Linux System Monitor Detailed Design

Working Design Proposal

v. 1.0, 9/30/09, Dan Graham

This is a working detailed design document for a system to monitor the Linux servers.

Introduction

The document “Linux_System_Monitor.doc” provides a rough design for this project. Additional architectural considerations, a database design, and an object-oriented design will be presented, here.

Motivations

It is desired that the Linux system monitor would have advantages over manually going to a server and examining it. The following advantages were identified:

1. Able to see things when not presently monitoring.

If a system failure occurs at an unexpected time, such as at night, the Linux system monitor would allow viewing of the state of the system when the crash occurred.

2. Statistics: Aggregations of data would be recorded so that trends can be determined.
3. Charts: This would allow a more intuitive picture of the performance of a system.
4. Immediate response: The web interface would allow immediate viewing of all of the systems in the Linux infrastructure.
5. Alerts: A critical condition in a system could trigger an email or page, to allow a proactive response, before a system failure occurs.

Architecture

Back End:

The probes can be derived from an object-oriented hierarchy of Perl classes. The probes push XML data to the central server, that will be loaded and transformed in the database. *A key idea, here, is to make the system plug-and-play, so additional monitoring probes can be added without changing the database structure or front-end.* The XML syntax can be made very generic and orthogonal, in an approach that would accept any XML data format. Extending on the example, from the previous document, an example XML data snippet could be the following:

Memory:

```
<mem     time="200910011213"    mem="System Memory Usage">
     <pid   key="3"    pid="Process ID">
        12345
     </pid>
    <command  key="2">
        ssh
     </command>
     <owner   key="1">
        degraha
     </owner>
     <percent  counter="percent,sum,avg, min, max">
        30
     </percent>
</mem>
```

The idea is that the system (database and front-end, or "Model" and "View") does not need to know anything about the System Memory measurement group prior to receiving this XML. This would allow fast development and extension of the system, by developing a new probe and leaving the rest of the code alone.

The only keywords in the, above, grammar are the "time", "key", and "counter" attributes.

Also, the meanings of the “counter” attribute values, “percent”, “sum”, “avg”, “min”, and “max” are pre-defined.

The meanings of these attributes will be explained, shortly.

Measurement Topologies:

The entities which are being measured can be described by a hierarchical topology. Each node in the topology can be “transient” or “non-transient”. A “non-transient” node is a site or server, for example. A “transient” node is a process being monitored or system statistic, for example. For our purpose, the only difference between a “transient” and “non-transient” node can be the way it is used on the web page. A topology contains *topology keys* and *topology key values.* The root key for the topology is “root” and its value is “LINUX”. The next level would be “site”, with a possible value of “stpaul”, for example.

In the, above, XML example, the sequence of *topology keys* would be:

root, **site**, **server, mem**, **owner, command, pid**

The *topology key values* could be (for example):

“LINUX”, “stpaul”, “ abc.com”, NULL, “degraha”, “ssh”, “12345”

Associated with this sequence of topology keys and topology values, is a *time, counter,* and *counter-value.* In this case, the time is ”200910011213”. The counter is “percent”. And, the counter-value is 30.

The definition of the XML attributes is as follows:

time : The time in GMT (YYYYMMDDHHMMSS)

key: This identifies the XML tag as a topology key whose sequence is ordered by the numeric value of this attribute.

counter: This identifies the XML tag as a counter. The value of this attribute may be one or more of the following comma-separated values:

percent: This counter is a percent, so other nodes of type “mem”, with *identical “times”*, will be grouped together to form a pie-chart.

s*um, avg, min, max:* These values indicate that these formulas will be aggregated on a hourly, daily, weekly, and monthly basis.

Attributes with identical names to the tag name: This defines the textual caption or column heading that will appear on graphs and tables in the web page.

So, a summary of the back-end (Controller), is that the probes generate XML which is sent to the central server. (Or retrieved by the central server, by sftp). This XML loads the MySQL tables and dynamically creates new measurement groups.

Notes about Front-End

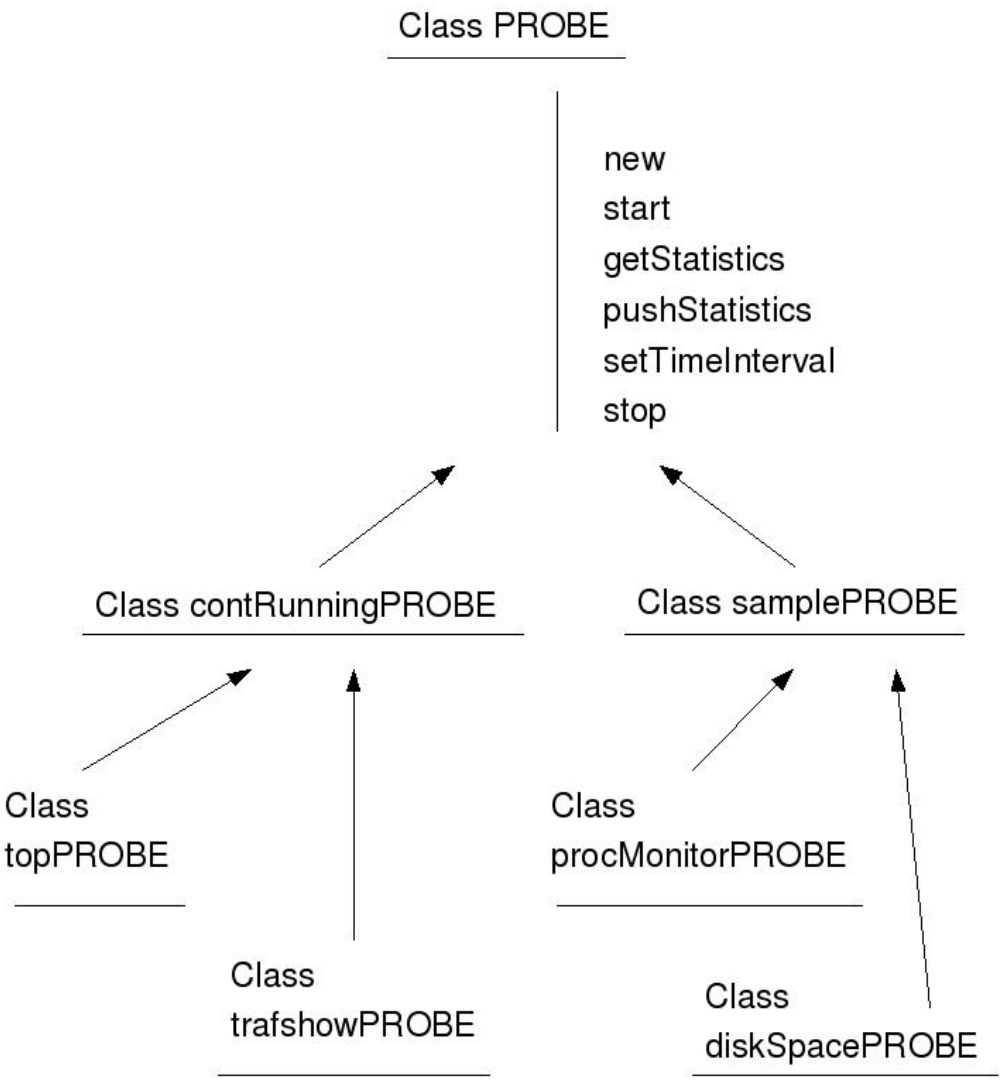
In order to take advantage of Template-Toolkit, the front end can be template-driven. The templates can be created with combo-boxes, etc., that select graphs and tables to display, from the data in the MySQL tables. Since this data is generic, the template can be written once and does not need to be modified for changes or additions to measurement groups, sites, or servers. The Template-Toolkit contains a rich macro-language for generating the HTML, so it should be no problem to create a generic template that shows charts using GD::Graph. If, for some reason, the template approach doesn't work, for this application – a fall back plan is to use a Java/Swing applet and the JFreeChart API.

Back End, Continued

The probes can be designed with an object-oriented Perl hierarchy. New probes can be derived from these classes. For some kinds of probes, data needs to be gathered from a

continuously running process such as the “top” or “trafshow” command. Substatistics can be computed and aggregated for the five-minute interval that is reported. (It wouldn't make sense just to get a snapshot of memory usage or CPU utilization every 5 minutes, with no data in between). There has been some success with using the CPAN module Term::VT102 to get information from a screen-oriented running command, like “top” or “trafshow”. Other kinds of probes just need a snapshot once every five minutes. Examples of this include disk partition usage (“df -k”) and monitoring to see if a daemon process is still running (“ps” command). Another example includes using the “last” or “ac” command to get user login times and durations.

The following is a diagram showing a possible object-oriented design for probes:



Database Design

The following MySQL tables can be created to support this application. This assumes that a topology would not exceed a depth of 10. They should be purged of data older than 90 days. Appropriate indexes should be added.

```
mysql> create table meas_vals (TopId int(5),
    -> Key1 varchar(100),
    -> Key2 varchar(100),
    -> Key3 varchar(100),
    -> Key4 varchar(100),
    -> Key5 varchar(100),
    -> Key6 varchar(100),
    -> Key7 varchar(100),
    -> Key8 varchar(100),
    -> Key9 varchar(100),
    -> Key10 varchar(100),
    -> counter varchar(100),
    -> meas_time date,
    -> val float(10));
Query OK, 0 rows affected (0.00 sec)

mysql> create table topo_def (TopId int(5),
    -> TKey1 varchar(100),
    -> TKey2 varchar(100),
    -> TKey3 varchar(100),
    -> TKey4 varchar(100),
    -> TKey5 varchar(100),
    -> TKey6 varchar(100),
    -> TKey7 varchar(100),
```

```
    -> TKey8 varchar(100),
    -> TKey9 varchar(100),
    -> TKey10 varchar(100));
Query OK, 0 rows affected (0.04 sec)


mysql> create table topo_desc (TopId int(5),
    -> TKeyDesc1 varchar(100),
    -> TKeyDesc2 varchar(100),
    -> TKeyDesc3 varchar(100),
    -> TKeyDesc4 varchar(100),
    -> TKeyDesc5 varchar(100),
    -> TKeyDesc6 varchar(100),
    -> TKeyDesc7 varchar(100),
    -> TKeyDesc8 varchar(100),
    -> TKeyDesc9 varchar(100),
    -> TKeyDesc10 varchar(100));
Query OK, 0 rows affected (0.00 sec)


mysql> create table meas_types (TopId int(5),
    -> Key1 varchar(100),
    -> Key2 varchar(100),
    -> Key3 varchar(100),
    -> Key4 varchar(100),
    -> Key5 varchar(100),
    -> Key6 varchar(100),
    -> Key7 varchar(100),
    -> Key8 varchar(100),
    -> Key9 varchar(100),
```

```
    -> Key10 varchar(100),
    -> counter varchar(100),
    -> percent bool,
    -> sum bool,
    -> avg bool,
    -> max bool,
    -> min bool);
Query OK, 0 rows affected (0.00 sec)


mysql> create table hourly_agg (TopId int(5),
    -> Key1 varchar(100),
    -> Key2 varchar(100),
    -> Key3 varchar(100),
    -> Key4 varchar(100),
    -> Key5 varchar(100),
    -> Key6 varchar(100),
    -> Key7 varchar(100),
    -> Key8 varchar(100),
    -> Key9 varchar(100),
    -> Key10 varchar(100),
    -> counter varchar(100),
    -> time_hour varchar(10),
    -> agg_type int(1),
    -> val float(10));
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> create table daily_agg (TopId int(5),
    -> Key1 varchar(100),
    -> Key2 varchar(100),
    -> Key3 varchar(100),
    -> Key4 varchar(100),
    -> Key5 varchar(100),
    -> Key6 varchar(100),
    -> Key7 varchar(100),
    -> Key8 varchar(100),
    -> Key9 varchar(100),
    -> Key10 varchar(100),
    -> counter varchar(100),
    -> time_day varchar(8),
    -> agg_type int(1),
    -> val float(10));
Query OK, 0 rows affected (0.01 sec)


mysql>mysql> create table weekly_agg (TopId int(5),
    -> Key1 varchar(100),
    -> Key2 varchar(100),
    -> Key3 varchar(100),
    -> Key4 varchar(100),
    -> Key5 varchar(100),
    -> Key6 varchar(100),
    -> Key7 varchar(100),
    -> Key8 varchar(100),
    -> Key9 varchar(100),
    -> Key10 varchar(100),
```

```
    -> counter varchar(100),
    -> time_week int(2),
    -> agg_type int(1),
    -> val float(10));
Query OK, 0 rows affected (0.01 sec)


mysql> create table monthly_agg (TopId int(5),
    -> Key1 varchar(100),
    -> Key2 varchar(100),
    -> Key3 varchar(100),
    -> Key4 varchar(100),
    -> Key5 varchar(100),
    -> Key6 varchar(100),
    -> Key7 varchar(100),
    -> Key8 varchar(100),
    -> Key9 varchar(100),
    -> Key10 varchar(100),
    -> counter varchar(100),
    -> time_month varchar(6),
    -> agg_type int(1),
    -> val float(10));
Query OK, 0 rows affected (0.00 sec)


mysql>
```

Note about Log Monitoring

In addition to gathering performance metrics on systems, it might be advantageous to scape log files, as well. One possible tool for doing this is the open-source “Simple Event Correlator”: http://simple-evcorr.sourceforge.net/

This tool allows specification of very complex rules for determining events based on information gathered from log files. Since, system problems can also be diagnosed from log files.