcpu':20.0,
  'max_pmem':5.0,
  'min_num_threads':6,
  'thread_checks':{'max_pcpu':50,'max_pmem':10},
  'check_file':'/online/log/server_acnet/serverAcnet.py-*':3800}
 },
 { another process metrics description, if any},
 ..........................
],
 ..........................
```

Here metrics blocks are organized as list of dictionaries, one dictionary for every process, with keywords describing what should be monitored

Cron jobs, disk space, EPICS IOC description

To monitor cron jobs there should be some output file which is updated when cron runs, then the metrics is done like that:

```
{
 'process':'crond',
 'user_id':'root',
 'check_file':'/online/log/runpusher/dbg-runDbTransfer.*':700}
},
```

The same way one can monitor disk space available

```
{
 'process':'crond',
 'user_id':'root',
 'check_disk_space':('/mnt/lum':95,'/mnt/daq':95,'/usr/products/':95}
}
```

To monitor EPICS IOC nodes (through CA protocol checks if IOC is accessible and timestamp is up-to-date) the special keyword ‘epics’ is used and the metric block in this case looks like:

```
{
 'epics':[ 'd0olctl26','d0olctl27','d0olctl33','d0olctl133',
           'd0olctl70','d0olctl71','d0olctl80', . . . . . ]
}
```

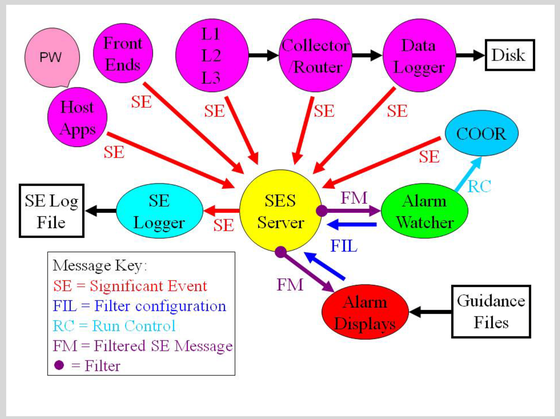
List of keywords used

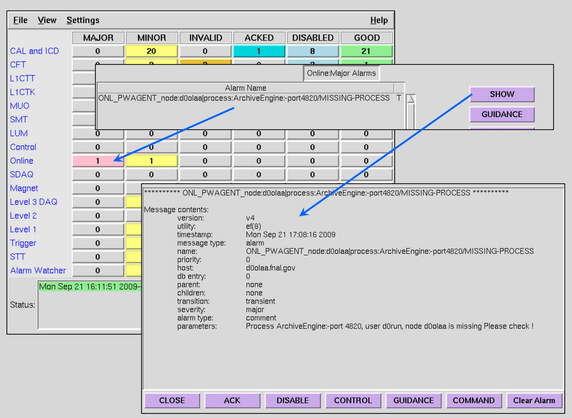
List of the possible keywords in the metrics block:

'process':'process name'	-	unique procID is constracted as 'process_name:at1:at2:. .'
'proc_att':['at1','at2',...]	-	[optional], attributes as seen in output of 'ps –efw' command
'user_id':'<name>'	-	if set, process must be run under given user_id <name>
'singleton':1	-	if defined then process must have a single instance
'top_history_depth':N	-	depth of the internal history file (default=5)
'min_num_threads':N	-	number of required threads (default=0)
'thread_checks':{'max_pcpu':N,'max_pmem':M}	-	if set, checks that any thread should not consume more than N% of CPU and M% of memory
'check_file':{'/mnt/archive/rad/current/200*-delta*'}	-	if set checks that given file is not older than deltaT seconds
'max_pcpu':N	-	if set checks that process CPU(%) must be < N
'max_pmem':N	-	if set checked that process MEM(%) must be < N
'skip_check':N	-	if not 0 check for alarms will be done every Nth run of agent
'alarm_confirmation':N	-	if not 0 alarm is send to SES only if re-appeared again after consecutive N pwagent cron execution
'check_stale_cpu':0	-	no check for stale state is done (default=1)
'store_check':1	-	alarm for missing process is send during the store only
'global_run_check':1	-	alarm for missing process is send during global physics run only
'check_disk_space':{'/mnt/lum':N,'/mnt/daq':M,'/usr/products/':K}	-	if set checks that used disk space for given disks is less that N/M/K(%)
'epics':['d0olctl26','d0olctl27', ...]	-	list of epics IOC to be monitored

3 Significant Event System

The Significant Event System (SES) collects and distributes all changes of state of the detector and the data acquisition (DAQ) components. These include alarms from the slow control system, physics data monitoring applications, and the PW server and agents. The DAQ coordination application (COOR) also sends the current run state to the SES. The SES has a central server that collects event messages from sender clients and filters them, via a Boolean expression, for receiving clients. Sender clients, which include the EPICS IOCs, connect to the server via ITC, a locally developed, inter-task communication package based upon TCP/IP sockets. All state changes on those clients, including alarm transitions, are sent to the server. In this scheme PW server and agents are SES clients and all alarms generated by them, are sent to the SES server.





4 D0 Online Applications Monitored by the PW

Application Name	check if running	check user	check if singleton	check stale file	check max cpu	check max mem	check if stale cpu	check disk	min # of threads	max thread	max thread
Epics Archivers (12)	X	X	X	X			X				
DM Server	X	X			X	X	X			X	X
DbMonitor	X	X	X	X	X	X	X				
RunGrabber	X	X	X		X	X	X				
RunPusher	X	X		X			X				
mu_missing_input	X	X	X		X		X				
l3xDAQSuper	X	X	X			X	X				
dagAI_XMLMonitor	X	X	X			X	X				
DAQMON_Scraper	X	X				X					
lmTCC	X			X		X	X				
lmServer	X	X		X		X	X				
lmL3	X	X		X		X	X				
lmACNETGW	X	X		X		X	X				
lmACNET	X	X		X		X	X				
serverAcnet	X	X	X	X	X	X	X		X	X	X
MuoExamine	X	X	X	X							
dq_calo_x	X	X	X								
l3monitor	X	X	X	X	X	X					
runTrigExAuto	X	X	X	X							
VertexExamine_x	X	X	X								
runBeamSpotMonitor	X	X	X								
runPhysExAuto	X	X	X	X							
readout_client	X	X	X		X	X	X		X		
ctt_gui	X	X		X							
CalMuoCheck	X	X	X		X	X					
l1calldmd	X	X		X							
selogger	X	X	X								
sealarmwatcher	X	X	X				X				
cron	X	X						X			
epics IOC watcher	X										

5 Conclusion

To monitor all of the vital components of the D0 online system, a special Process Watcher application has been developed. It constantly monitors the health of the running processes and immediately informs shifters in the D0 control room about any arising problems. We’d like to share our experience controlling collider experiment for the benefit of future HEP detector operations.