AstroGrid-D

Deliverable D6.1

Existing Monitoring & Steering Functionality in AstroGrid-D Applications[1]

Deliverable	D6.1
Authors	Thomas Radke (AEI)
Editors	Thomas Radke (AEI)
Date	7 August 2006
Document Version	1.0.0
Current Version	1.0.0
Previous Versions	

A: Status of this Document

Approved document for project deliverable D6.1.

B: Reference to project plan

This deliverable document refers to the task TA VI-I *"Analyse der vorhandenen Überwachungs- und Steuerungsmethoden"* and milestone M3 of work package WP-6 in the project plan.

[1]This work is part of the D-Grid initiative and is funded by the German Federal Ministry of Education and Research (BMBF).

C: Abstract

This document describes monitoring and steering methods as they already exist in AstroGrid-D applications at the time of the project start (end of 2005).

D: Changes History

Version	Date	Name	Brief summary
0.1.0	19 May 2006	Thomas Radke	Working Draft Creation
0.1.1	12 June 2006	Thomas Radke	Filled in use case descriptions
0.1.2	30 June 2006	Thomas Radke	Added comparison section; posted completed draft to WG-VI mailing list for final discussion
0.2.0	17 July 2006	Thomas Radke	Document approved by WG-VI
1.0.0	7 August 2006	Thomas Radke	Document finally approved

E:

Contents

1 Introduction **4**

2 Use Cases **5**

3 Comparison of Existing Monitoring & Steering Methods **10**

3.1 Interactive Access to Intermediate Files 10

3.2 Controlled Termination of Jobs (UC2, UC3) 10

3.3 Applications with Advanced Monitoring & Steering Methods 10

3.3.1 UC3 – NBODY6++ 11

3.3.2 UC5 - Cactus 11

4 Further Steps **13**

1 Introduction

The simulation codes used within our Astronomy Community often provide some application-specific methods already for monitoring and steering of parallel jobs at runtime and for real-time visualisation of intermediate results via live data streaming. Although such methods may be based on standard protocols (eg. HTTP, streamed HDF5 over sockets), up to now they usually are embedded in local environments only and thus not per se also grid-compatible.

In Work Package WP-VI we want to adopt existing and/or develop new application monitoring and steering methods for our astrophysical simulation codes which enable users to control their jobs interactively, no matter whether they are running in a local environment or somewhere on the Grid.

Grid Monitoring requires each job to register itself with a job metadata management system (WP-V, WP-II). A unique job identifier encodes the execution location of each job, along with a description and announcement of all the application-specific monitoring/steering/data streaming services it provides. Also necessary are mechanisms to dynamically open control connections into running Grid jobs, taking into account any given site-specific network administration issues (eg. client machines hidden behind fire walls, cluster compute nodes managed via a virtual private network).

Connections can be established both interactively by individual users or automatically by a remote control process; in any case the protocols for such peer-to-peer connections should be designed as generic as possible and implemented transparently by adopting the file-based access methods (as being developed in WP-III) and use standard user and Grid application programming interfaces (as provided by WP-VII).

As a first milestone, the monitoring and steering methods which already exist in present AstroGrid application codes (as of end of 2005) are summarised in this deliverable document. Along with a comparison analysis, they are evaluated according to their re-usability in a Grid environment.

2 Use Cases

The members of the AstroGrid architecture group have collected a couple of use cases representing typical astrophysical scenarios as they are currently performed, along with suggestions and ideas of how to use them in the future in a distributed Grid environment, i.e. the AstroGrid-D community Grid.

The use cases were developed in two phases. In the first phase (until November 2005) the users where asked to provide an overview of their applications, this was mainly to allow a freedom of specifying current and eventual future needs. Specifically for WP-VI, a questionnaire was sent around with specific questions in order to gather knowledge about existing monitoring/steering capabilities in AstroGrid-D applications. Unfortunately there was only very little feedback so that in a second phase (finished in Januar 2006) a more general questionnaire (including the questions from the first one) was distributed to ensured a more comprehensive view of each use case.

This section extracts from all the gathered use case descriptions the specific information that is relevant for application monitoring and steering as done in WP-VI. For a complete list of use case descriptions please refer to the AstroGrid intranet page `http://mintaka.aip.de:8080/lenya/intranet/live/project-documents/usecases.html`.

UC1 - NIRVANA

NIRVANA is a grid-based[2] computercode for the solution of the equations of magnetohydrodynamics and for the Poisson equation.
More information on the code can be found on `http://nirvana-code.aip.de`.

Each simulation writes startup messages to *stdout/stderr* but then outputs any further runtime information (names of files; parameters used throughout the run; current state such as time step, accuracy, etc; problems encountered during the run) into an ASCII logfile with a fixed location and name. Interactive monitoring of NIRVANA simulations is then done manually by periodically checking the constantly appended ASCII logfile via direct file access (ssh).

UC2 - AMIGA

AMIGA is a code for cosmological N-body simulations.
More information can be found on `http://www.aip.de/People/AKnebe/`.

`stdout/stderr` messages are written only at startup, further runtime output and progress information are then dumped entirely into an ASCII logfile (of known location and name). Interactive application monitoring is done by periodically checking the contents of this logfile.

AMIGA doesn't have integrated any high-level steering capabilities. It can be terminated in a controlled way though by periodically checking (at every timestep) for the existance of a file `terminateAMIGA` in the run's output directory.

[2] *grid-based* meaning the underlying implemented program algorithm, unrelated to Grid computing

UC3 - NBODY6++

NBODY6++ is a member of a family of high accuracy direct N-body integrators used for simulations of dense star clusters, galactic nuclei, and problems of planet formation.
More information can be found on `http://www.nbodylab.org`.

Diagnostic information is written to `stdout/stderr` which is used to monitor an NBODY6++ simulation.

An NBODY6++ job can be manually terminated by creating a certain file in the working directory (eg. "`touch STOP`"); the code periodically checks for the existance of such a file and will gracefully stop the run if found.

NOBODY6++, at least in its experimental version, also provides a graphical user interface for application-specific monitoring and steering: Xnbody (`http://www.fz-juelich.de/zam/xnbody/`). Users can log into a NBODY6++ job running on a remote machine, retrieve real-time data for local visualisation and also send back parameters used in the code. (see details in section 3.3.1).

The NBODY6++ code is being used and developed at a number of national and international institutes (which are not primarily a part of the AstroGrid-D community). Therefore the architecture of the application monitoring and steering services as well as the other services developed by AstroGrid-D, should acknowledge this broader application context and provide compatible solutions.

UC4 - Dynamo

Dynamo is an application for solving the induction equation with turbulent electromotive force (Alpha tensor) for modelling the turbulent dynamo in planets, stars and galaxies.

Startup information is written to `stdout/stderr`, further runtime information about the progress of the program goes to an ASCII logfile of known location and name (specified in the paramter file). Interactive monitoring is done by the job owner by periodically checking `stdout/stderr` and the ASCII logfile.

UC5 - Cactus

Cactus is used by the physicists in the Numerical Relativity department of the AEI to numerically simulate extremely massive bodies, such as neutron stars and black holes.
More information can be found on `http://www.cactuscode.org`.

Cactus simulations are typically submitted as parallel MPI jobs to local job management systems such as PBS or SGE.

Each job's `stdout/stderr` messages are stored in intermediate logfiles determined by the local queuing system (eg. in a local PBS spool directory located on a specific compute node, therefore not directly accessable from the headnode!); in addition, application-specific log- and ASCII datafiles may also need to be accessed during runtime for interactive monitoring. While the latter is achieved through existing monitoring functionality integrated in the simulation code (thorn HTTPD, see section 3.3.2), access to temporary `stdout/stderr` logfiles is done via a proprietary shell script (which determines the first compute node of the job, opens a remote shell on that node and fetches

the most recent contents of the corresponding logfile in the PBS spool dir).

Cactus also provides other, more advanced application monitoring and steering capabilities: thorn HTTPD implements a web-server, embedded into the program code, through which users can log into a running simulation, monitor its current state, view intermediate results, steer parameters, and interactively control the execution flow; for details on HTTPD see section 3.3.2. Another set of Cactus modules (thorns Formaline and Announce) can send metadata about an ongoing simulation to a Cactus user portal. This happens at startup (where static metadata such as start time, execution host, number of processors, parameter file and contents are announced), during the evolution at periodic intervals (update of dynamic metadata such as current iteration and timestep) and finally at program termination (when the simulaton unregisters itself from the portal).

UC6 - GADGET

GADGET is a freely available code for cosmological N-body/SPH simulations on massively parallel computers with distributed memory.
More information can be found on `http://www.mpa-garching.mpg.de/gadget`.

Beside runtime information on `stdout/stderr`, each GADGET job also produces a set of ASCII log files for evaluating code performance (`cpu.txt`), showing the number of performed timesteps (`info.txt`), the work load and particle load balance (`timings.txt`), or controlling the energy-balance (`energy.txt`). In a Grid environment it should be possible to interactively access `stdout/stderr` for monitoring purposes, and to visualise the status information in the other ASCII logfiles.

UC7 - Astrometric Matching

Astrometric Matching for classification of Spectral Energy Distributions (SED) is an essential research technique to discover new astronomical objects like obscured active galactic nuclei (AGNs), brown dwarfs, isolated neutron stars, or planetary nebulae (PNs).

For query optimizations, statistics about the data streams are collected (histograms, stream-frequency, stream-size) as well as statistics about network traffic and CPU load. These can be gathered for a monitoring tool to visualize the status of a network segment and the over-all network. Access to such statistics data is possible through a WebService interface (exchanging XML messages via SOAP).

UC8 - Clusterfinder

The purpose of clusterfinder is improving the source-identification of X-Ray galaxy clusters by correlating data of X-Ray photon maps (ROSAT All Sky Survey - RASS) and optical galaxy maps (Sloan Digital Sky Survey - SDSS).
More information can be found on `http://www.g-vo.org/portal/tile/products/services/clusterfinder/index.jsp`.

Clusterfinder writes runtime information to both `stdout/stderr` and into logfiles. These are periodically accessed (`tail -10 ...`) for monitoring purposes.

UC9 - Virtual Telescopes

In "Theory VO" (VO=Virtual Observatory !) parlance, a "Virtual Telescope" is a piece of software that mimics an astronomical observatory (telescope/instrument/satellite/spectrograph etc.) which can "observe" the result of a simulation and produce a synthetic observation that can be directly compared to the output of the instrument that is being simulated.

This use case writes runtime information to a logfile. The user usually downloads this file only after the job has finished. This indicates that no interactive monitoring of VT jobs is necessary.

UC10 - Millenium Query

The scenario Millenium Query is about selecting a subset of a large remote dataset and moving the result to the user.

The Millenium Query runs as a web application which produces logfiles containing certain state changes, user login, error messages, etc. The log4j package is used for Java logging. Logfiles can be accessed only from the machine where the job is running. Steering is not available.

UC11 - GEO600

The gravitational wave detector GEO600 near Hannover and other ground-based devices (eg. LIGO in the U.S., TAMA in Japan, VIRGO in Italy) aim at the direct detection of gravitational waves by means of a laser interferometer.
More information can be found on `http://www.geo600.uni-hannover.de/`.

GEO600 data analysis jobs are currently submitted as a series of independent single-CPU jobs to a Condor cluster. Information about the progress of the program is written to `stdout/stderr` which is available to the job owner through the Condor runtime system.

UC12 - The Planck Process Coordinator

The Planck Process Coordinator (ProC) is a workflow engine originally developed for running simulation and data analysis workflows (called "pipelines" in astronomical parlance) that occur within the Planck Surveyor project.
The use case enquiry didn't return any specific information about existing monitoring/steering functionality or requirements. So it was concluded that the ProC does not require any feature not already mentioned in the other use case descriptions.

UC13 - Integration of Robotic Telescopes

Two robotic telescope (RT) projects are part of AstroGrid: STELLA and PLANET. RTs can be seen as "normal" Grid resource similar to a compute resource. Instead of executing a compute job they perform observations which also produce data.

RT jobs are run remotely as Java programs, with program parameters encoded in XML strings

transported via SOAP messages. Monitoring information is written to ASCII logfiles (which are put into a local database) and to `syslog`. Error/warning messages as well as status information are available during execution via RMI (after registering as a listener). Program termination is notified via e-mail to the responsible PI.

The entire system is firewall-protected, ssh access is limited to authorised users.

3 Comparison of Existing Monitoring & Steering Methods

3.1 Interactive Access to Intermediate Files

Almost all use cases described in section 2 already use some simple form of runtime application monitoring by checking the `stdout/stderr` messages of a running job and/or specific logfiles continuously written by the application itself. This allows users to

- verify important metadata (eg. print most important runtime parameters to `stdout` at startup),
- monitor the progress of a running job (eg. continuously output the current timestep to `stdout` or some logfile),
- check for warnings and error messages from the code (usually printed to `stderr`),
- and even do some preliminary analysis of the data as they are written to logfiles and datafiles

In any case, this type of monitoring by interactive access to intermediate files is used so far as a local method only: the users actively logs into the machine/node (eg. via `ssh` remote shell) where her/his job is running, and accesses the intermediate files directly (eg. via `tail` or some other UNIX filter or script). No remote file access or staging is done at the moment.

The names and locations of intermediate files to be accessed are usually known beforehand, or they can be derived from startup parameters. When jobs are submitted as batch jobs to a local job management system (eg. PBS, SGE, or Condor on a cluster), this system is managing the redirection of `stdout/stderr` UNIX streams to some intermediate logfiles. The names and locations of such files are then dependent on the configuration of the local job management system and therefore machine-specific (for instance, `${PBS_SPOOLDIR}` which holds temporary `stdout/stderr` logs for all PBS jobs currently running can be located either on a cluster headnode, or on a filesystem on an internal compute node with no direct access from the headnode). Users typically use proprietary scripts to interact with the local job management system and get access to intermediate `stdout/stderr` logfiles.

3.2 Controlled Termination of Jobs (UC2, UC3)

AMIGA and NBODY6++ jobs can be triggered by a user to terminate gracefully by creating a special file (`terminateAMIGA` or `STOP` resp.) in the job's output directory. The simulation code is periodically checking after each timestep whether such a file exists.

3.3 Applications with Advanced Monitoring & Steering Methods

Two of the simulation uses cases, UC3 – NBODY6++ and UC5 – Cactus, already provide advanced functionality which can, at least partially, also be used for grid application monitoring/visualisation and steering.

3.3.1 UC3 – NBODY6++

Xnbody (`http://www.fz-juelich.de/zam/xnbody/`) is an online visualization tool designed for the NBODY6++ simulations. The basic idea is to run the simulator program on a supercomputer and to observe the simulation with Xnbody running on a local workstation. The exchange of data between the simulator program and Xnbody is performed by the Visualisation Interface Toolkit (VISIT, `http://www.fz-juelich.de/zam/docs/autoren2000/eickermann`) which has been developed in the Central Institute for Applied Mathematics at the Research Centre Jülich. VISIT provides an interface for establishing a client/server TCP connection through with data can be transfered. NetCDF is used as network data format. Client and server don't talk directly with each other but route their communication through a proxy server using a VISIT-internal service announcement protocol (SEAP). By this means it is also possible to connect to an NBODY6++ simulation running on an internal cluster node with no direct access from outside.

A VISIT plugin for UNICORE provides single sign-on and secure transportation facilities (based on X.509 certificates).

3.3.2 UC5 - Cactus

The Cactus simulation framework provides a generic code module (in Cactus terminology: a thorn) named `HTTPD`. This thorn implements a standard HTTP-1.0 webserver. It is written in C and uses standard UNIX socket communication calls on the network I/O layer. If this thorn is compiled into a Cactus configuration and activated at runtime, it serves as an ordinary HTTP webserver embedded in a running Cactus simulation: it listens on a chosen UNIX port for incoming HTTP requests from external clients and sends back appropriate HTTP replies. Thorn HTTPD provides both static (eg. executable name and version, parameter file, owner of the job, number of processors used, hostname of processor 0 where the webserver is running on) and dynamic information (eg. current iteration number and physical simulation time, estimated time to completion) about the current Cactus job on a dynamically generated simulation homepage.

Thorn HTTPD also defines a (Cactus-specific) API which can be used by other code modules to easily and generically extend the served webspace of a simulation by additional embedded monitoring capabilities, including simulation steering. Thorns can register their own HTTP reply callbacks to process incoming client requests for a certain URL (as a sub-URL of the main simulation homepage). The API also defines a set of utility routines to parse HTML query strings (for parameter extraction) and to compose user-defined HTTP replies (eg. to send back an ordinary HTML webpage, to authenticate a user, to set/remove HTTP cookies, etc.).

By convention the simulation homepage and all other webpages served by HTTPD and other thorns are publicly accessable – everybody can log into the simulation and monitor its current state, view visualisations of intermediate data, download output files etc. In addition to that, thorn HTTPD also provides remote steering capabilities: certain parameters (which are marked in Cactus as steerable at runtime) can be changed, the execution of a running simulation can be switched to single-step mode or set to continue until a given iteration number, physical simulation time, or some other user-defined breakpoint condition is met. A simulation can also be triggered to (gracefully) terminate immediately after the current evolution timestep.

Before any parameters can be set or the execution mode can be changed, a user has to authenticate her-/himself first on the Cactus Control Panel page. It uses the standard username/password

authentication method as implemented in the HTTP protocol (`Authorization` header field). All incoming HTTP requests for application steering webpages, such as the Control and Status Page or the Check/Modify Parameters Pages are then checked before further processing by matching the HTTP header field information against the corresponding entry in the user authorisation database.

4 Further Steps

Based on the information gathered in the use case enquiry about existing monitoring and steering functionality in AstroGrid applications, we will as a next milestone derive WP-VI-specific requirements and define an overall architecture which outlines the necessary steps towards the design and implementation of generalized grid-enabled monitoring and steering methods for AstroGrid.

These requirements and the architecture overview will be presented in the D6.2 deliverable document.