Monitoring Setup of Prague T2 Site

M. Burdik¹, J. Chudoba², L. Fiala², T. Kouba², J. Kundrat², M. Lokajíček², J. Svec²

1) Charles University in Prague, Faculty of Mathematics and Physics
2) Institute of Physics, Academy of sciences of the Czech Republic, v.v.i.
3) CESNET z.s.p.o.

koubat@fzu.cz

Abstract. Quality monitoring is a crucial part in struggle for reliability of every Tier 2 site. We present a solution based on Nagios installation in Prague. Our solution includes both third party and in-house developed add-ons and plug-ins. We focus on integrating results from grid-wise monitoring tools (such as SAM or Gstat) at one place (our Nagios instance). We also present an automated way to generate configuration files for Nagios from local database of hardware and services.

1. Introduction
One of the crucial problems in computing for LHC and also in the EGEE project is reliability. After a long phase of development now is the time when operations becomes very important because right operation procedures and tools can help increase reliability of the grid.

One of the part that helps administrators to keep good reliability of their services is monitoring. There are various tools available to monitor hardware and software nowadays. We will present our choice and setup in Prague T2 site.

Prague T2 site is a middle-sized site with about 490 CPU cores and 700 kSI2000. There are many special hardware units that must be monitored and some core services for virtual organizations VOCE and Auger.

2. Available software
When we were first choosing the monitoring software there were not many choices and Nagios[1] was the only mature enough tool. Now there are plenty of more integrated (but still open source) solutions like zenoss, zabbix, moodss, ganglia etc. But Nagios also developed a lot so we are happy with our choice and we consider Nagios (with proper configuration) to be still a good solution for a site of our size.

2.1. Why Nagios
There are several pros and cons for Nagios. The important pros for us are:

- Wide community of users and developers (a lot of plug-ins and add-ons)
- Simple plug-in API (it is quite straightforward to write a new plug-in)
- Clean philosophy (hosts, services are basic blocks with relations)

And cons:

- Static configuration (no automatic service discovery)
- Complex configuration (steep learning curve)
- Notification storms in case of massive failure

We have tried to solve the listed problems by introducing “problem summaries” and “configuration generator”. Both will be described below.

2.2. Used plug-ins and add-ons

As a Tier 2 site we use many additional plug-ins related to hardware and gLite middleware. We can recommend these:

- **default sensors** - ping, disk, procs, load, pbs_mom (tcp port check), swap, ldap, check_gentoo_glsa, check_gentoo_service_rc_all (by Wolfram Schlich, member of the gentoo development group)

- **SRCE sensors**
  - cert (certificates on nodes with gridftp)
  - DPM (sanity of dpm-qryconfoutput)
  - DPNS (sanity of dpns-1s output)
  - edg_broker (sends simple testing job via the tested RB)
  - globus_glam2 (submission to gatekeeper)
  - gridftp (gridftp availability)
  - LFC (sanity of lfc-1s output)
  - SRM(srmv1transfer)
  - SRM_ping (srm availability)
  - VOMS (availability of voms server to issue a proxy)

- **RAL sensors**
  - lcg_same (status of the given service in SAM)

- **locally written sensors**
  - hpacucli (status of physical disks on MSA20 disk array)
  - blade_temp (temperature in enclosure for BL460c)
  - ups (status of UPS)
  - gstat (status of GSTAT checks published on gstat web)

There are also few add-ons we use:

- **Npce** - For remote execution of commands that provide sensor information (for example swap and ram usage).
- **Nuvola** - Better look and feel of Nagios web interface.
- **NagiosGrapher** - This add-on allows administrator to use output of sensors as values in rrd-generated graphs. So, for example, the number of processes can be represented (drawn) only with Nagios and this add-on (see picture below).
2.3. Nagios digest

One of the problems we faced was a lot of notification mails when a major failure on site occurs (an example would be a case of power failures). This is solved in Nagios by service dependency, which means that every service can be dependent on another service and if the first does not work the notification is sent only for this first one not for the dependent ones.

Dependencies stop working if you cannot monitor the service that can be crucial and cause the failure (an example would be a case of failure of the fuse unit in the rack with worker nodes).

Even if you monitor all parts of the hardware infrastructure the number of notifications can still be too high for you.

In order to solve this problem, we have developed a NagiosReport. It is a tool that parses Nagios log files (for errors occurred), configuration (for list of services and hosts) and state files (to eliminate failures on hosts in downtime). We send the result report every 8 hours. The report also summarizes problems that occurred and were solved within the 8 hour window.

3. SiteQuery generator

3.1. Problem definition

The second main problem with Nagios is its static configuration. Every single host must be added by hand into text files. And all services for the host specified. The configuration philosophy makes this task easier by using templates. So most of the configuration options (that are the same for all nodes/services) can be specified only in template and the host definition just contains reference to the template. This reduces the amount of text but not the complexity itself.

At Prague Tier 2, we have a database with hardware already. So we decided to use it as the data source and only extend it with Nagios specific data and service information. Then we have specified a SOAP interface (WSDL file) which defines API to the database that allows client to view the database from Nagios point of view (hosts/services and their relations).

We have also developed a python client that actually generates the configuration for Nagios. There are various WSDL parsers and SOAP implementations available for python. We have tested three of them: pySOAP, SOAPpy and ZSI[2]. And only the ZSI is currently mature enough so we used this one (pySOAP is not developed any more and SOAPpy is being replaced by ZSI).
Because of complexity of the WSDL specification we do not write the WSDL file by hand but define the API in header file for C language and generate the WSDL from the header file by soap2cpp binary from package gSOAP. Then we generate the python wrapper functions and data structures by wsdl2py and wsdl2dispatch python tools from ZSI. This might sound a bit complicated but with simple Makefiles it is pretty straightforward way how to add web service (aka SOAP) interface to an existing application.

The development of this solutions continues and its home address can be found on central pages for many other Nagios extensions [3].

4. Conclusion

We have presented a view on Nagios utility from the point of a typical Tier 2 site. It is a very good and simple framework for executing tests and issuing notifications to administrators. We have also presented our solution for the biggest problems that we encounter in Nagios.

NagiosReport that eliminates the number of problem notifications and sends only a daily summary of problems.

SiteQuery generator - a tool for automatic creation of nagios configuration via SOAP interface.

References
[1] http://www.nagios.org/
[2] http://pywebsvcs.sourceforge.net/
[3] http://www.nagiosexchange.org/