SEVENTH FRAMEWORK PROGRAMME

CAPACITIES

Research Infrastructures

INFRA-2007-1.2.1 Research Infrastructures

DRIVER II

Grant Agreement 212147

"Digital Repository Infrastructure Vision for European Research II"

Monitoring Tools Specification

Deliverable Code: D7.3

Document Description

Project	
Title:	DRIVER, Digital Repository Infrastructure Vision for European Research II
Start date:	1st December 2007
Call/Instrument:	INFRA-2007-1.2.1
Grant Agreement:	**212147**

Document	
Deliverable number:	D7.3
Deliverable title:	Monitoring Tools Specification
Contractual Date of Delivery:	1st of May 2008
Actual Date of Delivery:	15th of July 2008
Editor(s):	CNR
Author(s):	Paolo Manghi, Marko Milkulicic
Reviewer(s):	
Participant(s):	
Workpackage:	WP7
Workpackage title:	Enhancing Infrastructure Sustainability and Research Integration
Workpackage leader:	CNR
Workpackage participants:	NKUA, ICM, CNR
Distribution:	Public
Nature:	Deliverable
Version/Revision:	3.0
Draft/Final:	Final
Total number of pages: (including cover)	
File name:	
Key words:	Monitoring, orchestration, self-administration, services, DRIVER applications

Disclaimer

This document contains description of the DRIVER II project findings, work and products. Certain parts of it might be under partner Intellectual Property Right (IPR) rules so, prior to using its content please contact the consortium head for approval.

In case you believe that this document harms in any way IPR held by you as a person or as a representative of an entity, please do notify us immediately.

The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated in the creation and publication of this document hold any sort of responsibility that might occur as a result of using its content.

This publication has been produced with the assistance of the European Union. The content of this publication is the sole responsibility of DRIVER consortium and can in no way be taken to reflect the views of the European Union.

The European Union is established in accordance with the Treaty on European Union (Maastricht). There are currently 25 Member States of the Union. It is based on the European Communities and the member states cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice and the Court of Auditors. (http://europa.eu.int/)

DRIVER is a project funded by the European Union

Table of Contents

Index

Table of Figures

Index

Summary

The purpose of this document is to describe the monitoring tools and the monitoring activities carried on for the D-NET v1.0 production infrastructure.

Introduction

1.1 Purpose of this document

The purpose of this document is to describe the monitoring tools designed for the D-NET v1.0 production infrastructure. These tools offer automatic and semi-automatic functionalities for controlling Quality-of-Service parameters of any Service in the infrastructure, but also to control the coherency of service interaction ans side effects when services are combined to deliver a DRIVER Application. To this aim, we shall first describe the kind of activities we are interested in and then present the tools we adopted to carry them on.

1.2 Document Outline

Section 2 describes the monitoring activities, while Section 3 presents in details the features of the tools *Munin* [2] and *SmokePing* [3], installed and configured in order to achieve the expected results. Section 4 describes

2 Monitoring Activities

2.1 Monitoring typologies

We recognized two main monitoring typologies: *service monitoring* and *application monitoring*.

2.1.1 Service monitoring

Service monitoring activities can be categorized into two main typologies, checking on different aspects of QoS: *host internal status* and *reachability*.

Host internal status monitoring

Services run on hosts which also run other software. Even if the host is completely dedicated for a given service, there is still system software that is required to run on it in order to provide basic operating system functionality. This external software uses resources which may affect the performance and the behavior of the service. Thus, a monitoring system has to be able to measure the heath status of the whole host in order to be able to detect possible malfunctions and react to them in a timely manner.

Host-wide measurement can be done in the following areas:

- Memory usage (ram, swap, cache)
- CPU usage
- Disk throughput
- Network throughput
- File System usage
- Entropy (for cryptographic use)
- Number of running processes.

By continuously following the status of these parameters, the monitoring system can detect potentially anomalous situation and react to them. The monitoring system's ability to follow the change of the parameters during time and to gain informations from their dynamic nature, is the key ingredient in adaptive and intelligent automatic resource management, which has to be able to recognize dangerous patterns and minimize false positives.

Reachability monitoring

Hosts can be perfectly healthy yet the services running on them cannot correctly interoperate with the rest of the infrastructure because of subtle networking problems which are normally hard to detect.

IP networks are best effort networks, and even the best links have some amount of packet loss which sometimes can interfere with the service interoperation. Because of their nature, these problems are very hard to detect.

One of the duties of a monitoring system is to be able to pinpoint systematic problems in network communication and also be able to automatically detect when those problems have been solved.

Independent monitoring tools check the reachability of services and keep every service informed about node failures. Services can thus avoid costly time-outs arising from attempts to contact unavailable services.

2.1.2 Application monitoring

As mentioned above, application monitoring is rather focused on the specific functionalities and on the specific properties expected by an application running over the infrastructure. In DRIVER, applications are "declared" in terms of (*i*) service *orchestration,* i.e. a number of actions the available services are supposed to perform in order to maintain certain functional conditions valid, and *(ii*) in the specific configuration of service orchestration, whose parameters may change from application to application.

3 Service Monitoring

In order to perform the monitoring activities underlined in section 2.1.1 two low-level tools were chosen: Munin [2] and SmokePing [3], to be described in the following. Such tool enable distributed control of QoS parameters, thereby alerting administrators when services fail to provide reasonable QoS or are in the process to do so. To some extent, such tools can be integrated with DRIVER Manager Services, in order for the service to take adjustment measures so as to limit the need of human intervention.

3.1 Munin

Munin is a lightweight monitoring tool, whose modular client-server architecture can be seen in Figure 1. It is able to efficiently fetch informations from hosts, produce graphs from the gathered informations and trigger alerts whenever the measured parameters go beyond a given threshold. More specifically, Munin *servers* fetch performance-counters from *munin nodes* which gather raw data from the system through the use of various *munin plugins*.

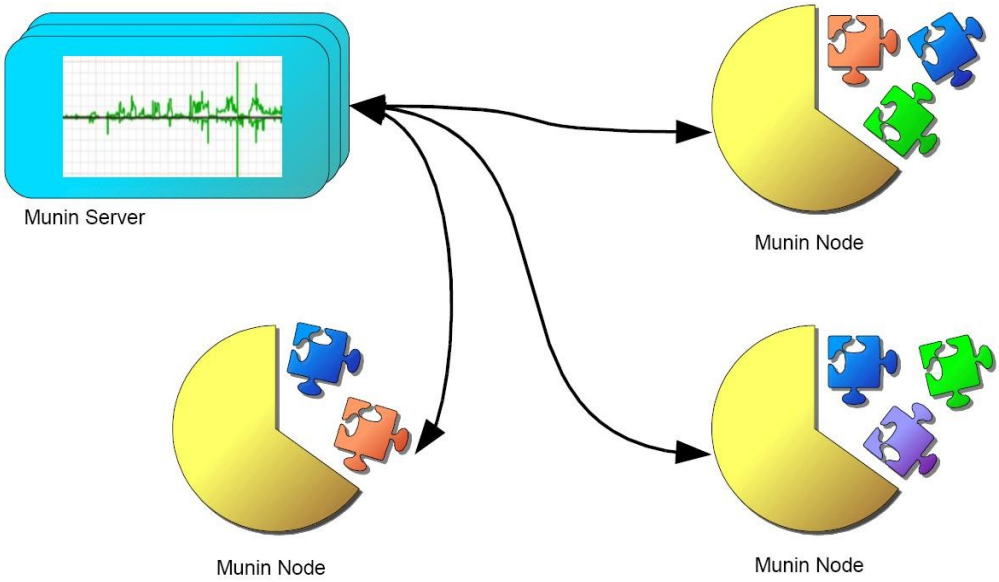

Figure 1: Munin architecture

Plugins

Munin offers a number of plug-ins for obtaining various system performance counters:

- **cpu usage**: categorized in *system, user, nice, idle, iowait, softirq*
- **cpu temperature**
- **entropy**: used for cryptographic-grade random number generation
- **load average**: unix-style load average
- **memory usage**: categorized in *apps, page_tables, swap_cache, cache, buffers, unused, swap, committed, mapped, active, inactive*.
- **swap in/out**: paging activity in pages per second.

- **IOStats**: disk IO, block per seconds
- **filesystem:** free space, open files
- **tomcat jvm memory**: categorized in *free, total, max*
- **networking**: traffic throughput, number of connections (categorized in *active, passive, failed, resets, established*)
- **MySQL, PostgreSQL**: categorized in *number of connections, number of queries*

Reporting

Munin servers store the data gathered from Munin clients data using *RRDTool* [4] (a "Round Robin" database providing detailed stats graphs for human consumption) and display examples of such graphical information, specifically relative to Memory Usage (see Figure 2 for an example) and Load Average of the original hosts.

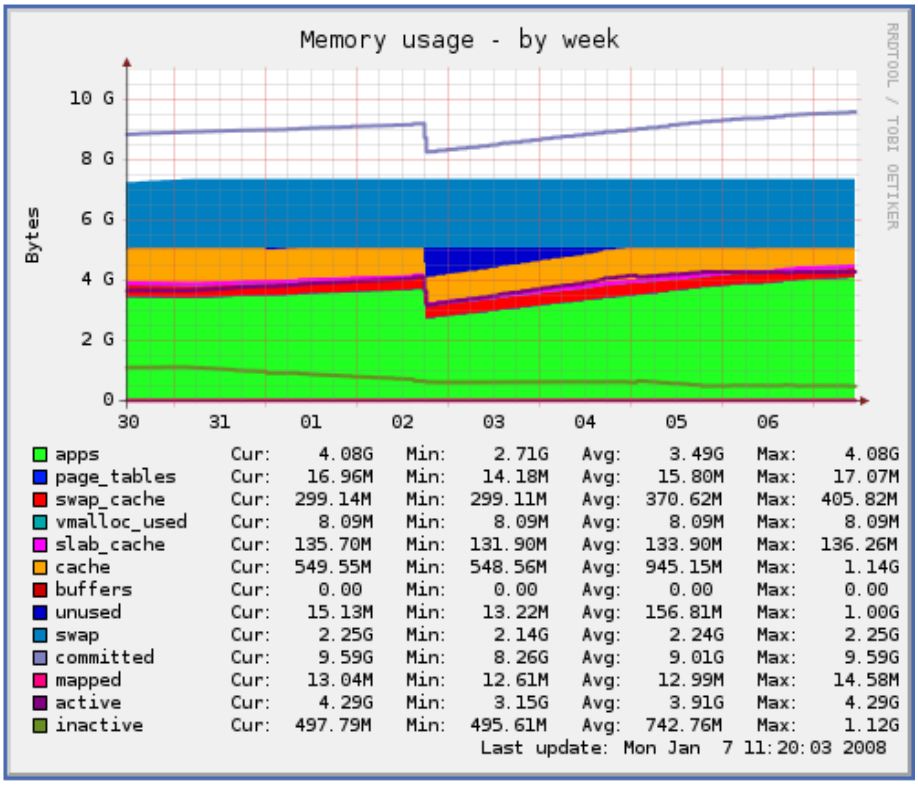

Figure 2: Munin Memory Usage graph

3.2 SmokePing

SmokePing is a latency measurement tool. It can measure, store and display latency, latency distribution and packet loss. It can be configured in a Master/Slave setup, in order to measure packet loss from different sources.

Master/Slave

SmokePing is able to probe a single target from multiple locations.

Probes

Despite the name, SmokePing does not only use ICMP echo packets to check reachability. As frequently happens, the system administrators may put firewalls which filter ICMP echo

traffic. SmokePing handles this situation by allowing the choice of the probing method through modular plugins. Here are the most important ones:

- Smokeping::probes::FPing
- Smokeping::probes::EchoPingHttp
- Smokeping::probes::SSH

Each probe can be configured with several parameters. The FPing probe, for example, can be configured to send big (1k) ICMP datagrams, which are more likely to suffer eventually from similar problems like real SOAP TCP traffic.

As a practical example, during the initial testbed deployment one of the switching hubs in one of the facilities hosting the DRIVER servers was misbehaving. The problem caused the dropping of 1% of the small packets (TCP handshaking packets and standard 56 byte ICMP packets used by "ping"), but the rate was as high as 68% for 1k sized packets which were used in the real SOAP payload. The problem would have been very difficult to pinpoint manually because the machine responded normally to standard "pings", and the remote SSH shell worked quite well, but SOAP communication was very slow and usually ended in timeouts. The monitoring service reported the problem and the services were quickly disabled during the outage.

Matchers

SmokePing collects the data and analyzes it using one or more "matchers". The matchers trigger an alert whenever the data is changing in a suspicious way, taking in consideration the past trends and using different algorithms:

- Smokeping::matchers::Avgratio: detects changes in average median latency
- Smokeping::matchers::CheckLatency: triggers alert to check latency is under a value for x number of samples
- Smokeping::matchers::CheckLoss: triggers alert to check loss is under a value for x number of samples
- Smokeping::matchers::Median: finds persistent changes in latency
- Smokeping::matchers::Medratio: detects changes in the latency median. By looking at the median value this matcher is largly imune against spikes and will only react to long term developments.

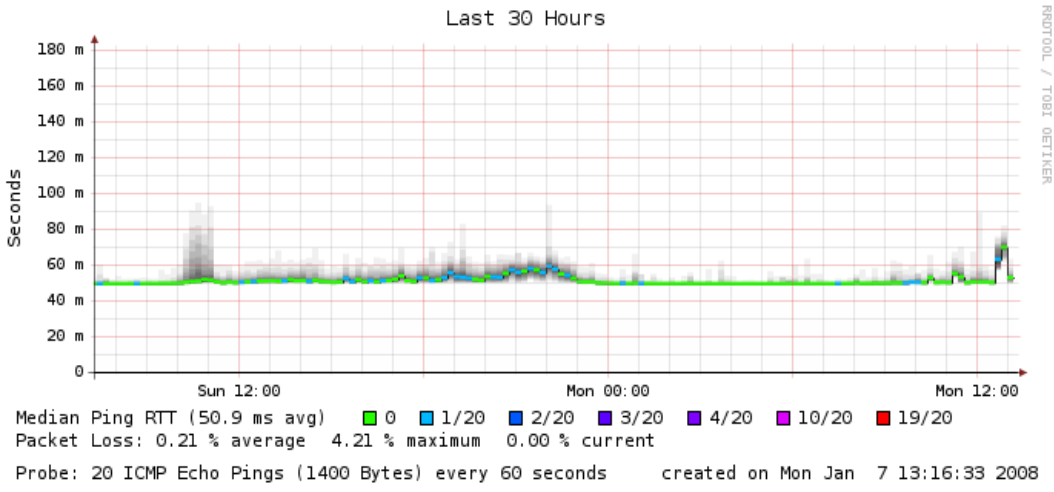

Figure 3: Smoke ping latency and packet loss graph

The administrator can take advantage of the SmokePing recording of the sampled data by looking the graphical representation (see Figure 4 for an example). This information is very precious in order to pinpoint problems that are not automatically detected by the monitoring service, especially during the monitoring system parameters configuration and fine tuning, but also in order to check whether the monitoring service is behaving correctly when taking automatic decisions in reaction to alerts generated by the SmokePing matchers. It is also useful as an "after the fact" analysis tool which can be helpful to provide hints to solving the original cause of the problem.

3.3 Integration with the Manager Service

The Manager Service (see Figure 4) generates the configuration files for the low-level tools, and schedules the execution of the sampler daemons. Low-level tools in the Integration Layer send alerts to the Manager Service, which takes decisions based upon the kind and the severity of the alert.

The possible decisions are currently:

- Lowering the priority of the service (when the machine is highly loaded, consumes a lot of IO or has a high network latency[1]);
- Disabling (temporarily) the whole node, with all services on it;
- Sending alerts to the administrators, when critical human intervention is needed.

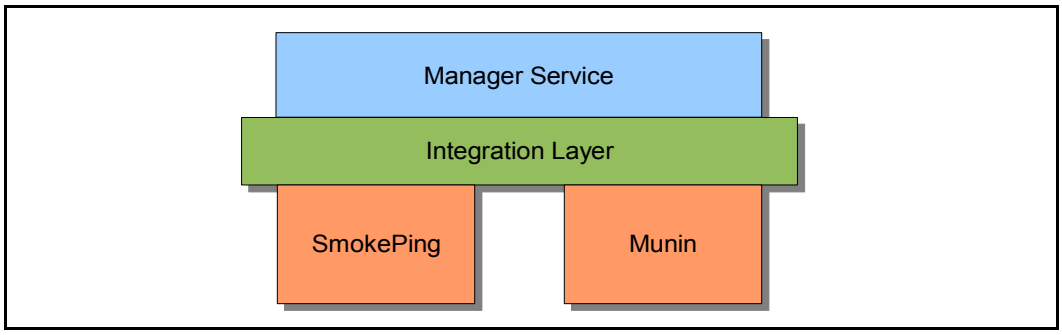

Figure 4: Manager Service Low-Level tools integration

[1]Especially for the higher level protocol probes, like SSH and HTTP, as ICMP usually is answered by the OS kernel and may be answered quickly also by heavy loaded machines.

4 Application Monitoring Activities

4.1 Introduction

Application Monitoring takes care of checking that the internal state of a DRIVER application is consistent.

An application consists of a set of rules, firing actions whose goal is that of preserving certain conditions, specialized by custom application parameters. The set of possible rules and parameters is application specific and encoded by one dedicated Manager Service, i.e. one Manager Service handles a specific DRIVER application. Note that different instances of the same Manager Service give life to different applications, each offering the same functionality potentially implemented over the same pool of shared services, but with different parameter-set and behavior.

In the following we shall describe the orchestration rules and relative parameters encoded by the Manager Service of the *Aggregation System Application.* An instance of such application is currently running on the DRIVER Production infrastructure under the name *DRIVER Information Space*, orchestrated by a specific instance of the Manager Service.

4.1.1 Aggregation System Application

The Aggregation System Application orchestrates the DRIVER Services in order to:

1. enable harvesting of external OAI-PMH repositories;
2. persist the harvested metadata records, together with copies of raw data referenced from within the records, inside store units allocated within the infrastructure;
3. enable processing of stores containing harvested data in order to transform, clean and enrich it so as to produce high quality uniform information, in turn stored into other local stores;
4. deliver processed data from stores to index services for indexing.

The application also includes user interface services and search services capable of answering user queries by means of the available index services .

The rules involved are:

1. whenever a new repository is created: (*i*) new store units must be allocated to host the data to be harvested from the repository and (*ii*) the repository should be "assigned" to a transformation service, so that aggregator manager administrators can define the processing rules to be applied to the harvested data in subsequent processing operations;
2. whenever the processing rules for a repository are provided by an aggregator manager, the repository data is ready to and must be harvested;
3. whenever an harvesting operation relative to a repository is finished, new store units must be allocated so as to host the processed data;
4. whenever store units at point 3 are created, the processing operation can take place;
5. whenever a processing operation is terminated, the data from the store units involved must be delivered to the index services available;

6. whenever fresh data is harvested from a repository and placed into the relative store units, both data processing and indexing operations must be performed in order to update the Information Space and align the user interfaces and search services with the current harvesting scenario.

The rules above can be instantiated with robustness parameters defining replica management specifics. All the data is kept replicated on a number N of different store units and M of different indices. Furthermore, this process takes into consideration the physical location of the replicas in order to reduce the probability of a service outage due to local issues such as connectivity, fire, theft, etc.

4.1.2 Aggregation System Manager Service

The Manager Service is made two main modules:

- *Monitoring module*: constantly checks whether the conditions established by the rules above are satisfied and, when this is not the case, will as the Orchestrator Module to fire the corrective actions so as to reach the intended state.
- *Orchestrator Module*: uses the information provided by the Monitoring Module and by the Service Monitoring activities (Section 3.3) and creates a sequence of lower level tasks which are then executed taking in consideration various aspects such as load balancing, throttling, task duplication removal, resource discover and allocation, etc.

For example, the Application Monitoring logic may detect that a given repository has not enough replicas and will schedule a task to the orchestrator in order to fix the situation. The orchestrator's ability to recover from errors is deliberately reduced in favor of giving to the application monitoring the responsibility to trigger a retry. This separation of concerts allows for a good degree of error recovery while simplifying the overall design.

When corrective actions cannot be taken automatically, special administration user interfaces can be used to spot and manually fix inconsistencies. For example, Figure 5 shows the administration interface used to verify that repositories reached a consistent state according to the specific instantiation of the Aggregation System Application, where N=M=1 (no replicas). The picture reveals that the repository status is consistent with the Manager Service configuration:

- one store unit created for its Dublin Core records (*oai_dc* format) with no index available (not needed in this application, where we only index DMF records);
- one store unit created for the processed data (DMF, *DRIVER Metadata Format*) with one index available.
- store units and index content are aligned, i.e up-to-date with the last harvesting operation.

If any of those condition would not be satisfied, relative error messages would be displayed, together with buttons enabling the corrective actions: create store, create index, create replica, process data, etc.

Repository Verification

Repository Name	memSIC : Memoires en Sciences de l'Information et de la Communication	**Number of elements**	147
Aggregated by:	212.87.15.95:9001	**Harvesting Instance:**	show profile

MD Format	MD-Store: 212.87.15.95:9002
oai_dc	**MDStore [HARV]:** 147 element(s) *No Index created*
DMF	**MDStore [AGGR]:** 147 element(s) **- Index 1:** 147 element(s)

Figure 5: Repository Verification

5 Future issues

Currently, only a subset of the corrective actions can be automatically performed by the Manager Service. Next steps in refining the Manager Service for Aggregation System Applications are those of stressing the encoding of self-administration mechanisms into the Service.

References

[1] DRIVER Annex I - "Description of Work", Proposal no. IST-034047.

[2] Munin Project. http://munin.projects.linpro.no

[3] SmokePing Project. http://oss.oetiker.ch/smokeping

[4] RRDTool Project. http://oss.oetiker.ch/rrdtool