Report of the Laboratory of Parallel and Distributed Systems
Computer and Automation Research Institute of the
Hungarian Academy of Sciences
H-1518 Budapest, P.O.Box 63, Hungary

Comparison of Representative Grid Monitoring Tools

Zoltán Balaton, Peter Kacsuk, Norbert Podhorszki and Ferenc Vajda

LPDS-2/2000

2000

Comparison of Representative Grid Monitoring Tools*

Zoltán Balaton, Peter Kacsuk, Norbert Podhorszki and Ferenc Vajda

Laboratory of Parallel and Distributed Systems
Computer and Automation Research Institute of the
Hungarian Academy of Sciences

{balaton, kacsuk, pnorbert, vajda}@sztaki.hu

Abstract. The report provides a short description of the grid monitoring architecture and existing event services. It deals with five monitoring tools that can be used in a grid environment, namely NetLogger, Globus HeartBeat Monitor, Network Weather Service, Autopilot with Virtue and a prototype implementation of the grid monitoring architecture at NASA. The comparison of the tools is based on the requirements of scalability, intrusiveness, validity of information, data format, extendibility, communication, security and measurement metrics.

1 Introduction

Over the last decade, collection of distributed resources, which act as a single computing platform, i.e. Computational Grids, have become an increasingly important alternative to parallel computers and MPPs. These platforms address

- the needs of resources intensive applications,
- applications which couple remote instruments with computational sites,
- applications which require huge amounts of data from distributed archives,
- applications which can profit from harvesting free cycles from a large number of available resources.

The grid technology gave birth to a couple of new applications and technologies as teleimmersion, collaboratory, etc. Recently, based on the grid technology both new extended research areas of Global Computing (as co-operation of autonomous and mobile entities in dynamic environments) and Peer-to-Peer Computing (a set of technologies enabling the direct exchange of services or data between computers particularly in business environments supported by an industry group of many leading companies as Intel, IBM, HP, etc.) appeared.

Computational grid has become critical platform of choice for a wide spectrum of high performance and resource-intensive applications. However, achieving performance in the environment of the grid is often difficult. The grid itself does not provide a stable platform. Its environments are dynamic, heterogeneous and shared by multiple users with computing resource demands,

Grid-oriented research and development can be grouped into different categories, e.g.:

- performance engineering
- resource management
- scheduling
- internet traffic engineering (performance measuring and data analysis of the computing fabrics)

The ability to monitor (and manage) distributed computing components is critical for enabling high-performance distributed computing.

* This work was supported by a grant of the Hungarian Scientific Research Fund (OTKA) no. T032226 and by the CERN DataGrid project (IST-2000-25182).

2 A grid monitoring service architecture

Large distributed systems such as a Computational Grid require a large amount of monitoring data for a variety of tasks as:

- performance analysis
- performance tuning
- performance prediction
- scheduling
- fault detection and diagnosis

Computational Grids require timely and accurate dynamic performance information. Ensuring that all necessary monitoring is turned on and the data is being collected can be a very tedious and error-prone task but the ability to monitor and manage distributed components is critical for enabling high-performance distributed computing. As distributed systems such as Computational Grids become bigger, more complex and more widely distributed, it becomes important that this monitoring and management is automated.

If the monitoring architecture is sufficiently general it could be adapted for use in a distributed environment other than a Grid. For example, it could be used in large compute farms or clusters.

The Grid monitoring service architecture proposed by the Global Grid Forum (see [1]) could consist of the following components (see Figure **1**):

2.1 Sensors

A sensor is any program that generates a time-stamped performance-monitoring event.

For example,

- CPU usage
- memory usage
- network usage.

Sensors also can be used to monitor error conditions, e.g.

- a server process crashing
- CRC errors on a router

Sensors can be grouped according to the following categories:

a. Host sensors
These sensors perform any host-monitoring task, e.g.

- CPU load
- available memory
- TCP retransmission

They can run remotely from the host being monitored (SNMP: Simple Network Management Protocol-based monitoring).

b. Network sensors
They perform SNMP queries to any network device, e.g.

- a router
- a switch

c. Process sensors
They generate events when there is a change in a process status, e.g.

- it starts
- it dies normally
- it dies abnormally
- some dynamic threshold is reached (e.g. number of users over a certain time period)

d. Application sensors
These sensors are embedded inside the applications. They might generate events in different conditions, e.g.

- user connect/disconnect
- change of user password
- static threshold is reached
- UNIX signal received
- other user-defined event

2.2 Sensor manager

The sensor manager is responsible for starting and stopping the sensors. The sensor manager determines which sensor to run that can be defined

- from a configuration file
- from request by a sensor manager GUI

2.3 Event consumers

Any program that requests data from a sensor is an event consumer. It can run on the same host as the sensors or may be remote.

Consumers can be categorised to the following types:

a. Archiver
It is used to collect data for archive service. It subscribes to the event suppliers, collects the event data and places them in the archive. To indicate the contents of the archive it creates an archive directory service entry.

b. Process monitor
This consumer can be used to trigger an action based on an event from a server process. For example,

- it might run a script to restart the process
- send an e-mail to a system administrator

c. Real-time monitor
It uses real-time analysis tools to collect monitoring data in real-time. It checks the directory service to see what data are available and then subscribes to all sensors it is interested in. The sensors then send data to the consumer as it is generated. Data from many sensors are then merged into a file for use by programs as a real-time event visualisation tool.

d. Overview monitor
This consumer collects information from sensors on several hosts. It uses the combined information to make some decision that could not be made on the basis of data from only one host (e.g. to inform the system administrator that both the primary and backup servers are down).

2.4 Directory service

The role of the directory service is to publish the location of all sensors and other event suppliers. This is the basis for the consumers to discover the monitoring data that are currently available for subscription. The necessary basic functionality for this is provided by query-optimised directory services as

- LDAP (Lightweight Directory Access Protocol)
- Globus MDS (Metacomputing Directory Service)
- Legion Information Base
- Novell NDS (Novell Directory Service)

LDAP v3 version provides a special service called „event notification" which notifies a client when an entry (e.g. a sensor running) becomes available or is updated. (Current implementations of LDAP servers do not work well in an environment with many updates.)

2.5 Event suppliers (producers)

They are responsible for keeping the sensor directory up-to-date and are listening to data requests from event consumers.

An event supplier can also be configured to compute summary data (e.g. average CPU usage for a given time period) and publish it for the directory service (for the CPU sensor entry).

Event suppliers can accept different types of requests from event consumers. A consumer may request all event data or only certain types of events or an event to be sent only if its value is more than a certain value. (e.g. CPU load is greater than 60 % or it changes by more than 30 %).

In a special query mode the consumer does not open an event channel but requests the most recent event only.

2.6 Event archive

To provide ability to do historical analysis of system performance it is important to archive event data. By this way it is possible to determine when and where changes occurred. Naturally, it is not desirable to archive all monitoring data but good samplings of both „normal" and „abnormal" system operations.

Data of the archive might be also used for performance prediction (e.g. NWS = Network Weather Service). Archive can be considered just another consumer.

The grid performances monitoring facilities must be scalable where thousands of resources and tens of thousand of grid users - using the grid simultaneously - might be present. Therefore, monitoring facilities should be able

- to scale with the Grid
- to allow dynamic sensors' joining and leaving
- to provide valid performance information.

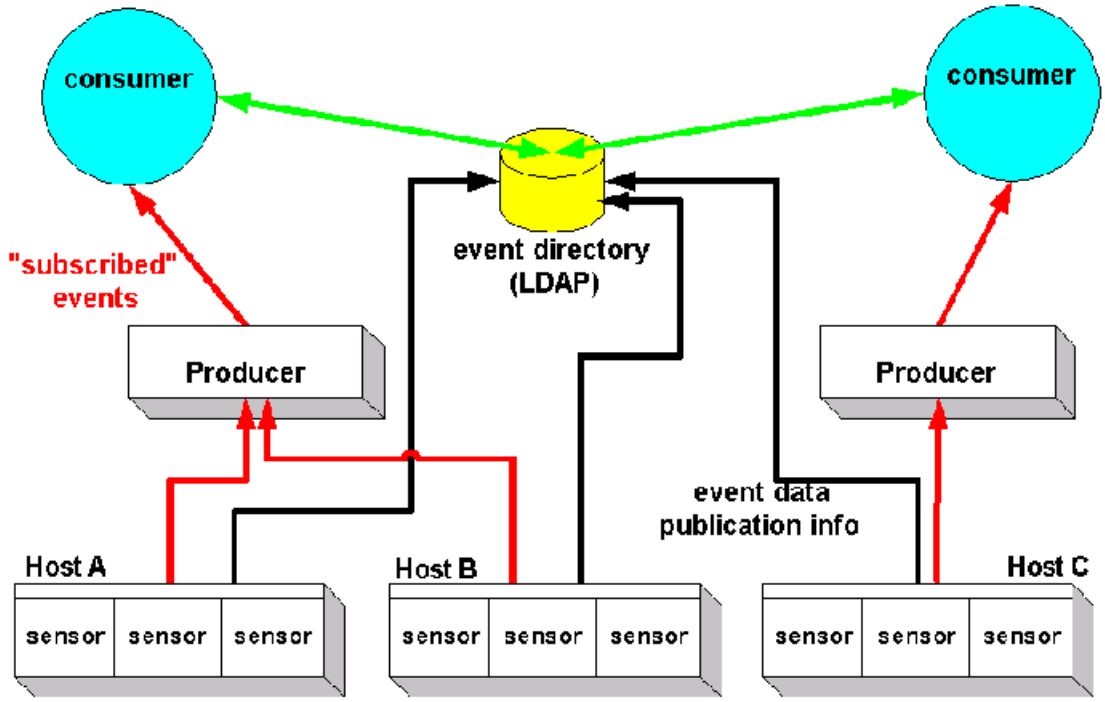

Figure **1**. Grid Monitoring Architecture proposed by the Global Grid Forum

3 Brief survey of event services

There are many existing systems with an event model. Here only short descriptions of the most important systems are provided. A survey of event services can be found in [2].

CORBA (Common Object Request Broker Architecture).
CORBA includes an event service (System Management: Event Management Service). It has a rich set of features (including push or pull events and ability for the consumer to pass a filter to the event supplier).

DCOM (Distributed Component Object Model)
Microsoft's DCOM supports communication among objects implemented on different computers connected by a LAN: Local Area Network, WAN: Wide Area Network or the Internet).

Jini
Sun's Jini has a „Distributed Event Specification". It is a simple specification for how one JVM (Java Virtual Machine) can register interest in the occurrence of an event in an object of some other JVM. Then the object receives a notification when that event occurs.

CCA (Common Component Architecture)
It is a tool from Indiana University Extreme Lab. This tool allows components to publish and subscribe to events. The event model may be a point-to-point generator/listener model or it may a push-based publisher-channel-subscriber model that does type filtering.

Kerix
Kerix is an Internet Notification Service from HP (Bristol Laboratory, Australia). It is based on a consumer/producer-publishing model. It includes an event cloud, event distributors and contains filters as well.

MQSeries
The IBM MQSeries family provides an open scalable information infrastructure centered on its messaging middleware.

Tuxedo
Tuxedo is a middleware transaction monitor from BEA System. It is a platform mainly for building E-commerce systems.

Gryphon
This system by IBM (T.J. Watson Center) is an advanced content-based publish/subscribe system.

Elvin
Elvin is a publish/subscribe Notification Service from Distributed Systems Technology Centre (Australia). It handles nominal event volumes over 10000 per second and up to 1000 clients for a single server.

BLIP (Basic Lightweight Information Protocol)
It also provides an application-level message service based on the publish-subscriber model. Notifications are delivered in real-time streams.

VisiBroker
It is an object request broker event service by Borland. It includes a new version for Java on the Linux platform.

InfoBus
InfoBus from Sun Microsystems enables dynamic exchange of data between JavaBeans component architecture. Its protocols are based on a notification of information bus. Components can exchange data with other component in a structured way (including arrays, tables, etc.).

Siena (Scalable Internet Event Notification Architecture)
It is a research project by University of Colorado aimed at designing and constructing a generic scalable event service.

Pablo toolkit
It is a portable performance analysis environment which combines portable performance data analysis with portable source code instrumentation and data capture.

4 Review of representative monitors

4.1 GRM & PROVE

GRM is a semi-on-line monitor (see [3]) that collects information about an application running in a distributed heterogeneous system and delivers the collected information to the PROVE visualisation tool (see [4]). The information can be either event trace data or statistical information of the application behaviour. Semi-on-line monitoring means, that any time during execution all available trace data can be required by the user and the monitor is able to gather them in a reasonable amount of time.

GRM consists of three main components (see its structure in Figure **2**):

1. **Client library**
   The application is instrumented with the functions of the client library (currently available in the C language). Event formats should be defined first and then event records can be generated by a simple funtion call giving data as funtion parameters. Both trace events and statistics can be generated by the same instrumentation.

   An application process places trace event records into a shared memory buffer provided by the local monitor and does not need to communicate outside of the host.

2. **Local monitor process**
   A local monitor is running on each host and it is responsible for the handling of trace of processes on the same host. It creates a shared memory buffer and event records are directly placed into this buffer by the processes. Thus, event at process failure, all trace events are available for the user to the point of the failure.

   In statistics collection mode, the shared memory buffer is used to store the counters and the local monitor is responsible for generating the final statistics data in an appropriate form.

3. **Main monitor process**
   A main monitor process is co-ordinating the work of the local monitor processes, collects trace data from them when the user asks or a trace buffer on a local host becomes full. Trace is written into a text file in Tape/PVM format which is a record based format for trace events in ASCII representation. Also, it performs clock synchronisation among the hosts.

GRM is designed for application performance monitoring and it can handle high volume of data. Host and network monitoring sensors can be easily connected to GRM using its client library that enables sensors to define new type of events. Besides, sensors can put their events into the shared memory buffer themselves (using the client library). However, sensors should be set up and stop manually. There is no support for sensor management in GRM. Using host and network sensors application performance can be examined in the context of the execution environment.

PROVE has been developed for performance visualisation of Tape/PVM trace files. It supports the presentation of detailed event traces as well as statistical information of applications. It can work both off-line and semi-on-line and it can be used for observation of long-running distributed applications. Users can watch the progress of their application and realise performance problems in it.

PROVE communicates with the main monitor of GRM and asks for trace collection periodically. It can work remotely from the main monitor process. With the ability of reading new volumes of data and removing any portion of data from its memory, PROVE can observ application for arbitrary long time.

GRM and PROVE are available as part of the *P-GRADE* graphical parallel program development environment. GRM and PROVE can be used separately from the package and installed on grid in the structure of Figure **2**. Information, tutorial and papers about P-GRADE can be found at

http://www.lpds.sztaki.hu/projects/p-grade

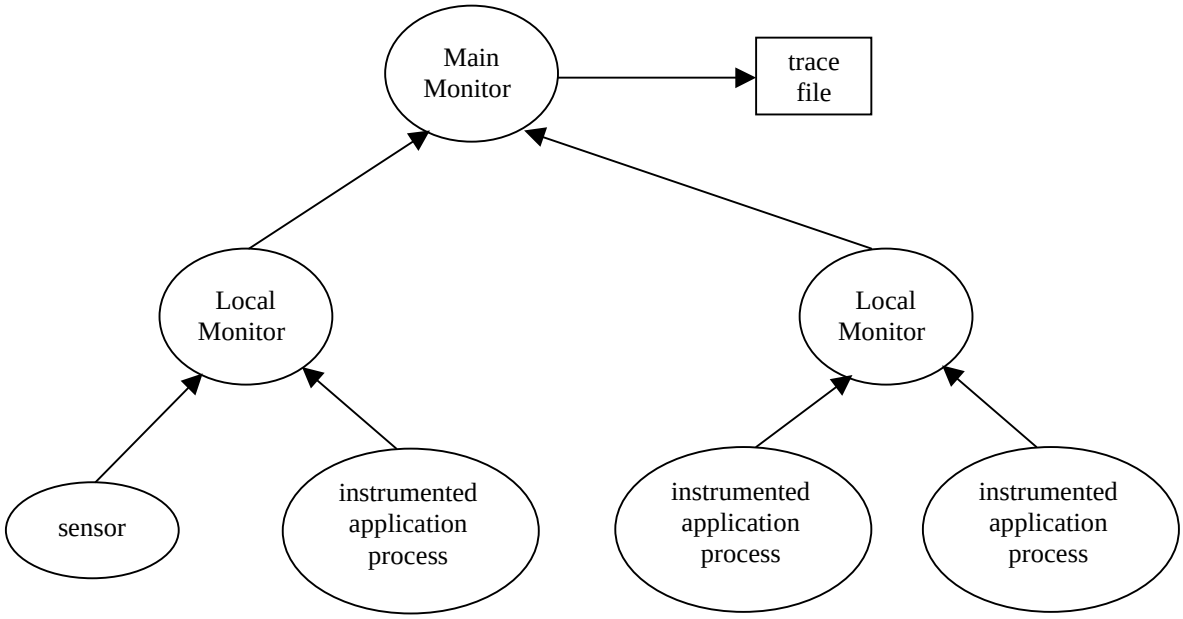

Figure **2**. Structure of GRM

4.2 NetLogger

NetLogger is a distributed application, host and network logger, see [5] and [6].

Main suggested application areas are:

- performance and bottleneck analysis
- selecting hardware components to upgrade (to alleviate bottlenecks)
- real-time and post-mortem analysis of application
- correlating application performance with system information

Components:

1. Message format
IETF (Internet Engineering Task Force) ULM (Universal Logger Message) format. Used for logging and exchange message.

2. API (Application Programmer Interface)
It links the application with the NetLogger library (currently available in the following languages: Java, C, C++, Perl, Python, Fortran, Tcl). It uses simple calls (e.g. NetLoggerWrite) which get timestamp, builds NetLogger message and sends it to destination.

3. Client library
Using the library one can send log messages to a local file, a remote host or local memory. It also includes wrappers for several systems monitoring utilities.

4. Host and network tools
Typical network tool examples:

- host monitoring
- router monitoring
- ATM switch monitoring

Monitoring tools that can be used wrapped with programs:

– netstat (used e.g. TCP retransmission)
– vmstat (used e.g. system load)
– iostat (used e.g. disk activity)
– ping

5. Visualisation tool
It is used for interactive graphical representation of system-level and application-level events. The visualisation tool can display several types of events at once. It is user configurable and can play, pause, rewind, provides slow motion, zoom, etc. It can run post-mortem or in real-time.

It uses three types of primitives:

- *lifeline*
  It represents the „life" of an object (datum or computation) as it travels through a distributed system. The slope of the lifeline gives a clear visual indication of latencies. Each object has a unique identifier.
- *loadline*
  It connects a series of scaled values into a continuos segmented curve. Most often it is used for representing changes in system resources (e.g. CPU load or free memory).
- *point*
  It is used to graph single occurrences of events (e.g. error or warning conditions). It can be scaled to a value producing in a scatterplot.

NetLogger toolkit components, User Guide, Tutorial, APIs, papers and talks on NetLogger etc. can be downloaded from *http://www-didc.lbl.gov/NetLogger/*

A more detailed structure of NetLogger is shown in Figure **3**, where *netlogd* is a data logging daemon, that writes trace data into the trace file. Sensors and instrumented application processes can generate events in ULM formats using the client library of NetLogger and send them to the netlogd daemon. Sensors can run remotely from netlogd and send data through the network.

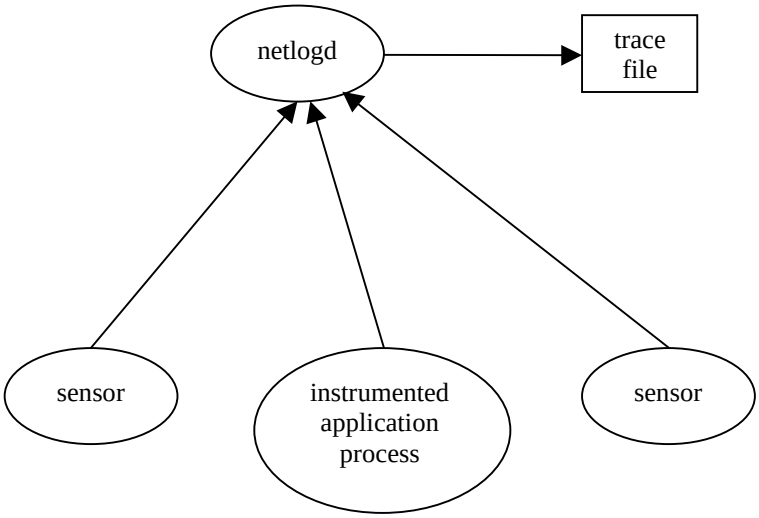

Figure **3**. Structure of NetLogger

4.3 Heartbeat Monitor (Globus)

The Globus Heatbeat Monitor (HBM, see [7]) was designed to provide a mechanism for monitoring the state of processes. It detects and reports the failure of processes identified to the HBM. It allows simultaneous monitoring both Globus system processes and application processes. To distinguish

process failure from other failure events (e.g. network partitioning or host failure) the HBM detects process failure when the host and network connections are functioning properly. The HBM also provides notification of process status exception events for recovery actions to be taken. To ensure reliability and robustness the HBM has no dependence on other Globus components (e.g. MDS: Metacomputing Directory Service).

The HBM consists of three main components (see Figure **4**):

1. Client Library (HBMCL)
It is used to register each monitored client process with the unique local monitor on the same host. It can be used either as a library API that can be linked into the program for the monitored process or as an independent program that accepts as one of its parameters (the process id) of the process to monitor. It uses four procedures to activate/deactivate modules and register/unregister clients.

2. Local Monitor (HBMLM)
There is one HBMLM running on each host, checking and reporting the status of the monitored system and application processes on that host. Each Local Monitor periodically performs a review cycle in which it checks the status of the client processes it is monitoring. It updates its local status information and sends a report on each monitored process to one (or more) external Data Collector agents (specified at registration).

The main functions of the HBMLM are:

- initialisation
- process registration
- monitor/report cycle
- checkpointing
- process unregistration

3. Data Collector (HBMDC)
There can be any number of HBMDCs but typically one is used for tracking all the monitored processes associated with the metacomputing environment, plus one for each distributed application. Each Data Collector receives the reports sent by the Local Monitors and incorporates these reports into its local repository. It also infers the unavailability or failure of monitored components based on Local Monitor's reports that are expected but not received (time-out situations) and periodically adjust the status of client processes accordingly. The status information in each repository is checkpointed regularly. The HBMDC can recognise specific exception status changes and generate appropriate notifications via callbacks. The Data Collector API is a library of functions that performs monitoring and notifications of exception events. Procedures can be incorporated into programs that use the API notification callback mechanism to trigger responses to exceptional changes of the status.

Detailed information and program can be downloaded from *http://www.globus.org/hbm*

Infrastructure image is taken from: *http://www.globus.org/hbm/heartbeat_spec.html*

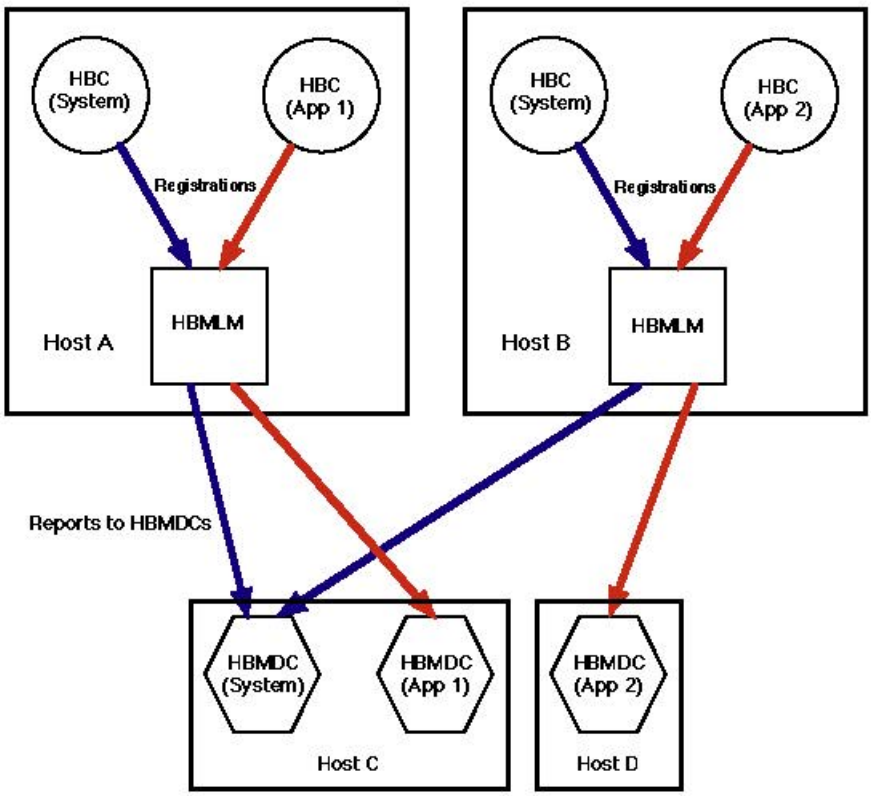

Figure **4**. Structure of HeartBeat Monitor

4.4 Network Weather Service (NWS)

Recently, a new methodology has appeared called Performance Prediction Engineering that addresses the problem of modelling time-sensitive, dynamic and heterogeneous performance information and using it to predict performance of distributed applications in a metacomputing environment. This methodology involves the design and development of

- structural models that consist of components representing the performance of computation and communication activities of applications
- performance forecasting for which a typical example is the NWS
- quality of information measures which can characterise the generated prediction with qualitative values as its „accuracy", the „overhead" of computing the prediction, etc.

The Network Weather Service (see [8] and [9]) is a distributed system that periodically monitors and dynamically forecasts the performance of various networks and computational resources can deliver over a given time interval. It operates a distributed set of performance sensors (CPU monitors, network monitors, etc.) from which it gathers readings of instantaneous conditions and then uses numerical models (mean-based, median based and autoregressive methods) to generate forecasts of what the conditions will be for a given time frame.

It forecasts process-to-process network performance (QoS readings as latency and bandwidth) and available CPU percentage for each machines that it monitors.

The NWS uses four component processes (see Figure **5**):

1. Persistent state process (*Memory*)
It stores and retrieves measurements from persistent storage. A persistent state must be able to survive the failure of a process memory. It provides a simple text string storage and retrieval

service where each string is to be associated with an optional time stamp. Data are fetched and stored in some permanent medium outside the NWS.

2. *Name server* process
It implements a directory capability used to bind process and data names with low-level contact information (e.g. address pairs, TCP/IP port number). Active processes must register their bindings periodically. Name service uses an implementation of the LDAP (Lightweight Directory Access Protocol). The address of the NWS Name Server Process is the only well-known address used by the system allowing both data and services to be distributed.

3. *Sensor* process time stamp
It gathers and sends for storing time-stamp performance measurements from a specified resource. Each sensor process may measure several different performance characteristics. For example, the TCP/IP sensor provides both bandwidth and end-to-end round-trip latency measurements (named and stored separately). The CPU sensor uses the one-minute measurement to calculate the fraction of the CPU occupancy time that a process would get if it were run at the moment the measurement was taken.

The NWS network sensor is capable of measuring network performance characteristics as small-message round-trip time, large-message throughput and TCP socket connect-disconnect time.

4. *Forecaster* process
It produces a predicted value of deliverable performance during a specified time frame for a specified resource. To generate a forecast, a Forecaster process requests the relevant measurement history from a Persistent State process. The persistent state is stored as a circular queue and ordered by time stamp. The measurements may then is treated as a time series for the purpose of forecasting. It applies a set of forecasting models to the entire series and dynamically chooses the forecasting technique that has been most accurate over the recent set of measurements. The model generating the lowest prediction error for the known measurements is used to make the forecast of future values. Application can retrieve performance forecasts via an (lightweight and portable) API that contacts the system via sockets.

Information on the current Network Weather Service is provided at: *http://nws.npaci.edu/NWS*

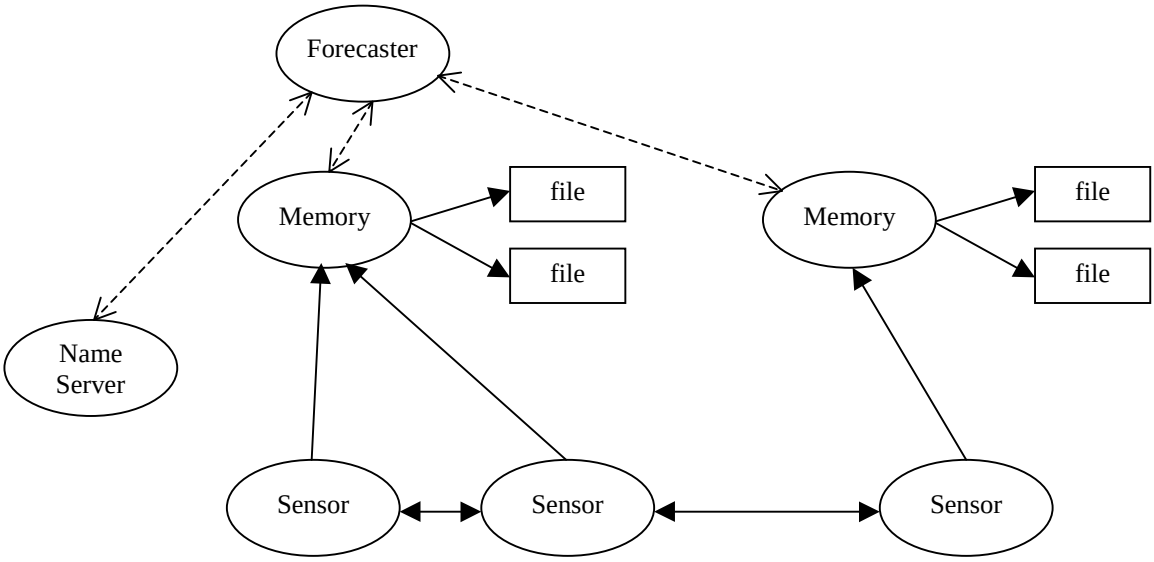

Figure **5**. Structure of NWS

4.5 Autopilot and Virtue (Pablo Group)

Autopilot is a distributed performance measurement and resource control system (see [10]) that is based on the Pablo performance toolkit ([11]). It is complemented by Virtue (see [12]) that is an environment which accepts real-time data from Autopilot and allows users to change software behaviour and resource policies.

Autopilot library containes the following components:

- distributed performance sensors
- software actuators
- behavioral classification tools
- self-defining data format (SDDF)
- decision procedures
- distributed name servers
- sensor and activator clients

Sensors and actuators are placed in source code (program variables and control points). They can work in either procedural or threaded mode. The infrastructure of Autopilot is shown in Figure **6**.

Autopilot provides performance daemons to capture network and operating system data on distributed hosts. Typical data include processor utilisation, disk activity, context switches, interrupt, memory utilisation, paging activity, network latencies. It includes a portable Java interface (called Autodriver) for sensor data display and actuator control.

Virtue is a collaborative virtual environment to manipulate the abstract world of software in the same way as a physical environment.

The Virtue toolkit includes the following components:

- hierarchical groups for software structure
- software component control
- manipulation tools to interact with the virtual environment
- multimedia annotation tools
- standard interfaces

Virtue and Autopilot work together to allow users to monitor and steer distributed computations in real-time. In the context of computational grids, Virtue defines a visualisation hierarchy for managing performance details. Using data from Autopilot daemons, users can judge communication network latencies, site utilisation and network routing. They can get insights about communication and computation delays. The Virtue 3D display allows a variety of statistics mapped to different graphs defined in different levels of the system hierarchy (from distributed site to single tasks).

Virtue is using a generalised magnifying glass (called magic lens) which allows users to interactively focus on a portion of the visualisation. A set of actualisation interfaces allows users to directly manipulate visualisation and to change software behaviour (e.g. changing cache size).

See details on the Pablo Group home page: *http://www-pablo.cs.ninc.edu*

Image is taken from:

http://www-pablo.cs.uiuc.edu/Publications/Presentations/Autopilot%20Overview/sld036.htm.

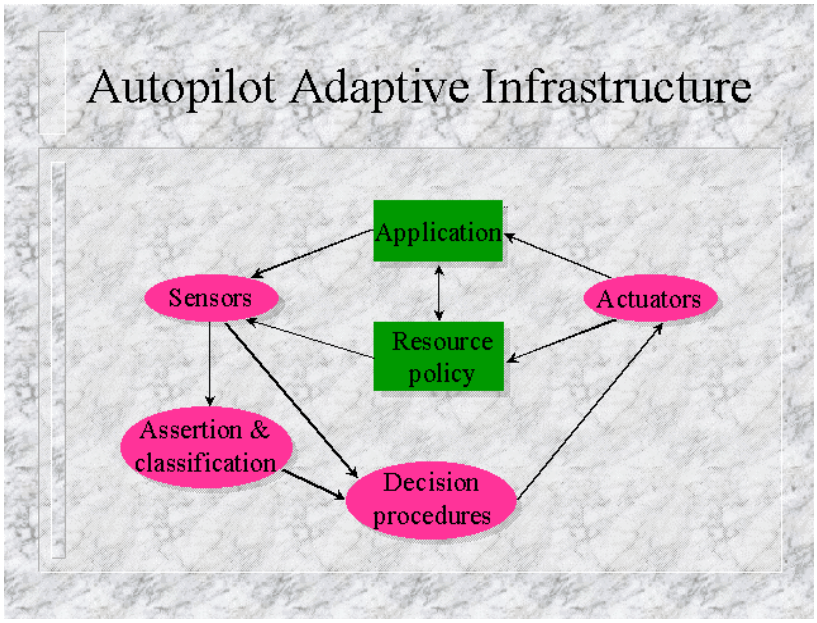

Figure **6**. Adaptive Infrastructure of Autopilot

4.6 Information Power Grid (NASA-IPG) Monitoring Infrastructure

The grid-monitoring infrastructure developed at NASA (see [13]) includes three basic components (see Figure **7**):

1. Sensors
External sensors typically execute one Unix utility (e.g. df, ps, ping, vmstat, and netstat) and extract sensor-specific measurements. Internal sensors are also available which can collect resource-usage information from within the calling process.

2. Actuators
They use the shell to perform configuration, process control or other user-defined tasks. (They can be invoked in a similar fashion to sensors.)

Typical actuators: kill process, send mail, execute a shell command, LDAP (Lightweight Directory Access Protocol) server, queries, Globus commands.

3. Event Service
It supports a publisher-subscriber paradigm. It facilitates clients' request for specific information and servers to forward that information. It also provides facilities to forward application-specific event data from a user process to a consumer one. Monitoring services can be represented using XML (eXtendible Marking Language).

Higher level components can be built using the basic monitoring components. For example a local sensor manager that can execute user-specified sensors and forward event data, data archives and query servers which can extend monitoring and management infrastructure (to support tools and applications not using publisher-subscriber paradigm).

Applications of the monitoring infrastructure include fault management for MDS (Globus Metacomputing Directory Service) and monitoring and fault management (e.g. host accessibility, disk space availability, application process status) of parameter studies.

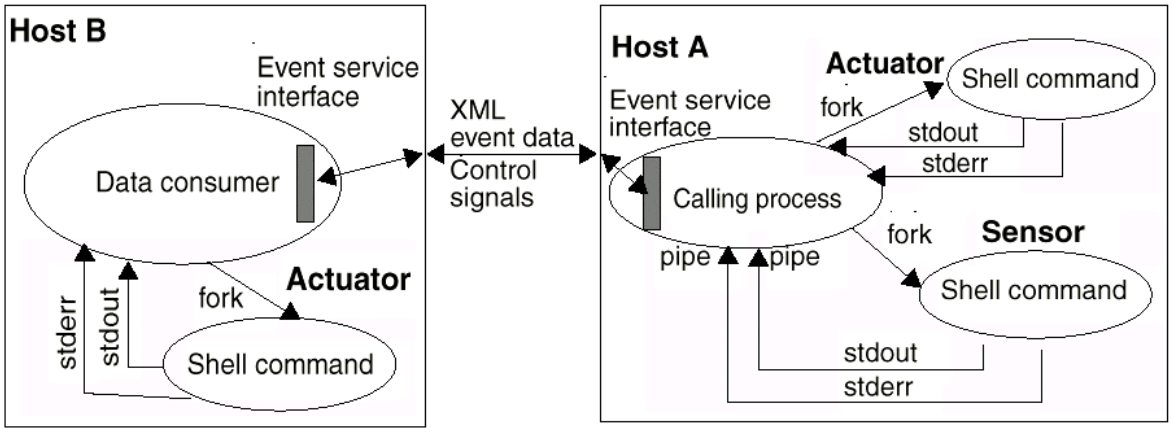

Figure 7. Monitoring Architecture of NASA-IPG

Infrastructure image is taken from [13].

5 Comparison of monitoring tools

Table 1 contains the comparison of the five monitoring tools. For the comparison we used the following requirements that a grid monitoring tool should satisfy:

- *Scalability*
  Scalability is one of the most important requirement in grid monitoring because the grid is naturally dynamically changing in several dimensions, i.e. in the
  - *number of users,*
  - *number of resources,*
  - *number of events.*

- *Intrusiveness*
  This feature defines the overhead of measurements performed by the tool. The tool can have effect on the
  - *monitored systems,*
  - *network.*

  Another question is whether

  - *sensors can control their intrusiveness.*

- *Validity of information*
  This feature could be defined by two parameters, i.e.
  - *freshness (time-to-live information),*
  - *accuracy (error bounds).*

- *Data format*
  Parameters of the data format are:
  - *easy-to-use*
  - *compactness*
  - *portability*
  - *standard*

- *Extendibility*
  For generality of grid monitoring the tool should be open ended and extendable with:
  - *new sensors,*
  - *new type of data.*

- *Communication*
  Applicability of the tools are restricted and defined by their
  - *communication model*

- *Security*
  Basic security requirements in grid should be supported by monitoring tools as well.
  - *authentication,*
  - *authorisation,*
  - *encrypted communication.*

- *Measurement*
  It is concerned with what is actually measured.
  - *metrics* (cpu, memory, disk, network, process).

- *Application performance*
  Not only resources but also application performance should be monitored in grids.
- *Visualisation tool support*
  Some monitoring tools give also tools to present monitoring information to the users.
- *Achievable time resolution*
  It refers to the time resolution of the measurements.
- *Archive handling*
  Archiving could be useful for several purposes (e.g. prediction, analysis).
- *Tool availability*
  Status of the development. In which form the tool is available for testing and general use.
  - promise of further developments

Table 1. Comparison of monitoring tools from the viewpoint of grid

	GRM	NetLogger	NWS	HBM	NASA-GMI	Autopilot
Scalability						
number of users	no (single user)	no (single user)	yes	yes	yes	no (single user)
number of resources	no (central collector)	no (central collector)	yes	yes but clients→ HLM not scalable	yes	yes
number of events	yes	no (processes send data themselves)	n/a (per seconds)	n/a (per seconds)	?	yes (data reduction)
Intrusiveness						
monitored systems	low: data collection separated from execution high: use semaphore	high	low (measure-ments per seconds) active/ passive methods	? (use unix 'ps')	?	?
network	low: generally high: at collection	high	low (cliques)	low	low (demand driven)	low (data reduction)
Can sensors control their intrusiveness?	no	no	no	no	no	no
Validity of information						
freshness (time to live information)	n/a	no	no	yes: heartbeat is valid until the next heartbeat	no	no
accuracy (error bounds)	no	no	yes (forecast)	n/a	no	?
Data format	Tape/PVM	ULM		own format	XML	SDDF
easy-to-use	yes (ASCII)	yes (ASCII)	yes (ASCII)		yes (with tools)	yes (with tools)
compactness	low	low	yes (single value series)		low	yes (binary version)

standard	no	IETF standard	no	no	yes	only for Pablo
portability	no	yes	no		yes	within Pablo toolset
Extendibility						
new sensors	possible	yes	yes	n/a	yes	?
new type of data	yes (event def.)	yes (ULM)	no (single value series)	n/a	yes	yes
Communication	socket + shared mem.	socket	socket	socket (UDP)	Globus	Globus
communi-cation model	push (buffered)	push (buffered)	query/ response (sensor→memory: push)	push	publisher/ subscriber	?
Security						
authentication	no	no	no	no	yes (via Globus)	yes (via Globus)
authorisation	no	no	no	no	no	no
encrypted communi-cation	no	no	no	no	no	no
Measurement						
metrics	only application	comprehensive	comprehensive	only process heartbeat	comprehensive	comprehensive
application performance	yes	yes	no	n/a	no	yes
Visualisation tool support	PROVE on-line	NLV on-line	no	no	no	Virtue on-line
Achievable time resolution	microsec	microsec	second	second	microsec	microsec
Archive handling	no	no	yes	no	no but possible	no but possible
Tool availability	no (further development)	yes (closed sw)	yes (further development)	yes	no (further development)	yes (further development)

Legend of symbols used in Table 1.
n/a non-applicable, **?** no information

References

[1] Grid Performance Working Group (Grid Forum) White Paper: A Grid Monitoring Service Architecture.

[2] Survey on Event Service
http://www.unix.mcs.anl.gov/~peng/survey.html

[3] N. Podhorszki, P. Kacsuk: Design and Implementation of a Distributed Monitor for Semi-on-line Monitoring of VisualMP Applications,
In: Proceedings. DAPSYS'2000 Distributed and Parallel Systems, From Instruction Parallelism to Cluster Computing, Balatonfüred, Hungary, pp. 23-32, 2000.

[4] P. Kacsuk: Performance Visualization in the GRADE Parallel Programming Environment, HPCN Asia, Beijing, China, 2000.

[5] B. Tierney et al.: The NetLogger Methodology for High Performance Distributed Systems Performance Analyser
Proc. of the IEEE HPDC-7 (July 28-31, 1998, Chicago, IL)
LBNL-42611

[6] D. Gunter et al.: NetLogger: A Toolkit for Distributed System Performance Analysis
Proc. of the IEEE Mascots 2000 Conference (Aug. 2000)
LBNL-46269

[7] The Globus Heartbeat Monitor Specification
http://www-fp.globus.org/hbm/heartbeat_spec.html

[8] R. Wolski et al.: The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing
Journal of Future Generation Systems, 1998.

[9] The Network Weather Service
http://www.cs.utk.edu/~rich/publications/

[10] J.S. Vetter and D.A. Reed: Real-time Monitoring Adaptive Control and Interactive Steering of Computational Grids,
The International Journal of High Performance Computing Applications (2000)

[11] Scalable Performance Tools (Pablo Toolkit)
http://vibes.cs.uiuc.edu/Project/Pablo/ScalPerfTools/Overview.htm

[12] E. Shaffer et al.: Virtue: Immersive Performance Visualisation of Parallel and Distributed Applications, IEEE Computer, December 1999, pp. 44-51.

[13] A. Waheed et al.: An Infrastructure for Monitoring and Management in Computational Grids,
Proc. of the 5th Workshop on Languages, Compilers and Run-time Systems for Scalable Computers (March 2000).