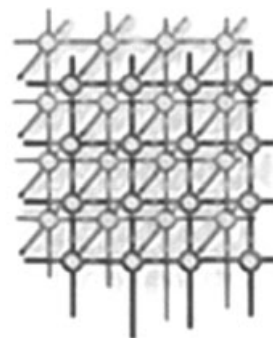


Fault-tolerant execution of large parameter sweep applications across multiple VOs with storage constraints



Shahaan Ayyub^{*,†}, David Abramson, Colin Enticott, Slavisa Garic and Jefferson Tan

Faculty of Information Technology, Monash University, Melbourne, Vic. 3145, Australia

SUMMARY

Applications that span multiple virtual organizations (VOs) are of great interest to the e-science community. However, our recent attempts to execute large-scale parameter sweep applications (PSAs) for real-world climate studies with the Nimrod/G tool have exposed problems in the areas of fault tolerance, data storage and trust management. In response, we have implemented a task-splitting approach that facilitates breaking up large PSAs into a sequence of dependent subtasks, improving fault tolerance; provides a garbage collection technique that deletes unnecessary data; and employs a trust delegation technique that facilitates flexible third party data transfers across different VOs. Copyright © 2008 John Wiley & Sons, Ltd.

Received 11 December 2007; Revised 30 March 2008; Accepted 7 May 2008

KEY WORDS: e-science; parameter sweep applications; Grid

1. INTRODUCTION

The computational Grid aggregates computational power and storage capacity by coupling together distributed CPU, network and storage resources [1]. The scale and nature of Grid testbeds make it possible to solve particular challenging problems in science and engineering using *parameter sweep*

^{*}Correspondence to: Shahaan Ayyub, Faculty of Information Technology, Monash University, Melbourne, Vic. 3145, Australia.

[†]E-mail: ayyub.shahaan@infotech.monash.edu.au

Contract/grant sponsor: Australian Research Council

Contract/grant sponsor: CSIRO Division of Atmospheric Research



applications (PSAs). PSAs are generally specified as a set of experiments, each representing a *parameterized instance* of a user-defined model. In principle, each instance is executed with a distinct set of parameter combinations on a number of resources. The parameters control the behaviour of the model and allow users to explore different design scenarios or options. The case study in this paper, for example, varies environmental conditions in a simulation of burning grasslands over a period of 21 years in a particular geographical area. PSA models have been applied in different areas including physical sciences [2,3], mathematics [4] and biology [5].

Conceptually, implementing a PSA is not difficult. For example, a simple Cartesian product of a range of parameters would generate a set of experiments, which can then be executed in parallel on a number of resources. However, in practice, effective deployment of PSAs on the Grid poses intricate management issues, making the handling of the experiments error-prone and unwieldy to perform manually. As a result, parametric modelling tools such as Nimrod [1], APST [6] and GridBus [7] were developed to simplify this process. These tools allow e-scientists to concentrate on their science while hiding the complexities of deploying and managing applications on widely distributed resources.

We recently attempted to execute extremely large PSAs on the Grid, and this challenged solutions to task, data and trust management.

Task management: If an individual computation takes a long time, then executing it as one continuous run becomes error-prone and unwieldy. This is particularly true in the presence of unreliable network and resource conditions found on dynamic platforms such as the Grid. When a runtime error occurs, the computation may abort its execution and must be restarted from the very beginning. All intermediately generated data will be lost. To address this problem, applications can be split and *check-pointed* [8]. In the event of a failure, a program can resume from the most recently saved check-point, rather than restarting the entire application. Although this idea is not new, we have implemented a simple user-driven technique supporting Grid-based check-pointing in order to achieve fault tolerance and improved application performance. The check-pointing is explicit; the user can set the granularity of check-points when specifying their application.

Data management: Task-splitting and check-pointing exacerbate data management problems because it becomes necessary to store the application state for each check-point. Moreover, it might also be necessary to transfer application's state to another resource should the task be migrated there. It is desirable to simplify the specification of data-handling mechanisms as we have done, through basic primitives built into the specification syntax and semantics that we use in describing applications. On the other hand, data transfer costs must be considered carefully. Some scheduling heuristics are more effective than others because they place downstream computations on resources such that the cumulative cost of transferring the associated files and estimated computation time is minimized [6,9]. This improves the overall application performance by opportunistically migrating downstream sub-computations to better resources. However, the amount of data generated by multiple sub-computations may exceed the storage limits set on resources, which may restrict the further scheduling of jobs to those resources. To solve this problem, we built a garbage collection technique that keeps the amount of generated data within the permitted storage limits, even when multiple applications are simultaneously executed on a single resource.

Trust management: Large-scale applications require as many computational resources as possible to achieve high throughput. Normally, a single *virtual organization* (VO) [1] is formed by several entities with common goals and an agreement to share computational and data resources. However,



one VO alone may not provide sufficient resources for particularly large projects; hence, it may be necessary for a Grid testbed to span multiple VOs. Typically, Grid resources belonging to a particular VO only trust users authorized by the *Certificate Authority* (CA) of that VO. As a result, data transfers across resources, often referred to as *third party* transfers, may not be possible because those resources belong to different VOs. We encountered such difficulties in the Australian climate studies discussed throughout this paper. To address this, we have implemented a trust delegation model that facilitates flexible third party data transfers across different VOs.

The devised techniques are implemented in Nimrod/G, a tool that executes Grid-based PSAs [1]. Nimrod/G is a parameter sweep middleware that includes the ability to describe PSAs and launch them across a Grid testbed using a meta-scheduler. Apart from the ability to launch parameterized instances as jobs across the testbed, Nimrod/G is able to manage their execution dynamically with the ability to respond to errors on the network or by individual nodes on the testbed. As a result, Nimrod/G gracefully migrates the downstream sub-computations to better resources when a job or resource fails or when there is resource contention. In addition, Nimrod/G efficiently co-schedules multiple computations on a single resource, while keeping the cumulative data generated by these computations within permitted storage limits. Furthermore, results show that, while migrating sub-computations to resources of different VOs, Nimrod/G appropriately copies the state files associated with every sub-computation.

The next section describes the challenges encountered while executing large PSAs on a multi-VO Grid. Section 3 describes our implemented techniques and the resulting improvements to the application model. Section 4 illustrates the efficacy of the implementation through experimental analysis. In Section 5, we summarize the lessons that we have learned during our experience with the real-world climate studies related in this paper. In Section 6, we briefly present related works. Section 7 concludes the paper and mentions some of our future endeavours.

2. CHALLENGES

The experiment described in this paper studies the effects of the burning of the Savannah on the climate of Northern Australia [10], using the Conformal-Cubic Atmospheric Model (CCAM) [11]. The inputs to CCAM, which include a set of files, can be tuned to simulate different climatic scenarios of a particular geographic location. For example, one of the parameters, the time parameter, specifies the duration for which the analysis should be performed on a particular geographical location. We generated an order of 90 instances of CCAM for the experiment [12]. We describe the challenges encountered in task, data and trust management below.

2.1. Task management

The application was initially set up as shown in Figure 1. The same application executes as 90 separate instances, each with a distinct set of parameter values stored within distinct files. Each instance takes such a file as input, as well as some common data files that do not vary, computes a scenario and then generates an output for further investigation. Each computation modelled 21 years of weather conditions requiring 6–7 weeks of CPU time to complete. Nimrod/G harnesses several computational nodes in order to execute these instances in parallel as distinct jobs. However, as

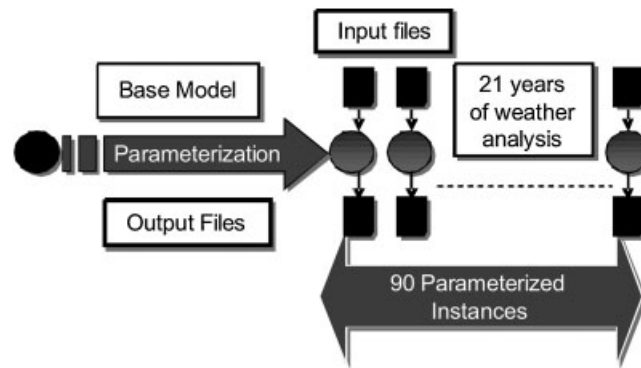


Figure 1. Old application model.

mentioned earlier, executing such long running jobs as a single, continuous run is prone to failures. Apart from possible I/O or processing failures, network failures will also disrupt the execution of those jobs. Although Nimrod/G is able to restart failed jobs, or start them on another node if the previous node had failed, restarting a long running computation (6–7 weeks in our case) due to a job or resource failure would waste huge amounts of time already invested in the failed run. Additional time is also incurred to re-run the computation from the very beginning, particularly when the job requires pre-processing or if it must be submitted to a new computational resource instead. We therefore addressed this issue with long duration jobs by *task-splitting*, structuring each as a group of 252 subtasks in a sequential workflow. The CCAM application must process data from 252 months in sequence, whereas each month's data input data can be treated independent of others. Each month, therefore, was used as a *time step* within each of the 90 workflows. The challenge is therefore twofold, in terms of application design and check-point process. The CCAM application was neither originally designed for this purpose nor did the PSA specification include sequential workflow features to begin with. At the same time, while the notion of check-pointing is well known, the fact that we are check-pointing subtasks within a sequential workflow requires special consideration of the dependencies among subtasks.

2.2. Data management

Task-splitting, as described earlier, generally adds to the data management problem. Each subtask must generate persistent output that will be taken as input by the next subtask in the sequential workflow. In comparison, if we do not exercise task-splitting, then each long-duration job will handle data internally, e.g. as data in the memory allocated to the running job. This is obviously simpler and, in fact, quicker in the absence of file I/O. On the other hand, appropriate task allocation techniques can improve an application's performance by migrating the downstream computations to resources with better CPU performance and network connectivity. This provides the additional advantage of dynamic and adaptive resource allocation in response to drastic changes in resource conditions across the Grid testbed, such that subtasks can be initiated or migrated across to better resources. On the other hand, these subtasks must have access to the data they require. Although



convenient file transfer mechanisms are available to Nimrod/G, user applications can easily conflict with disk quotas. Existing heuristics assume unlimited disk space on Grid resources [13,14], which is unrealistic on Grid testbeds where users are typically allocated limited disk space. In our test case, each subtask generated about 70 MB of output. Combined with the common input files of around 250 MB in size, this exceeded the storage available on some testbed resources. On the other hand, much of the output data are only required temporarily as the output of one subtask is required only as input to the next sequential task. Therefore, it is not a straightforward case of requiring a more generous disk quota, something that is not always possible to begin with. What is required is an appropriate data management technique that will properly delete files that become obsolete as the Grid workflow progresses.

2.3. Trust management

To deal with the scale of our application, we executed our experiments on the Pacific Rim Applications and Grid Middleware Assembly or PRAGMA Grid testbed [10]. The PRAGMA testbed consists of a large set of resources spanning across several VOs. The Globus toolkit [1] was used to build the basic infrastructure for providing resource access and remote data transfer services.

Figure 2 shows a sample of such Grid testbed models integrated from several individual VOs. Table I lists the exclusive CA domains that we involved in the testbed. Typically, all resources in an individual VO trust a single CA [1], and thus all resources can communicate freely and authenticate data transfers against this single CA. This is important when intermediate files are transferred directly from one resource to another as part of a data-dependent computation within the PSA. However, if the testbed spans multiple VOs as discussed, then a sending resource and a receiving resource may trust different CAs, making third party transfers difficult. One solution to this problem is to route the data through a resource trusted by both parties. However, this incurs additional communication overhead, as the data now need to be transferred through an intermediary resource. To address this, we have designed and implemented a new mechanism that delegates trust across multiple VOs, allowing the direct transfer of data between resources across different VOs despite having mutually exclusive CA domains.

A relevant but separate challenge is the issue of internetworking restrictions. Single VOs involve independent real-world organizations, for which the underlying internetworking is typically built on top of the Internet. The risky nature of the Internet provides the motivation for using firewalls to protect the networks within these organizations. The restrictions caused by firewalls sometimes result in the failure of Grid transactions. Note that firewall issues are distinct from trust issues addressed by CA domain infrastructures. In the latter context, trust domains are represented and addressed within VOs, with authorization based on certificates signed by the CA [1]. Firewall issues reflect the autonomy of *real-world* organizations, not virtual ones. Firewall restrictions are not based on certificates, therefore, but at lower levels of the network infrastructure, typically at the transport (TCP or UDP ports) or internet levels (IP addresses). We address firewall issues in a separate project, *Remus*, which reroutes connections through tunnels and proxies using the most appropriate transport mechanism for a given firewall situation [15]. However, these issues are beyond the scope of this paper, as we did not encounter them in the testbed described here. Despite the involvement of different institutions across the U.S., Japan, Taiwan, Thailand and Australia, we did not have to deal with firewalls at all.

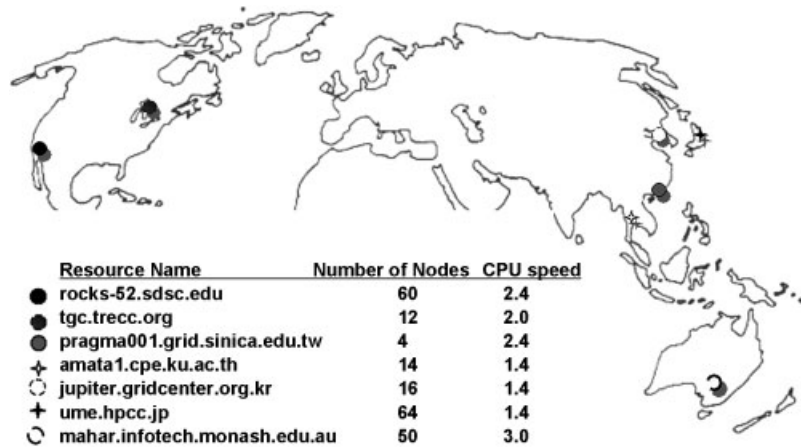


Figure 2. Grid testbed model.

Table I. List of CA domains involved in the climate study described in this paper.

CA domain	Region
Australian Partnership for Advanced Computing (APAC)	Australia
National Partnership for Advanced Computational Infrastructure (NPACI)	U.S.
National Institute of Advanced Industrial Science and Technology (AIST)	Japan
Korea Institute of Science and Technology Information (KISTI)	Korea
Academia Sinica (AS)	Taiwan
National Center for Supercomputing Applications	U.S.

3. SOLUTIONS

In this section, we present the solutions that we came up with in order to meet the challenges described previously.

3.1. Task management

Many scientific applications, such as CCAM, provide explicit check-pointing at the application level. In our case, we split large simulations, each computing 21 years of weather scenarios, into a sequence of 252[‡] dependent monthly sub-computations, linked through *state files* containing check-pointed data (Figure 3). A state file is an intermediary file containing the state information from one subtask that is to be used by its subsequent child. As stated earlier, challenges in task management are in modelling the task-splitting as well as in handling the dependencies across subtasks.

[‡]21 yearly computations * 12 months = 252 monthly computations.

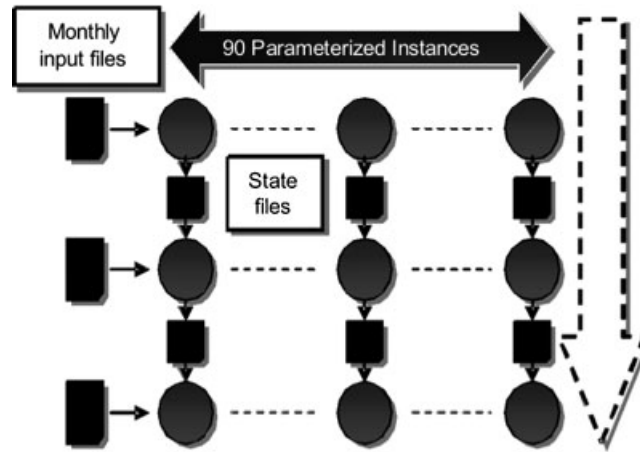


Figure 3. New application model with sequential dependencies.

It is difficult to express task-splitting as described in this paper succinctly using existing parametric modelling languages embedded in existing tools such as GridBus [7], ICENI [16] and earlier versions of Nimrod/G [1]. Other declarative languages, such as APST [17], use comprehensive but large XML-based descriptions to specify experiments [18].

Nimrod/G uses a simple declarative language, called *planfile*, to specify parametric modelling experiments. Figure 4 shows the syntax of the current implementation of what we refer to as a *planfile*. Normally, Nimrod/G takes the Cartesian product of the declared parameters and creates a parametric hyperspace of Cartesian points[§], each representing a single task. To specify check-point dependencies in parametric spaces, we introduce the *seqameter* (Figure 4, line 3.3), i.e. a *sequential* parameter. A seqameter behaves like a normal parameter, but its inclusion breaks a parametric instance into a set of temporally dependent sub-instances (see Figure 3). In our case study, the values in the seqameter range distinguish one subtask from another within a set of subtasks that must execute in sequence. In comparison, the values in the parameter space distinguish one independent parametric instance from another, making them distinct as they may execute in parallel. We have integrated seqameters in Nimrod/G's planfile syntax, which provide explicit control to the user in setting the granularity of the check-points.

As seen in Figure 3, instead of having 90 jobs for 90 parametric instances, we instead model the PSA as 90 sequences of subtasks. Each subtask receives monthly input files as well as initial data coming from the subtask above it, if any. Figure 4 describes the PSA and how seqameters are used. The 'expnr' integer parameter distinguishes each of the 90 parametric instances. The 'timestep' seqameter distinguishes, for each parametric instance, each of the 252 check-point steps: one for each month during the simulation period of 21 years. The various execute commands in the planfile use the parameter and seqameter values as references to the input and state files to use for the next subtask to execute. Note that the splitting process introduced dependencies among the subtasks.

[§] A Cartesian point can be defined as (a, b) where $a \in A$ and $b \in B$, A and B being the two parametric sets.



```

1. parameter expnr integer range from 1 to 90 step 1;
2. seqameter timestep integer range from 1 to 252 step 1;
3. task main
  3.1. inputs cms.${expnr}.`/bin/echo ${timestep} 1`.tar.gz
      vf_ccam.$timestep
  3.2. output cms.${expnr}.${timestep}.tar.gøn create f1
  3.3. sequemeters timestep
  3.4. parameters expnr
  3.5. node:execute tar xfz cms.${expnr}.*.tar.gz
  3.6. node:execute /bin/mv vf_ccam.$timestep
      cms.${expnr}/input/vf_ccam.$timestep
  3.7. node:execute cd cms.${expnr} && /cms_prep.ksh
      ${expnr}.${timestep} >> ../../output 2>&1 && cd..
  3.8. node:execute cd cms.${expnr}/output && .
      ${HOME}/bin/globpea >> ../../output 2>&1 && cd ../../
  3.9. node:execute cd cms.${expnr} && /cms_postp.ksh
      ${expnr}.${timestep} >> ../../output 2>&1 && cd..
  3.10. node:execute tar cfz cms.${expnr}.${timestep}.tar.gz
      cms.${expnr}
4. endtask
5. task f1
  5.1. parameters None
  5.2. seqameters None
  5.3. copy node:$FILE local,root:
  5.4. node:execute $HOME/Sreplicate s brecca -2.vpac.org Obj1
6. endtask

```

Figure 4. New planfile syntax.

As the existing heuristics [19] in Nimrod/G were incapable of handling such dependencies, new scheduling heuristics were implemented to schedule tasks to resources based on

- resource performance and network conditions;
- exploiting the parallelism among the application components and
- respecting both task precedence and data affinity.

We implemented some existing algorithms, namely Sufferage, XSufferage [6] and a LoadBalance heuristic, in the Nimrod/G scheduler. Each user-selected heuristic attempts to optimize task allocation with a specific objective function. Sufferage allocates a node to a task that would ‘suffer’ most if it is not allocated to that node. XSufferage is the cluster-level implementation of Sufferage. LoadBalance distributes equal computational workload across testbed nodes with homogeneous CPU configuration and network connectivity. Sufferage and XSufferage require information about the network links and computational resource performance, which can be obtained from resource monitoring tools such as the *network weather service* (NWS) [20] and Ganglia [21]. Apart from this information, statistical logs about the past performance of the resources can also be used in order to optimize task allocation. Nimrod/G uses both NWS traces and statistical logs to optimize task scheduling. It is therefore necessary to assume that the Grid resources support both NWS and other forms of statistical logging.



3.2. Data management

As mentioned earlier, Nimrod/G provides mechanisms for PSAs to conveniently transfer files across the testbed, e.g. the ‘copy’ command. In addition, the planfile syntax allows for the invocation of command line operations or the execution of system binaries. In Figure 4, apart from being able to execute shell scripts included in the application, the plan file may also cause the execution of Unix utilities such as ‘tar’ for archiving or unarchiving files. However, users are occasionally interested in performing operations on the state files, for example, backup instructions to store the file at a particular storage resource broker [22] location. Nimrod/G can be instructed to do this using the new planfile syntax. A specialized Nimrod task can be associated with these files, which can then be executed when a specific event occurs, such as file generation or copying (Figure 4, line 3.2). This function is declared in the plan file (Figure 4, lines 5–6) as a normal task but it consists of some extra processing instructions.

If specialized operations are not associated with the state files, then those files are removed once the subtask requiring them finishes its execution. Similarly, all other files that are not required by any other task as an input could be removed from the resources to free up some space. We therefore implemented garbage collection to remove all such files to comply with storage restrictions. This allowed unrestricted scheduling of multiple jobs on the same resource without the scheduler having to bother with space constraints. Collecting garbage in this manner solved data management problems to a large extent.

3.3. Trust management

Coupling resources belonging to multiple VOs together provides a larger resource pool than what would be available in a single VO. However, in multi-VO Grid testbeds, the scheduler might allocate a task to a resource within a VO that is different from that of its parent. As the parent’s state file must be passed on to its child subtask, a third party transfer across resources in different VOs will become necessary. As described earlier, third party data transfers across VOs cannot proceed unless trust conflicts are first resolved. In principle, if the user is authorized to access resources from two different VOs, then it should be possible to perform third party data transfers between the resources. In the Globus Security Infrastructure (GSI) [1], a client must possess legitimate credentials that are validated against the CA trusted by the resource to be accessed. To access resources from different VOs, and thus, across different CAs, distinct credentials valid for each CA are required. Each valid credential is the basis for a *proxy certificate* that authenticates the user and authorizes access to resources of the associated VO.

Proxy certificates, which can be used to perform third party transfers, are available to Nimrod/G. However, Nimrod/G supports a range of different Grid middleware apart from Globus. For example, Nimrod/G currently supports file transfers across resources managed by Condor [13], PBS [23], Sun GridEngine [24], Bproc [25] and Fork. We therefore implement the third party transfer mechanism within Nimrod/G to transcend the heterogeneity among transfer methods that might otherwise be involved. Third party transfers are implemented using *Nimrod agents* and *Nimrod Remote File Servers* and secured through public key authentication. Nimrod/G manages the execution of tasks using Nimrod agents that run on the resource where a user’s tasks are. Nimrod agents also manage data transfers normally required by tasks; however, third party transfers require Nimrod remote file



servers. To validate transfer requests from Nimrod agents involving a given resource, a remote file server is launched on it. Distinct session keys for each resource, say x and y , are generated and maintained at the user's local site in the Nimrod database. In order to perform the transfer from resource x to y , the session key required to access resource y is copied to x . The Nimrod agent on x then uses the copied key, on behalf of the user, to validate the request with the Nimrod remote file server on y . Once validated, the Nimrod remote file server facilitates the transfer operation.

In this manner, we provide a secure third party mechanism across resources from different VOs, which may support different standards. The actual transfer method relies on Nimrod/G, without having to introduce another delegation layer such as WebCom's CCA [26]. Note that the user does not need to install Nimrod/G on remote resources. Besides, Nimrod/G performs these operations by launching remote agents and file servers on distributed resources. We are thus able to use resources belonging to multiple VOs as a single 'Grid,' leveraging its distributed storage capacity.

4. EXPERIMENTS AND RESULTS

Here, we present the results of the Savannah experiment on the PRAGMA [26] testbed (Figure 5), consisting of seven resources from six different countries. Each resource had its own CA domain and showed dynamic availability patterns during the experiment.

4.1. Task management

Figure 6 shows the state transition diagrams of two sub-computations. The upper half of each state node shows the allocated resource, whereas the lower half describes the current state of the job. The sub-computations were migrated from one resource to another mainly due to

- Grid resource failures, when the resource that was allocated to the job eventually failed.
- Input file copy failures, when the file copy failed due to resource failure or network failure.
- Resource unavailability, when the front node's job queues were too long or the resource was too busy.
- Job failures, when the job failed due to missing libraries or some unexpected runtime error.
- Disk space shortage, when the input files could not be copied to the scheduled resource due to disk space limitations.

Nimrod/G is designed to migrate the jobs to other available resources to recover from these faults. Any of the faults above invariably results in the failure of Nimrod to guarantee a working environment in computational nodes. Computational nodes are remotely managed through Nimrod agents. Faults will be relayed by agents to the Nimrod scheduler or will cause agents themselves to fail and lose contact with the scheduler and other server processes. Either way, Nimrod/G reacts by presuming that the associated jobs have failed and then launching them on other nodes. Furthermore, the heuristics implemented in Nimrod/G provides a range of objective functions to optimize scheduling.

Such fault recovery and adaptability are difficult to achieve if the applications, of the scale described in this paper, were executed under the conventional parameter sweep model.

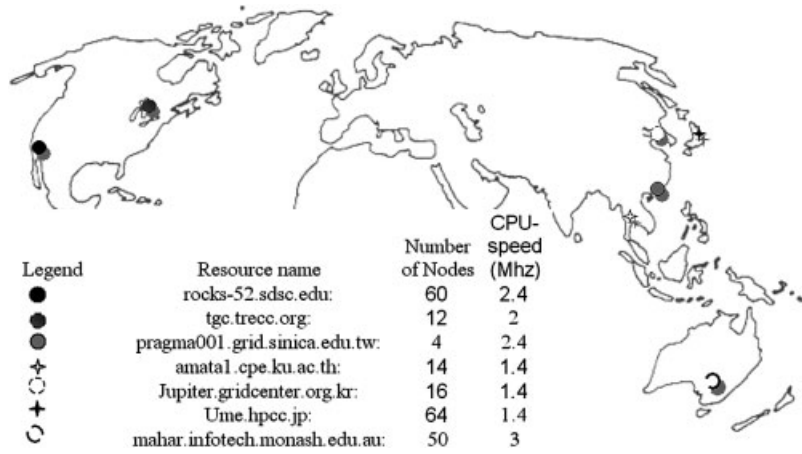


Figure 5. Testbed configuration.

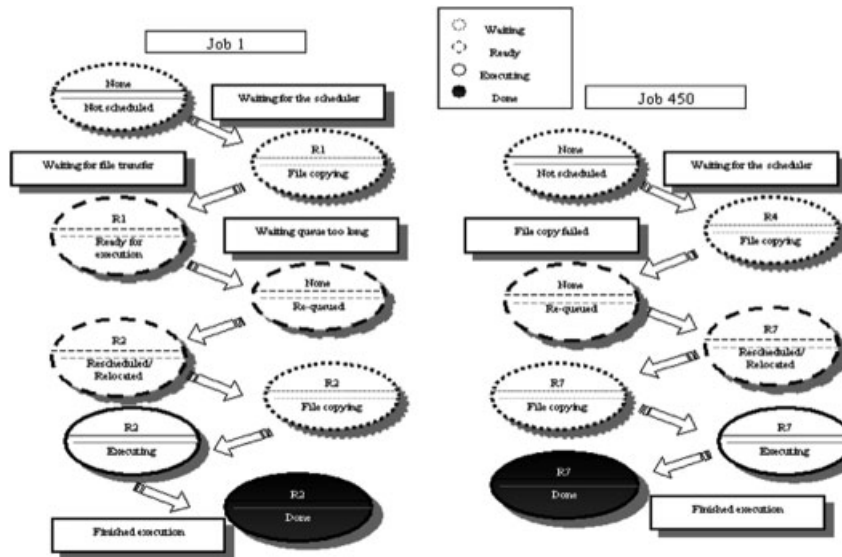


Figure 6. Diagrams of two sub-computations.

4.2. Data management

Figure 7 shows the disk space usage of the experiment running on different resources. The graph, which covers a few days of the whole experiment, illustrates dynamic data management activities

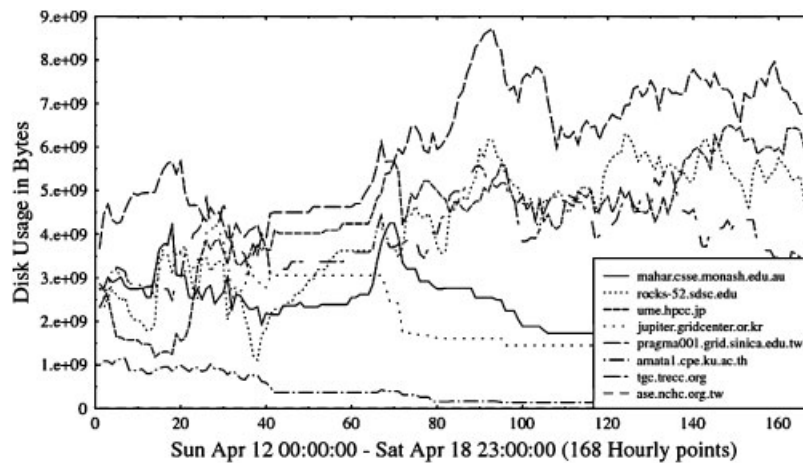


Figure 7. Disk space usage patterns.

as disk usage grows and shrinks. An increase in the y-coordinate represents data accumulation, whereas a decrease represents file removal. In the Savannah experiment, the disk space usage was directly proportional to the number of jobs executing on the resource. Therefore, a low y-coordinate (disk usage) value in the graph means that fewer jobs were executed on that resource at that moment. An oscillation shows that the data are being generated and removed uniformly. The job vs data relationship is important in order to understand the following behavioural analysis.

It can be observed that before the 70th hour (approximately), all resources had a smooth oscillatory behaviour, but the disk usage of *jupiter.gridcenter.or.kr*, *mahar.csse.monash.edu.au* and *amata1.cpe.ku.ac.th* decreased afterwards, because these resources became unavailable. The constant disk usage at the end of the graph is due to the presence of common files that were retained on the resources for subsequent use by other computations. Owing to resource unavailability, the jobs were distributed among other available resources, which is evident from the increase in their disk usage patterns. Note how resources such as *ume.hpcc.jp* have shown a more noticeable increase in disk usage compared with others. This happened because these resources had a higher number of computational nodes available, which made them a preferable choice for job allocation. Note as well that the oscillation in disk usage curves became quite uniform after the 100th hour when the load was distributed again properly. This also illustrates the effectiveness of the garbage collection technique.

4.3. Trust management

As mentioned before, we had 90 parametric instances of the main model, which were broken into a sequence of 252 sub-computations and interconnected by the state files (see Figure 3). In this set-up, every successive sub-computation required the state file generated by its immediate predecessor as its input for execution. Each line in Figure 8 represents the first 20 sub-computations of randomly chosen parametric instances. It can be observed that sub-computations, belonging to a particular

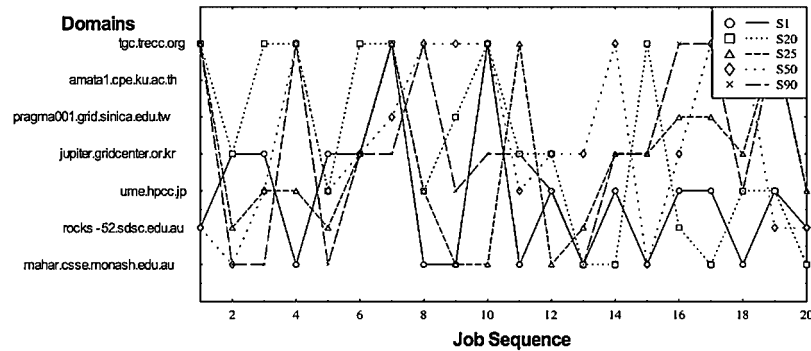


Figure 8. Inter-domain third party file transfers.

instance, were scheduled across seven different resources with exclusive CA domains. The situation required the state files to be copied accordingly (more resources were added at a later stage of the experiment). For example, to execute the 12th sequence of the 25th parametric instance on mahar.csse.monash.edu.au, the state file generated by the 11th sequence was copied from domain A (tgc.trecc.org) to domain B (mahar.csse.monash.edu.au). We performed file transfers using our third party communications mechanism, which allowed us to maximize the usage of the multi-VO Grid testbed.

The random load-balancing scheduler was used in the experiments. This was because certain mechanisms required by more complex schedulers to gather network and load statistics were not uniformly available across the testbed. For example, NWS was not supported on PRAGMA resources at that time, resulting in the under- or non-allocation of those resources by the scheduler. Other means of obtaining statistical data were also thwarted in various ways, given the non-uniformity of the resources available to us on the multi-VO testbed. Such issues are not unexpected in such a large and heterogeneous collection. It was therefore impossible for us to rely on the more sophisticated algorithms such as XSufferage. The latter would have been ideal as this algorithm considers both estimated file transfer time from source to target (resource) and the estimated execution time. This is crucial as rescheduling a job means migrating both the job and its input files. However, our attempts to use XSufferage failed to deliver optimal schedules given the lack of accurate statistical information, despite attempts to modify it to suit the suboptimal situation. We therefore employed our scheduling technique that was mostly but not completely arbitrary in allocating resources. Jobs generated from task-splitting were actually sorted by priority first, and then they were scheduled randomly on resources other than where their respective parents executed. This is not ideal but nevertheless achieves a satisfactory distribution across the whole testbed.

In any case, had XSufferage or similar APST algorithms been used, the fact is that third party transfers would still be required, simply because the resources on this testbed come from independent CA domains. The only reason for such transfers to become unnecessary is if all the tasks were scheduled within one VO, something that would reflect a poor degree of distribution and load balancing. Using the random load balancer, we achieved an acceptable level of resource utilization across all VOs involved.



5. SOME LESSONS LEARNED

Apart from the strategies and solutions discussed above, other problems were identified in the course of the climate study described in this paper. Some of the lessons we have learned are briefly presented below:

- *Task specification*: Application specification languages should be developed, which could succinctly describe complex e-science problems and simplify the specification of PSAs. For PSAs, the specification of parametric instances should be intuitive, as in our planfile syntax, rather than requiring an independent specification for each parametric instance, in the same way that one might specify independent and unrelated jobs. Although the utilization of parameters and seqameters as indices of input and state files may seem unconventional, it does present a very concise mechanism that lends to easy use for e-science applications.
- *Application deployment*: The complexity of Grid testbeds should be hidden, such that the deployment of PSAs should not require significant changes to the way the applications are specified or launched. We sought to achieve this with the very simple design of the planfile syntax, even when we included seqameters: there is no marked difference in deployment despite the ability to perform task-splitting across sequential subtasks.
- *Garbage collection*: Garbage collection and relocation are equally important under storage constraints. It is very common in the e-science domain to perform garbage collection manually. However, it should be an integral part of the middleware itself, as it is aware of the data dependencies among the tasks precisely.
- *Trust delegation*: The trust delegation layer implemented in what we consider lower middleware, Globus, should be extended to provide support for third party communication. The solution provided in this paper can be considered a high-level solution, given that Nimrod/G, one layer above Globus, provides the third party transfer mechanism. Similar methods would have to be integrated with similar middleware. However, if Globus and other lower middleware could support such a mechanism, then upper middleware layers can focus on other issues, such as scheduling.

The above lessons clearly point towards more work in this problem space, constituting practical work as well as fundamental research for the future. Based on the above lessons, we have set new goals as described in Section 7.

6. RELATED WORK

A number of projects complement our work. The Apples Parameter Sweep Template (APST) [17] is a parametric modelling tool that allows the scheduling of complex workflows on the Grid, specified using large APST XML descriptions [18]. Such descriptions are unlike the simple planfile syntax developed for Nimrod/G and subsequently used by GridBus [7]. The APST scheduling model also assumes unlimited disk storage and single VO testbeds. GridBus' core technology is quite similar to that of Nimrod/G, but the former assumes unlimited disk storage and single-domain testbeds. ICENI [16] provides a dynamic service-oriented framework for Grid environments that are restricted to single domains, without any task-splitting and garbage collection.



7. CONCLUSION AND FUTURE WORK

In this paper, we have presented a new task, data and trust management techniques to deploy large-scale applications on the multi-VO Grid testbed. These techniques were implemented during the first phase of the experiment involving the execution of large PSAs to study how the burning of Savannah affects the climate of Northern Australia. The implemented techniques include

- A task-splitting and check-pointing approach to break up large PSAs into a sequence of dependent subtasks for better fault recovery and improved application performance.
- A garbage collection technique to delete unnecessary data to comply with storage limits.
- A third-party file communication technique that delegates trust to agents deployed across multiple VOs to perform cross-domain file transfers.

Results show that Nimrod/G migrates sequential sub-computations to better resources when failures occur, while still executing as many tasks in parallel as possible. Moreover, disk usage patterns on computational resources illustrate that Nimrod/G efficiently co-schedules multiple computations on a single resource. However, Nimrod/G restricts the amount of cumulative data generated by these computations within permitted storage limits. Further results show that while migrating the downstream sub-computations across resources of multiple VOs, Nimrod/G effectively copies the state files along with other input files required by individual sub-computations to appropriate resources.

Future work we will set out to undertake include the following:

- specialized CPU and network state monitoring to optimize task scheduling;
- an intelligent back-up and restore mechanism to minimize state back-up costs and to improve fault recovery;
- implementation of novel clustering-based heuristics to optimize scheduling of dependent tasks and
- development of new models of computation for describing scientific applications, and new scheduling algorithms for exploiting concurrency between application components.

ACKNOWLEDGEMENTS

The Australian Research Council under an e-Research special initiative grant supports this work. We wish to thank Amanda Lynch and her colleagues for their assistance in running the climate study. We also acknowledge the CSIRO Division of Atmospheric Research who provided the computational models. We thank our PRAGMA partners for the resources used in the testbed.

REFERENCES

1. Abramson D, Giddy J, Foster I, Kotler L. High performance parametric modeling with Nimrod/G: Killer application for the global Grid? *International Parallel and Distributed Processing Symposium (IPDPS)*, Cancun, Mexico, May 2000; 520–528.
2. Amsden A, Ramshaw J, Rourke OP, Kiva DJ. A computer program for 2- and 3-dimensional fluid flows with chemical reactions and fuel sprays. *LASL Report: LA-10245-MS*, Los Alamos National Laboratory, 1985.
3. Basney J, Livny M, Mazzanti P. Harnessing the capacity of computational grids for high energy physics. *Conference on Computing in High Energy and Nuclear Physics*, Padova, Italy, 2000.



4. Rogers S. A comparison of implicit schemes for the incompressible Navier–Stokes equations with artificial compressibility. *AIAA Journal* 1995; **33**(10):2066–2072.
5. Stiles JR, Bartol TM, Salpeter EE, Salpeter MM. Monte Carlo simulation of neuromuscular transmitter release using MCell, a general simulator of cellular physiological processes. *Computational Neuroscience*, Bower JM (ed.). Plenum Press: New York, 1998; 279–284.
6. Casanova H, Berman F, Zagorodnov D, Legrand A. Heuristics for scheduling parameter sweep applications in Grid environments. *Ninth Heterogeneous Computing Workshop (HCW00)*, Cancun, Mexico, May 2000; 349–363.
7. Buyya R, Venugopal S. The Gridbus toolkit for service oriented grid and utility computing: An overview and status report. *The 1st IEEE International Workshop on Grid Economics and Business Models (GECON 2004)*, New Jersey, 2004.
8. Tamir Y, Sequin CH. Error recovery in multicomputers using global checkpoints. *Proceedings of the International Conference on Parallel Processing*, Bellaire, MI, 1984; 32–41.
9. Meyer LAVC, Annis J, Wilde M, Mattoso M, Foster I. Planning spatial workflows to optimize grid performance. *ACM Symposium on Applied Computing, Proceedings of the 2006 ACM Symposium on Applied Computing (SAC 2006)*, Dijon, France, 2006; 786–790.
10. Zheng C, Abramson D, Arzberger P, Ayyub S, Enticott C, Garic S, Katz M, Kwak J, Lee BS, Papadopoulos P, Phatanapherom S, Sriprayoosakul S, Tanaka Y, Tanimura Y, Tatebe O, Uthayopas P. The PRAGMA testbed-building a multi-application international Grid. *Sixth IEEE International Symposium on Cluster Computing and the Grid Workshops (CCGRIDW'06)*, Singapore, May 2006; 57.
11. McGregor JL, Nguyen KC, Katzfey JJ. Regional climate simulations using a stretched-grid global model. *Report No. 32, WMO/TD-No. 1105*, Research Activities in Atmospheric and Oceanic Modelling, Geneva, 2002; 3.15–3.16.
12. Beringer J, Hutley L, Lynch A, Siems S, Tapper N. The Savanna Fire Experiment (SAFE). <http://arts.monash.edu.au/ges/research/climate/fire/safe.php> [1 May 2008].
13. Buyya R, Abramson D, Giddy J. An evaluation of economy-based resource trading and scheduling on computational power Grids for parameter sweep applications. *Workshop on Active Middleware Services (AMS 2000)* (in conjunction with *Ninth IEEE International Symposium on High Performance Distributed Computing*). Kluwer Academic Press: Dordrecht, 2000.
14. Abramson D, Buyya R, Giddy J. A computational economy for Grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems* 2002; **18**(8):1061–1074.
15. Tan J, Abramson D, Enticott C. Bridging organizational network boundaries on the grid. *IEEE Grid 2005*, Seattle, November 2005.
16. Gulamali MY, McGough AS, Newhouse SJ, Darlington J. Using ICENI to run parameter sweep applications across multiple Grid resources. *Case Studies on Grid Applications Workshop, Global Grid Forum 10*, Berlin, Germany, March 2004.
17. Casanova H, Obertelli G, Berman F, Wolski R. The AppLeS parameter sweep template: User-level middleware for the Grid. *Proceedings of the Super Computing Conference (SC'2000)*, Dallas, TX, 2001.
18. Casanova H. APST XML Man Page, 2001, http://grail.sdsc.edu/projects/apst/apst.xml_man.html.
19. Buyya R, Abramson D, Murshed M. A deadline and budget constrained cost-time optimization algorithm for scheduling task farming applications on global Grids. *The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, NV, 2002.
20. Wolski R, Spring NT, Hayes J. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems* 1999; **15**:757–768.
21. Massie ML, Chun BN, Culler D. The Ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing* 2004; **30**(7):817–840.
22. Chaitanya B, Moore R, Rajasekar A, Wan M. The SDSC storage resource broker. *Proceedings of CASCON'98*, 30 November–3 December, Toronto, Canada, 1998.
23. The Portable Batch System (OpenPBS). http://www.pbsgridworks.com/PBSTemp1.3.aspx?top_nav_name=Products&item_name=OpenPBS&top_nav_str=1 [1 May 2008].
24. Gentzsh W. Sun grid engine: Towards creating a compute power grid. *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2001; 35.
25. Hendriks EA. BProc: The Beowulf distributed process space. *The 16th International Conference on Supercomputing*, New York, NY, 2002; 129–136.
26. O'Callaghan DW, Coghlan BA. Bridging secure WebCom and European DataGrid security for multiple VOs over multiple Grids. *Proceedings of the Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (ISPD/HeteroPar'04)*, Washington, DC, 2004; 225–231.