CDF/DOC/COMP_UPG/PUBLIC/5556

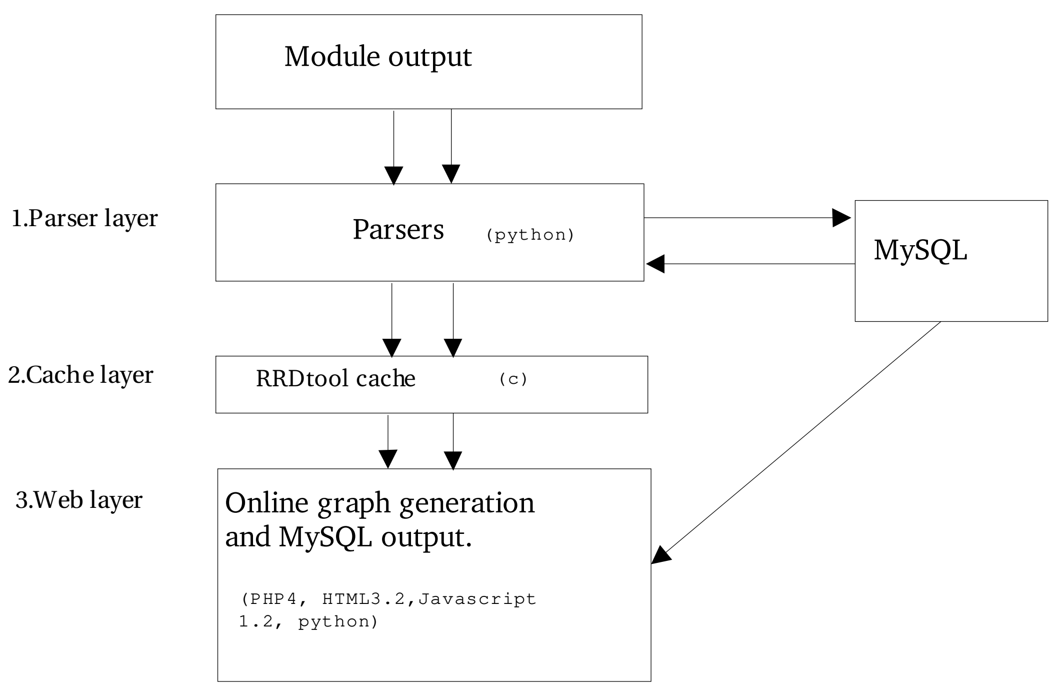
Design of the Production Farm web interface.

Jaroslav Antos, Marian Babik , Yen–Chu Chen, Miroslav Siket, Stephen Wolbers

2.february 2001

The CDF experiment at the Fermi National Accelerator Laboratory will record data in the year 2001 with a maximum throughput rate 20 MB/s . To analyze data in quasi–real time, great computing power is needed. The major requirements for the CDF production farm are high throughtput rate and I/O capability, easily configurable system, excellent bookkeeping and error handling . In order to accomplish these requirements a great effort went into the monitoring of hardware performance as well as status reporting and tuning of the software part of Production Farm (so called FPS–Farms Processing Software [1]). This document will describe the design, source code and configuration of the production farm web interface (PFWI). The interface should be able to parse, calculate and display all major characteristics of the FPS (including progress status, farmlet information, nodes information, reconstruction progress, resource allocation, file tracking). Since the progress and performance indicators are changing in real–time and thorough understanding of the mechanisms and influences is native to the operation of the farm, the interface needs to report on–line results .

In the design of the PFWI on–line response , real–time calculations and processing were considered as the major requirements.The whole system was divided in two parts: a system of parsers (pre–processing and storing of the data) and a web–interface (for on–line formatting and display). The design had three layers of functionality : parsing; cache ; web (figure 1 shows the layen–line formatting and display) .



There were various ways to accomplish the requirements of the project and there is no doubt several other designs would led us to more robust and efficient system. I have tried to choose a system with the best time/performance ratio. CGI [2] and perl [3] (or C language) were considered but the difficulties with debuging the CGI scripts and complicated object–oriented approach in perl led us to try HTML[4] embedded scripting languages like Cold Fusion, PHP[6] or JSP[7] . Probably the best choice would be the JSP , but since this wasn't a huge project and it was expected to be done in relatively short period of time (with non–commercial tools) the PHP language was preferred to the JSP. Also the majority of the FPS is written in python [12], thus choosing Java would mean rewriting most, if not all the libraries for the parser layer.

The IO between layers is based on accessing files, the majority of which are FPS output or configuration files (since FPS is divided into modules, I will sometimes refer to FPS output as module output). Sometimes posting a request to the MySQL database system is needed (this is the database that stores FPS's crucial informations such as opened sockets, configuration of farmlets, status of individual modules etc.[8]). The parser layer is thus accessing MySQL or FPS output files (module output) and after processing, the data are fed to the cache layer. The web part is accessing cache layer or MySQL directly as its input and generates the HTML pages with graphs or tables on user request.

The parser layer is used to transform the modules output, to the form, that could be than transferred to the cache layer. The transformation is accomplished most of the time by reading file or querying the MySQL database. Text processing and calculations (average, max values) are made on the data. The cache layer is Tobi Oetiker's rrdtool [9]. RRD is the Acronym for Round Robin Database. RRD is a system to store and display time–series data (i.e. network bandwidth, machine–room temperature, server load average). It stores the data in a very compact way that will not expand over time, and it presents useful graphs by processing the data to enforce a certain data density. It can be used either via simple wrapper scripts (for us python) or via frontends that poll network devices and put a friendly user interface on it. The web layer is accessing RRD files, special files or queries from the MySQL database. Then it generates HTML pages using PHP4 scripting language.

Parser layer.

The parser layer consists of four parsers :

- progress status parser (progressPd.py)
- process parser (nodes system information parser , processPd.py)
- resource parser (resourcePd.py)
- rates parser (ratesPd.py)

The progress status parser (progressPd.py) is accessing MySQL database tables phyfhist and phyfiles. The queries are :

```
data1=db.query("select
dataset,exe_version,comment,rfsize,num_event,lostevnts from phyfhist")
data2=db.query("select dataset,exe_version,comment,file_size,num_event
from phyfiles")
```

This will create a dictionary (data1) that has all the data for the files in a given dataset, exe_version and comment ; another dictionary (data2) that has the same information about output files. Using DFCFileTool [10], totals are available for a given dataset. DFCFileTool is tool for accessing main database server for the off–line part of the experiment. A simple ratio calculation is performed and the resulting data are written to the file specified in the parsers.ini. Since DFCFileTool is accessing Oracle database and queries totals, the cycle of the progressPd.py is 30 minutes (reconstruction of one file could take up to four hours).
To represent data for the given dataset , exe_version and comment, a three dimensional dictionary was used (dictionary of dictionary of dictionary). This way we have very simple access to any data.

The resource parser (resourcePd.py) is accessing resource manager's (FPS module) output file stats.dat. Stats.dat is representing all the information about acquiring and releasing of resources used by farm (see the description of the FPS for the details). The file has the following format:

```
#time: resource1, resource2, resource3 .....
farmlet1: absolute value for resource1, farmlet's percentage for
resource1, farm's percentage for resource1; same for resource2;
farmlet2: .....
.
.
Farm: totals for all farmlets
```

The resource parser is reading the file stats.dat and passing transformed data to the RRD library. RRD library is used to perform all the basic functions of the rrdtool using method's call and arguments passing (see rrd.py for the details).

The rates parser (ratesPd.py) is accessing MySQL database tables configured in the file rates.ini. The queries are:

```
query="select
"+file_size+",unix_timestamp("+time1+"),unix_timestamp("+time2+") from
"+histtable+" where unix_timestamp("+time1+")>"+self.start_time
d1=self.db.query(query)
if table!="":
    d2=self.db.query("select
"+file_size+",unix_timestamp("+time1+"),unix_timestamp("+time2+") from
"+table)
```

The algorithm to calculate rates divides time into 30 second intervals and at the end of each interval it is checking MySQL. If there are entries for the given time and if it finds valid entries it will calculate the average rate by dividing the total size of the file by the time interval, the total rate is calculated by adding all partial rates, as seen from source code:

```
 while(self.step_time <= self.last_time
      if self.data!=[]:
         for item in self.data:
```

```
            i_rfsize=float(item[0])
            i_1time=float(item[1])
            i_2time=float(item[2])
            if (self.step_time>i_1time) and (self.step_time<i_2time):

                # rfrate=rfsize/(ptime-ttime)
                rfrate=i_rfsize/(i_2time-i_1time)
                # print "RATE:",rfrate
                rate=rate+rfrate
```

The resulting totals are passed to the RRD library.

The process parser (processPd.py) is part of the system that was written from scratch and relates to the hardware monitoring part. The general idea was to run process daemon (procd) on every node that will collect system data (CPU, load average, memory usage, network load) and store it in the file. The process parser is reading this file and after text processing it's passing the data to the RRD library. Procd is built on top of the daemon core, that is general for the all FPS's modules, but could be easily implemented without it. There is also launcher.py that executes procd.py on the nodes specified in the file (see launcher.py −−help). There are three basic commands : start, stop and status. It's using SSH to connect to the nodes.

Cache layer.

The cache layer is used to do statistical preprocessing (average), easy storage of the data (not growing) and has interface to visualize the data (graphs). The library is using rrdtool (described earlier), it is parsing command line arguments to the shell, which executes rrdtool. (for details see rrd.py and many documents describing use of the rrdtool [9])

Web layer.

The web layer's initial objectives were to display all the information collected by the parsers and to gather data that needs no preprocessing and thus could be requested on−line (farmlet information, file tracking − static data or data that doesn't need any calculations etc.). To generate web pages PHP4pl2 was used. I have written the basic three libraries : RRD library (to create graphs on−line), MySQL library (to connect and query the MySQL library) and HTML build library (for the details see rrd.php , mysql.php and html_build.php). There are seven basic pages : resources, farmlet information, progress status, rates, file tracking, messenger, nodes(see [11]). Resources, rates and nodes are accessing RRD files and with the use of html_build library are generating the web−pages on user's request. Farmlet information page is accessing MySQL table farmlets to gather the information. Progress status is accessing file specified in the config.php and after processing it's displaying its content. Messenger is querying messenger table in the MySQL. File tracking is gathering data from the rawfiles table.

Summary.

The engine of the PFWI system has been finished in less than three weeks, since then lots of the parser and the web functionality has been added, but the main core hasn't been changed. RRD tools as well as PHP and python has shown a very easy and flexibile way to develop real–time processing and display. Further development of the libraries for the web layer (generators of pages and graphing library) as well as cache layer would yield more efficient system. Also a development of independent cache layer in python would led to a less complicated design, thus more efficiency.

References:

[1] M.Siket, FPS manual and note
[2] Common gateway interface http://www.cgi–resources.com
[3] Sriram Srinivas, Advanced perl programming
    Scott Guelich, Shishir Gundavaram, and Gunther Birznieks , CGI programming
    http:/www.perl.com
[4] HTML http://www.w3.org/MarkUp
[6] PHP , http://www.php.net
[7] Java Server Pages, http://www.sun.com
[8] MySQL, http://www.mysql.com
[9] Round Robin Database, http://ee–staff.ethz.ch/~oetiker/webtools/rrdtool/
[10] DFCFileTool documentation
[11] PFWI sources (fnpcc.fnal.gov)
[12] Python, http://www.python.org
    Mark Lutz, Programming python
    Mark Lutz and David Ascher, Learning python