



Computational Science Technical Note **CSTN-008**

Scientific Data Management in a Grid Environment

H. A. James and K. A. Hawick

2005

Managing scientific data is by no means a trivial task even in a single site environment with a small number of researchers involved. We discuss some issues concerned with posing well-specified experiments in terms of parameters or instrument settings and the metadata framework that arises from doing so. We are particularly interested in parallel computer simulation experiments, where very large quantities of warehouse-able data are involved. We consider databases and other framework technologies for manipulating experimental data and controlling the outputs from parallel runs arising from large cross products of parameter combinations. More complex issues arise when experimental data must be managed in a distributed grid context and multi-site users are sharing or exchanging simulation outputs. Considerable extra value can be obtained from simulation output that can subsequently be data-mined.

Keywords: distributed computing; cluster computing; metadata; data mining; data management

BiBTeX reference:

```
@ARTICLE{CSTN-008,  
  author = {H. A. James and K. A. Hawick},  
  title = {Scientific Data Management in a Grid Environment},  
  journal = {Journal of Grid Computing},  
  year = {2005},  
  volume = {3},  
  pages = {39-51},  
  number = {1-2},  
  month = {September},  
  note = {ISSN: 1570-7873 (Paper) 1572-9814 (Online)},  
  doi = {10.1007/s10723-005-5464-y},  
  keywords = {distributed computing; cluster computing; metadata; data mining; data  
             management},  
  series = {CSTN-008}  
}
```

This is an early preprint of a Technical Note that may have been published elsewhere. Please cite using the information provided. Comments or queries to:

Prof Ken Hawick, Computer Science, Massey University, Albany, North Shore 102-904, Auckland, New Zealand.
Complete List available at: <http://www.massey.ac.nz/~kahawick/cstn>

Distributed Scientific Simulation Data Management

K.A. Hawick and H.A. James
 Institute of Information and Mathematical Sciences
 Massey University – Albany
 Private Bag 102-904 NSMC, Auckland, New Zealand
 Email: {k.a.hawick & h.a.james}@massey.ac.nz
 Tel: +64 9 414 0800 Fax: +64 9 441 8181

8 July 2004

Abstract

Managing scientific data is by no means a trivial task even in a single site environment with a small number of researchers involved. We discuss some issues concerned with posing well-specified experiments in terms of parameters or instrument settings and the metadata framework that arises from doing so. We are particularly interested in parallel computer simulation experiments, where very large quantities of warehouse-able data are involved. We consider databases and other framework technologies for manipulating experimental data and controlling the outputs from parallel runs arising from large cross products of parameter combinations. More complex issues arise when experimental data must be managed in a distributed grid context and multi-site users are sharing or exchanging simulation outputs. Considerable extra value can be obtained from simulation output that can subsequently be data-mined.

Keywords: distributed computing; cluster computing; metadata; data mining; data management.

1 Introduction

A common *modus operandi* for computational scientists running numerical simulations is shown in figure 1. A numerical model for the phenomena under study is constructed. The model is initialised and is spun-up into a realistic or at least representative state, whereupon measurements can be taken. Depending upon the model involved, measurements are made from

static configurations, which may be stored separately, or measurements are made as part of the evolutionary process of taking the model configuration from one state to another. These configurations can usefully be warehoused for later mining.

Some important examples include numerical models for weather and climate study [2], where a set of model variables such as atmospheric temperature, pressure and wind velocity are time-evolved from one configuration to the next, to predict how real weather systems will develop. Climate study is similar except that the time scales simulated are much longer and the model granularity generally coarser. Other models in computational physics and engineering studies fall into this *general pattern of operation*. Some examples we consider in this paper (section 3) are Monte Carlo lattice models [4]; stochastic network models [3]; and artificial life growth models [5].

In running models that have even a few separate parameters it is necessary to manage the range and combinations of parameters. Sometimes the (computational) cost of running a model is small and it is feasible to throw away the configuration outputs and just preserve the few measurements that are made “during the run”. It is sometimes however either too expensive to be able to justify re-running models with the same parameters or in some cases it is important for legal or other operational reasons to keep all model output in an archive. Ideally the working computational scientist would like to afford the storage capacity to preserve the output from all past runs for possible future further analysis or for bootstrapping new model runs. There are tradeoff costs for this storage that must be

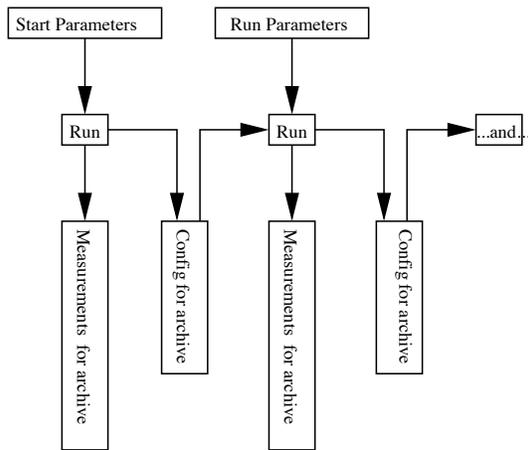


Figure 1: Flow of archive-able data from simulation runs in a common pattern for numerical experiments.

weighed against the computational cost of regenerating data from runs. However, more than the physical cost of storage media there are “total cost of archiving” issues that need to be considered more deeply.

We explore one important complexity contribution to the total cost of archiving. Managing data that is simulated from codes that are continually evolving in a way which is forward-compatible is non trivial. The output formats are likely to differ slightly as the codes evolve. One step towards this is to consider the cross-products of all possible parameters values that could be used and to explore the implications of labelling experimental run outputs by these parameter values. Another step is to use a robust textual tagging of output values that offers some defence against changing output formats through the use of generalised data extraction utilities and scripts.

Consider a simulation with just two parameters, as shown in figure 2. We can imagine the computational scientist steering this simulation does not systematically explore all of the possible parameter space, but rather poses some preliminary experiments that explore some combinations of parameter values that span some area of parameter space he expects have some “interesting” properties. Parameter one is represented by rows, and parameter two by columns. The scientist has carried out runs over the parts of parameter space shown and subsequently wishes to keep track of the “runs” obtained. Run data may be reused later and to have “archive value” it must be easily retrievable.

It has been our own experience that scientists often use some *ad hoc* approach to keeping track of model output data often involving file or directory names, perhaps with some README files to describe what exper-

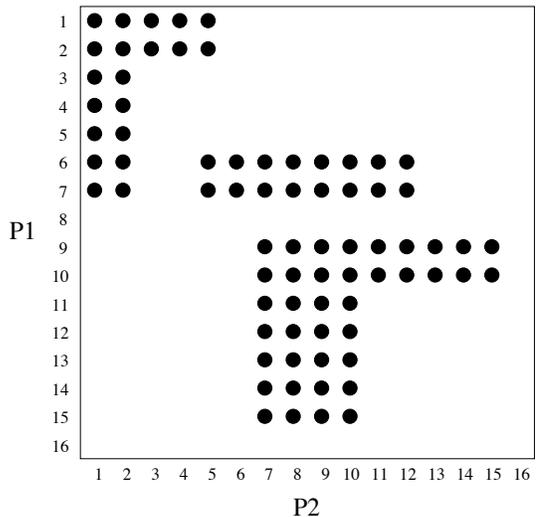


Figure 2: Sparse cross-product of a two-parameter system. A simple two-parameter system where various experiments have been carried out yielding non-null entries in the data base or matrix of possible values. The extreme values of the parameters and the discretisation scheme have set the bounds and size of the matrix table. The outer product need not and likely will not be fully populated.

iments have been done. This is analogous to the online “lab notebook” easy to use for very small sets of runs but rapidly becomes hard to manage for even just a three parameter experiment. A common situation for our running stochastic simulations is that we rapidly accumulate new parameters as we develop the model. Two main ones accrue from the model itself, a further one is the sample number if we are averaging over many different stochastic simulation sequences, and a fourth is the random number generator seed if we wish to keep track of separate repeatable stochastic sequences. It is common to rely on metadata [6] in the form of long file names or sometimes in terms of headers stored in the configuration files themselves. These can be opaque to the browsing scientist planning a follow-on experiment however.

In this paper we explore how relational database technology can be augmented with some simple conventions and easily produced tools to help manage more complex batches of runs and archives of data.

2 A Virtual Spreadsheet

Consider the parameters available in a multi-dimensional virtual spreadsheet. We have a hyper-space of parameter values, that is sparse as not all com-

binations of parameters are necessarily deemed worth running, and of course the actual values used over a range will be limited. We want the scientist to be able to pose the questions: **1)** What have I run already?; and **2)** What can I now run to pose a new research question, making best use of my existing data sets? We want the “housekeeping” operations for keeping track of the data to be as automated as possible, while still being compatible with the simulation programs already in use. We also want new programs to be able to easily access previous data.

One classic approach to these problems is to write simulation programs that “log” all sorts of extra information and measurements as well as the parameters actually used. The log files can then be scanned for relevant information and often new questions can be posed using old data that was generated before the question had been thought of. We have often adopted conventions for logging data to support this. For instance a textual tag is invented to describe the numerical measurement in question and the output value is prefixed with this tag in the log file. Standard `Unix` tools such as `grep`, `cut` and `paste` and other text manipulation programs or languages such as `perl` or `python` can readily be combined into scripts to extract relevant values from an archive of log files, in suitable form for plotting for example. Another common technique is to encode some parameter values in the filenames or in the names of the directories. This can aid casual browsing and experiment planning up to a point, but rapidly becomes cumbersome when many values are involved. How can the scientist view and visualise the sparse hyper-space of parameters, and hence assess existing data availability and plan experiments?

Imagine a virtual spreadsheet that supports looking at any two axes from the hyper-dimensional data set. We would like the virtual spreadsheet tool to allow specification of these axes and to cope with what is a sparse set of data that may not be stored online or locally. The posed queries or high level commands can in principle be organised by the virtual spreadsheet tool into the necessary data retrieval requests to be scheduled and the resulting “plot” assembled. Ideally the tool would have enough information to at least estimate how long satisfying the request will take, if it is not in “interactive time”.

3 Application Examples

A recent real experiment, of interest to us, involved the simulation of the Ising model under Small-World conditions, as described in [4]. The model uses Markov-

Chain Monte Carlo (MCMC) [7] sampling of a data space. There are four major parameters that we wish to track, as well as at least two extra parameters that can be significant. These parameters are shown in table 1. Some of the parameters combine to give other properties, such as: the total number of points in the system is calculated by the number of lattice points per dimension raised to the power of the dimensionality. This number can then be multiplied by the probability of small world effects on the lattice to give the number of lattice points that need to be modified as per the effects. Recording the random number seed ensures that any experiment we perform is reproducible, an important consideration in any scientific investigation.

Our simulation was originally developed by choosing an initial T and p , and then refining the values as the simulation produced results. When we had identified a promising (or “interesting”) area of the $T \times p \times L$ parameter space we performed a production run using the local supercomputer cluster. Each simulation took approximately 28 hours to complete 11 million update steps. By the end of the study we had produced approximately **0.5TB** of data. The problem was organising it and searching through it in an efficient manner. Our first approach was to write a series of `Unix` shell-scripts. We used long-named data files, including most of the relevant parameters as either part of the file name or at least as a comment-style metadata within the file. The scripts were a good *ad-hoc* solution - they enabled us to do basic searching and sorting on the data, and allowed us to execute our custom-written analysis programs on the data files. However they were not good at helping us identify “holes” in the data. It is particularly difficult to develop statistical analysis scripts that can cope with gaps in the data space.

Parallel supercomputers and clusters are often shared resources. Our is no exception, being shared between computer scientists, computational chemists and computational biologists; it is not under our control. We observed that sometimes due to queue failures or lack of scratch space our simulations would either not start, crash upon startup, crash part-way through the simulation, or simply freeze - and not make any progress. The latter condition was later identified as a transient hard-drive problem in a small number of the compute nodes. To compound matters, we were using the resulting configurations from previous runs as a starting point for subsequent runs in the same parameter space. When a “gap” was identified this meant that before we could start the next iteration of that configuration the gap would have to be filled. Because of the number of jobs that were being created as the $T \times p \times L$ product, it was very difficult to spot a single

Parameter Description	Primary?	Min	Max	Min Stride
dimensionality of study (d)	yes	1	5	1
number of lattice points per dimension (L)	yes	1	1024	1
temperature of system (T)	yes	4.000	5.000	0.001
probability of small world effects on lattice (p)	yes	0	1	0.00000001
number of update steps in data file ($steps$)	no	1	999999999	1
random number generator seed (s)	no	0	999999999	1
update method (u)	no	0	1	1

Table 1: An enumeration of the relevant parameters in our Small-World Ising model simulations. The abbreviation used to refer to each parameter is given in parenthesis after the parameter’s description. Some parameters are marked as primary, signifying they are crucial quantities to keep track of; in practice we use these parameters to define the data table’s primary key. For other parameters, such as *seed*, min, max and stride has no real meaning – as the random number generator will use a random seed for each run.

job that had failed. A series of scripts written to help with this task proved to be large and quite unwieldy. An integrated approach allows us to harvest data from the distributed nodes directly into the database - or even to keep track of data that remains distributed amongst nodes’ local disks. We believe these sorts of operational problems are quite common amongst computational scientists.

Our data analysis programs also require exploration of different parts of the parameter space we measured. For example, some require the first series of data collected for each of the different values of the parameters’ cross-product, while others require all the series for a given value of the parameters’ cross-product in a single sequence. It has been quite difficult maintaining the scripts necessary to extract all the required data in a portable manner and to cope with the exception handling routines for dealing with missing data.

We also run our own programs which simulate *Ad-hoc* network structures [3] and Artificial Life (ALife) [5] predator-prey models for studying species evolution. The *Ad-hoc* network simulation involved the variation of four distinct parameters: the number of radio transmitter sites in the simulation, the radius of perception of each transmitter, the type and degree to which the network was perturbed by Small-World effects, and the random number seed for configuration. A parameterised study was performed using a wide-range of parameter values; not all possible parameter values between the minimum and maximum were used. Searching the database of values means that it is not necessary to compute the complete parameter cross-product. We had the aim in the experiment of *steering* the experiment while it was in progress, through the modification of parameters, to investigate interesting phenomena.

The ALife simulation uses six independent parameters to represent predator and prey birth rates, their

longevity, evolutionary periods and a random number seed for configuration information. This model was particularly interesting as we had parallelised it. The simulation was implemented as a parallel program using an optimal number of **42 processors**. Configurations had to be consistently stored to obtain meaningful statistics on the experimental runs. When analysing the results of this experiment we had to ensure that individual processors’ output logs were safely archived to prevent data being lost, particularly should subsequent manual intervention be required for exception handling.

In summary, these simulations require a mix of integer and floating-point parameters, and have potentially large parameter spaces to explore. Gaps in the run-sequences are difficult to cope with for statistical analysis purposes without a good management framework.

4 Framework Architecture

Our prototype scientific data management framework is based around the use of a database (MySQL [8]) and a Java driver program (using JDBC). We have defined the table structure in a way that we hope will lend itself to being able to represent many different types of simulation systems. Each simulation system will have its own experimental parameters. The parameters that have been defined for our Ising experiment are shown in table 1. The database not only records the *meta-data* about the parameters such as their description and minimum and maximum values, but also the values of the parameters that have actually been used in real experiments.

Our control scripts have been modified to test for successful experiment completion. In the case of our

Using model a simple example of this is to ensure the number of lines on the output log file is 10 million, and that there exists a final configuration file. After the experiment is deemed to be successful the file is moved to a known (standard) directory and the relevant data on the experiment run is inserted into the database. A typical output log from our Ising experiment is 50MB, which needs to be subsequently analysed. Instead of copying the actual data into the database (perhaps as a Binary Large Object - BLOB) in the prototype we insert the absolute path name to the file in the database.

We recognised very early that our scientific data archive will likely consist of experiments with a **sparse** cross-product of parameters as shown in figure 2. Using a series of `SELECT` statements allows us to determine whether all the required data is in the database. It also allows us to select only sequences of data that we are interested in for our data analysis programs. While the ability for a tool to compute the necessary cross-product of a parameter set and initiate experiments is not new (e.g. Nimrod [1]), as we conceive of new data ranges and analysis techniques that we would like to investigate, we are able to play "what if" games and interrogate the database as to what data is already in the database, and what will need to be generated. In the case that more data may need to be generated, the database queries are able to output the precise values that are required to generate execution scripts and then schedule the jobs.

In future versions of the framework we hope to add a graphical user interface that will easily allow novice users to interrogate the system, perhaps using some of the ideas mentioned in section 2. At present, our prototype uses a programmer interface and a library of interface routines that can be linked with our simulation programs written in C++ or Java.

We recognise that many application scientists are uncomfortable with having to remember complicated database access routines. To alleviate the need for users to remember the routines we introduce a new Library layer to shield the user. This library layer is shown in figure 3. The library actually serves two purposes. The first is to shield the users from the database. The second is actually to protect the database from the users.

A significant problem could arise if the files that store the raw experimental data are moved, deleted, or renamed. We have considered a number of options, including creating a file system space that is only writable by programs using the database as an access control mechanism, but as yet this issue is unresolved. We can also write our own "safe" versions of the `cp` and `mv` programs that update the database as any data files

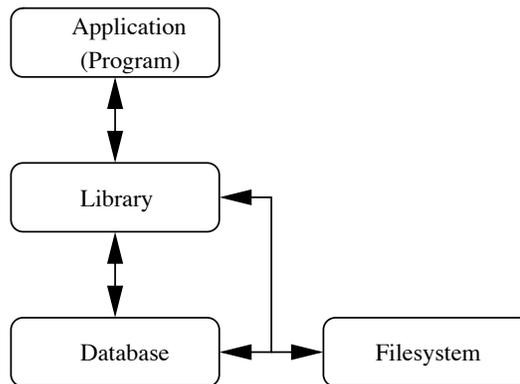


Figure 3: Layered software stack diagram showing indirect access to the filesystem via the database.

are moved. These new programs would work in conjunction with the library layer to ensure the database's consistency in the face of different file-system operations.

In an idea similar to that reported in [9] we are attempting to "fool the application program" into believing that separate data files actually exist where they are actually pseudo-files accessed by the underlying database and manipulated by our database access routines. The application and user writes to "files" in the normal way, but the calls are intercepted by library functions that access the database rather than the normal file-system.

5 Database Implementation

In our current prototype we have identified a number of issues that we have not yet been able to adequately resolve. These include: the difficulty in assigning data ranges; ensuring links to data files remain valid; and computing the cross-product of a variable number of parameters. These points are discussed below.

A naive implementation of parameter spaces, as used above, is useful when a simple range of parameter values is required. It works simply because most parameters to our modelling experiments are continuous variables that we can give a reasonable delta (or smallest change amount). A problem arises when we wish to represent a data range which is not sampled at even intervals. An example of this is when the scientist wishes to have their values evenly spaced on a *log-log* graph: the anti-log values are not evenly spaced! The only real solution to this is either to specify a formula for the calculation of values in the required range, or alternatively to enumerate the entire set of valid values.

At present we represent all numerical parameters as a fixed-length (Java) string and perform increment/decrement operations using the fixed length strings. This is done for two reasons: firstly when our data analysis programs iterate over the raw data files they have a consistent representation of the data value, and secondly we wish to specify a value's precision unambiguously. We don't want to have any rounding effects (due to the floating point representation) creeping into the system. We will not end up with a situation that 2.00000000001 is stored instead of 2.0 due to rounding errors and machine epsilons.

We often wish to inspect the database to find out what data is present and what is missing. This involves a cross-product of the parameters used to record the data. A large problem is then efficiently and algorithmically computing cross-products of variable numbers of parameters. The number of elements in a cross-product increases exponentially with each new parameter added to the vector. The traditional method for enumerating the cross-products is via nested loops. In the situation that the number of parameters is not known in advance, the enumeration is difficult to achieve.

6 Summary and Conclusions

We have identified a common operational pattern for scientific experiments - and which is particularly common for numerical simulation experiments. We have described some of the problems facing a computational scientist managing "runs", their measurements, and the resulting configuration files. We have described how *ad-hoc* solutions can be augmented using commonly available public domain software tools, and how a "Computational Laboratory Information System" can be based around a database. Our prototype and the ideas arising from it can be usefully applied to situations where large amounts of data are generated and must be curated. We believe the tools we describe here can be used to construct a system that is capable of coping with quite large repositories, but which is also open enough that distributed components can be readily added to cope with collaborative grid environments.

We have described the sparse data structure that arises from partial cross-products of parameters into a simulation, when the scientist does not want to explore the full parameter space. We have shown that this need not be an obstacle to a simulation management system, and that gaps in the data can be handled. We are presently extending our prototype to include some sim-

ple data-mining utilities that will be compatible with the data management system.

References

- [1] Abramson D., Sasic R., Giddy J. and Hall B., "Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations", The 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.
- [2] Hawick, K.A., Coddington, P.D. and James, H.A., "Distributed Frameworks and Parallel Algorithms for Processing Large-Scale Geographic Data" in *Parallel Computing* 10, 1297 (2003).
- [3] Hawick, K.A. and James, H.A., "Small-World Effects in Wireless Sensor Networks", submitted to Hawaii Int. Conf. on System Sciences, 2005.
- [4] Hawick, K.A. and James, H.A., "Ising Model Scaling Behaviour on Small-World Networks", Computational Science Technical Note CSTN-006, March 2004.
- [5] James, H.A., Scogings, C.J., and Hawick, K.A., "A Framework and Simulation Engine for Studying Artificial Life", in Res. Lett. in the Information and Mathematical Sciences, Vol 6, May 2004, ISSN 1175-2777.
- [6] Metadata at W3C. Available from <http://www.w3.org/Metadata/> Last visited July 2004.
- [7] Metropolis, N., Rosenbluth A.W., Rosenbluth M.N., Teller, A.H. and Teller E., "Equation of state calculations by fast computing machines", J. Chem. Phys., 21(6), pp 1087-1092, June 1953.
- [8] MySQL Database, Available from <http://www.mysql.com> Last visited July 2004.
- [9] Patten, C.J., Vaughan, F.A., Hawick, K.A. and Brown, A.L., "DWorFS: File System Support for Legacy Applications in DISCWorld", in Proc. of the 5th IDEA Workshop, Fremantle, February 1998.