Automatic metadata capture and grid computing

RP Tyer, PA Couch, K Kleese van Dam, IT Todorov
CCLRC, Daresbury Laboratory, Warrington, Cheshire WA4 4AD

RP Bruin, TOH White, AM Walker, KF Austen, **MT Dove**
Department of Earth Sciences, University of Cambridge, Downing Street, Cambridge CB2 3EQ

MO Blanchard
Royal Institution, 21 Albemarle Street, London W1S 4BS

Abstract

We report a pragmatic approach to enable non-intrusive automatic metadata harvesting from grid-enabled simulation calculations. The framework, called RCommands, gives users a set of unix line commands and a web interface to the metadata database. The harvesting relies on the use of XML for output data file representation, and new developments of the `my_condor_submit` tool incorporating AgentX.

Introduction

This paper concerns a new set of tools developed by the *e*Minerals project [1] to facilitate automatic metadata harvesting from molecular-scale simulations. The key design requirements were that the tools should be non-intrusive for users, and pragmatic in design. This work represents a close collaboration between the developers and scientists.

The *e*Minerals project studies environmental processes at a molecular level, using a range of atomistic simulation methods. These are performed on the *e*Minerals minigrid [2], which integrates grid computing with grid data management methods based on the San Diego Storage Resource Broker (SRB). Job submission is via the `my_condor_submit` (MCS) tool [2,3], which also supports data and metadata management.

Grid computing enables large scale combinatorial studies. For example, studies of molecular pollutants on mineral surfaces requires comparing the energies of up to 210 members of a single family of molecules (the polychlorobiphenyls). It is necessary to perform calculations of the energies of each molecule in isolation and in contact with a mineral surface, together with repeat calculations using different levels of the theory within the simulation method. Other examples are where calculations are performed over a wide sweep of one or more input parameters, such as temperature or pressure. In all these cases, metadata is used to document the exact conditions of each simulation to enable scientists to locate simulation outputs easily. This replaces the role of the logbook or README file.

Metadata organisation model

The CCLRC model proposes three tiers within which metadadata are organised. The top level is the *study* level. This level is self explanatory. It is possible to associate named collaborators with this level, enabling collaborators to view and edit the metadata within a study. The next level down is the *dataset* level. This is the most abstract level, and users are free to interpret this level in a variety of ways. The third level is the *data object* level. This is the level that is associated with a specific URL (e.g. an SRB URL). The data object may include the files generated by a simulation run, and/or the outputs from subsequent data analysis. In combinatorial studies, there will be many data objects associated with a single dataset, and different types of calculations within a single study are organised with the dataset level. Examples of our usage of this hierarchy are given in Table 1. Our tools can attach metadata to each of the three levels.

Metadata to capture

Typically metadata associated with the study and dataset levels will be added by hand, with the automated metadata capture to be provided at the data object level (although we provide tools for metadata to be automatically captured at the other two levels as well). We define five types of metadata to capture:

Table 1. Examples of how the study / dataset / data object levels have been used to organise data.

Study	Molecular dynamics simulation of silica under pressure	*Ab initio* study of dioxin molecules on clay surface
Data set	Identified by the sample size and the interatomic potential model	Identified by number of chlorine atoms in the molecule and the specific surface site
Data object	Collection on the SRB containing all input and output files	Collection on the SRB containing all input and output files

Simulation metadata: information such as the user who performed the run, the date, the computer run on etc.

Parameter metadata: values for the parameters that control the simulation run, such as temperature, pressure, potential energy functions, cut-offs, number of time steps etc.

Property metadata: values of various properties calculations in the simulation that would be useful for subsequent metadata searches, such as final or average energy or volume.

Code metadata: information to enable users to determine what produced the simulation data, such as code name, version and compilation options.

Arbitrary metadata: strings to enable later indexing into the data, such as whether a calculation is a test or production run.

The RCommand framework

To facilitate automatic metadata capture, we have developed a set of scriptable unix line commands that can upload metadata to the metadata database (Table 2). The RCommands are a standard three-tier application:

Client: A set of binary tools written in C using the gSOAP library. The motivation for using C was the requirement that the tools be as self-contained as possible so they can easily be executed server side via Globus.

Application Server: Written in Java using Axis as the SOAP engine and JDBC for database access. The code essentially maps from procedural web service calls to SQL appropriate for the underlying database [4].

RDBMS: The backend database is served by a Oracle 10g RAC cluster. Although Oracle is used, there is no requirement for any Oracle specific functionality.

One of the main reasons to use a three tier model is that the backend databases are heavily firewalled and cannot be accessed directly from client machines.

The SOAP messages are sent to the application server via SSL-encrypted HTTP. The application server is authenticated using its certificate while the client requests are authenticated using usernames and passwords.

The use of web service technology allows the network related code to be autogenerated. On the server, the Axis SOAP engine is configured to expose specified methods of certain classes as RPC web services. The client code is generated on the fly by gSOAP from the WSDL file produced by Axis.

Metadata Manager: the web interface to the metadata database

The RCommands were written primarily to provide tools that can be used in scripts, but nevertheless they give scientists a useful interface to the metadata database. However, there are cases when a web interface is better, particularly when requiring a graphical overview that cannot be provided by a unix shell interface. Thus RPT has developed a web interface to the metadata database called the “Metadata Manager”. This gives an overview of the study level, from which the user can drill down into the various layers. The user can perform a number of the functions that are provided by the RCommands.

The MDM design uses the JSP Model 2 architecture, which is based on the Model-View-Controller (MVC) pattern. In addition, the Front Controller pattern is also used. The majority of the code in the Model layer is common to both the RCommands and the MDM. Hence, as with the RCommands, the database connectivity is provided using the JDBC libraries.

Collecting metadata: the role of XML in output files

Much of the metadata we collect is harvested from output data files. To facilitate this, we have enabled our key simulation programs to write the main output files in XML, using the Chemical Markup Language [5]. CML specifies a number of XML element types for representing lists of data, including:

`metadataList`: contains general properties of the simulation, such as code version;

`parameterList`: contains parameters for with the simulation;

`propertyList`: contains property values computed by the simulation.

Table 2. The ten RCommand unix line commands.

RCommand	Action
`Rinit`	Starts an RCommand session by setting up session files
`Rpasswd`	Changes the password for access to the metadata database
`Rcreate`	Creates study, dataset and data object levels, associating the lower levels with the level immediate above, adding a name to each level, adding a metadata description and topic association in the case of creating a study, and associating a URI in the case of creating a data object.
`Rannotate`	Adds metadata. In the case of studies or datasets, this enables a metadata description, and in the case of datasets and data objects it also enables metadata name/value pairs. It also enables more topics to be associated with a study.
`Rls`	Lists entities within the metadata database. With no parameters, it lists all studies, and with parameters it will list the entries within a study or dataset level. It can also be used to list all possible collaborators or science topics.
`Rget`	Gives the metadata associated with a given study, dataset or data object. In the case of a study, it can also list associated collaborators and science topics.
`Rrm`	Removes entities or parameters from the metadata database.
`Rchmod`	Add or remove co-investigators from a study.
`Rsearch`	Search the metadata database, tuned to search within different levels and against descriptions, name/value pairs and parameters.
`Rexit`	Ends an RCommand session, cleaning up session files.

It is usual to have more than one of each list, particularly the `propertyList`. These lists correspond to the metadata types described above. An example is given in Figure 1.

Automatic metadata capture within a grid computing environment

As described in the introduction, the *e*Minerals scientists run their simulations using the MCS tool [3]. MCS deals with four types of metadata:

1. An arbitrary text string specified by the user.
2. Environment metadata automatically captured from the submission and execution environment.
3. Metadata extracted from the first `metadataList` and `parameterList` elements described in the previous section.
4. Additional metadata extracted from the XML documents. These specifications take the form of expressions with a syntax similar to XPath expressions. These are parsed by MCS and broken down into a single term used to provide the context of the metadata (such as ‘`FinalEnergy`’) and a series of calls to be made to the AgentX library [6].

MCS uses calls to the AgentX library to query documents for data with a specific context. For example, AgentX could be used to find the total energy of a system calculated during a simulation. The user specified expression might have the form:

```
AgentX = FinalEnergy, output.xml:/
PropertyList[title='rolling
averages']/Property
[dictRef='dl_poly:eng_tot']
```

The term providing the name of the metadata item is '`FinalEnergy`' and the document to be queried is `output.xml`. The string following ‘`output.xml:`’ is parsed by MCS and converted to a series of AgentX library calls. In this example, AgentX is asked to locate all the data sets in `output.xml` that relate to the concept ‘`property`’ and which have the reference ‘`dl_poly:eng_tot`’. The value of this property is extracted and associated with the term `FinalEnergy`. The RCommands are then used to store this name value pair in the metadata database.

AgentX works with a specification of ways to locate data in documents (such as a CML document) that have a well defined content model. There are two components to the AgentX framework:

1. An ontology that specifies terms relating to concepts of interest in the context of this work. These terms relate to classes of real world entities of interest and to their properties. The ontology is specified using OWL and serialised using RDF/XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<cml xmlns="http://www.xml-cml.org/schema"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<metadataList>
 <metadata name="identifier" content="DL_POLY version 3.06 / March 2006"/>
</metadataList>

<parameterList title="control parameters">
 <parameter title="simulation temperature" name="simulation temperature"
 dictRef="dl_poly:temperature">
 <scalar dataType="xsd:double" units="dl_polyUnits:K"> 50.0 </scalar>
 </parameter>
</parameterList>

<propertyList title="rolling averages">
 <property title="total energy" dictRef="dl_poly:eng_tot">
 <scalar dataType="xsd:double" units="dl_polyUnits:eV_mol.-1"> -2.7360E+04
   </scalar>
 </property>
</propertyList>

</cml>
```

Figure 1. Extracts of a CML output file, showing examples of the `metadataList`, `parameterList` and `propertyList` containers.

2. The mappings, which are used to relate terms in the ontology to document fragment identifiers. For XML documents, these fragment identifiers are XPointer expressions that may be evaluated to locate data sets and data elements in the documents. Each format is associated with its own set of mappings and serialised using RDF/XML.

AgentX is able to retrieve information from arbitrary XML documents, as long as mappings are provided. Mappings exist for an increasing number of simulation codes.

Post-processing using the RParse tool

Although XML output will capture all information associated with the simulation, it is inevitable that the automatic tools may miss some of the metadata; it is not always obvious at the start of a piece of work what properties are of most interest. We have used the AgentX libraries to develop the *Rparse* tool to retrospectively extract metadata by scanning over the XML files contained within the data objects in a single dataset. Rparse uses the SRB Scommands, the RCommands, and the AgentX library. The user specifies a collection in the SRB, the relevant output files, AgentX query expressions, and the dataset into which the metadata is to be inserted.

Examples of applications

We have used the metadata tools for the following applications:

- collaborative studies of adsorption of molecules onto mineral surfaces;
- parameterisation of computations of PCB molecules [7];
- study of silica glass under pressure [8]
- sharing literature search results, with the data object linking to the on-line publication URL.

We are grateful for funding from NERC (grant reference numbers NER/T/S/2001/00855, NE/C515698/1 and NE/C515704/1).

References

1. MT Dove *et al*. The *e*Minerals project: developing the concept of the virtual organisation to support collaborative work on molecular-scale environmental simulations. *Proceedings of All Hands 2005*, pp 1058–1065, 2005
2. M Calleja *et al*. Collaborative grid infrastructure for molecular simulations: The *e*Minerals minigrid as a prototype integrated compute and data grid. *Mol. Simul*. **31**, 303–313 (2005)
3. RP Bruin *et al*. Job submission to grid computing environments. *Proceedings of All Hands 2006*
4. M Doherty, K Kleese, S Sufi. "Database Cluster for e-Science". *Proceedings of UK e-Science All Hands Meeting 2003*, pp 268–271, 2003
5. TOH White *et al*. Development and Use of CML in the *e*Minerals project. *Proceedings of All Hands 2006*
6. PA Couch *et al*. Towards Data Integration for Computational Chemistry. *Proceedings of All Hands 2005*, pp 426–432 (2005)
7. KF Austen *et al*., Using escience to calibrate our tools: parameterisation of quantum mechanical calculations with grid technologies. *Proceedings of All Hands 2006*
8. MT Dove *et al*. Anatomy of a grid-enabled molecular simulation study: the compressibility of amorphous silica. *Proceedings of All Hands 2006*