The Virtual Laboratory: Enabling Molecular Modeling for Drug Design on the World Wide Grid

Rajkumar Buyya[†], Kim Branson[*], Jon Giddy[†], and David Abramson[†]

[†] School of Computer Science and Software Engg.
Monash University, Caulfield Campus
Melbourne, Australia
{rajkumar, davida, jon}@csse.monash.edu.au

[*] Structural Biology
Walter and Eliza Hall Institute
Royal Parade, Parkville, Melbourne
kbranson@wehi.edu.au

Abstract: Computational Grids are emerging as a new paradigm for sharing and aggregation of geographically distributed resources for solving large-scale computational and data intensive problems in science, engineering, and commerce. However, application development, resource management and scheduling in these environments is a complex undertaking. In this paper, we illustrate the development of a virtual laboratory environment by leveraging existing Grid technologies to enable molecular modeling for drug design on geographically distributed resources. It involves screening millions of compounds in the chemical database (CDB) against a protein target to identify those with potential use for drug design. We have used the Nimrod-G parameter specification language for transforming a molecular docking application as a parameter sweep application for executing on distributed systems. We have developed new tools for data management to organize and provide access to chemical databases as a network service in a scalable and distributed manner. They enable transparent access to molecule records in the CDB of interest from remote clients. The Nimrod-G resource broker along with molecule CDB service and access management infrastructure is used for scheduling and on-demand processing of docking jobs on the World-Wide Grid (WWG) resources. The results demonstrate the ease of use and power of the Nimrod-G and virtual laboratory tools for grid computing.

1 Introduction

Computational Grids [1] enable the sharing of a wide variety of geographically distributed resources including supercomputers, storage systems, databases, data sources, and specialized devices owned by different organizations in order to create virtual enterprises and organizations. They allow *selection* and *aggregation* of distributed resources across multiple organizations for solving large-scale computational and data intensive problems in science, engineering, and commerce. The parallel processing of applications on wide-area distributed systems provide a scalable computing power. This enables exploration of large problems with huge data sets, which is essential for creating new insights into the problem. Molecular modeling for drug design is one the scientific applications that can benefit from the availability of a large computational capability.

Drug discovery is an extended process that can take as many as 15 years from the first compound synthesis in the laboratory until the therapeutic agent, or drug, is brought to market [11]. Reducing the research timeline in the discovery stage is a key priority for pharmaceutical companies worldwide. Many such companies are trying to achieve this goal through the application and integration of advanced technologies such as computational biology, chemistry, computer graphics, and high performance computing (HPC). Molecular modeling has emerged as a popular methodology for drug design—it can combine computational chemistry and computer graphics. Molecular modeling can be implemented as a master-worker parallel application, which can take advantage of HPC technologies such as clusters [2] and Grids for large-scale data exploration.

Drug design with molecular modeling technique involves screening millions of ligand[1] records or molecules of compounds in a chemical database (CDB) to identify those that are potential drugs. This

[1] An ion, a molecule, or a molecular group that binds to another chemical entity to form a larger complex.

process is called *molecular docking*. It helps scientists explore how two molecules, such as a drug and an enzyme or protein receptor, fit together (see Figure 1). As docking each molecule in the target chemical database is both a compute and data intensive task, it is our goal to use Grid technologies to provide cheap and efficient solutions for the execution of molecular docking tasks on large-scale, wide-area parallel and distributed systems.

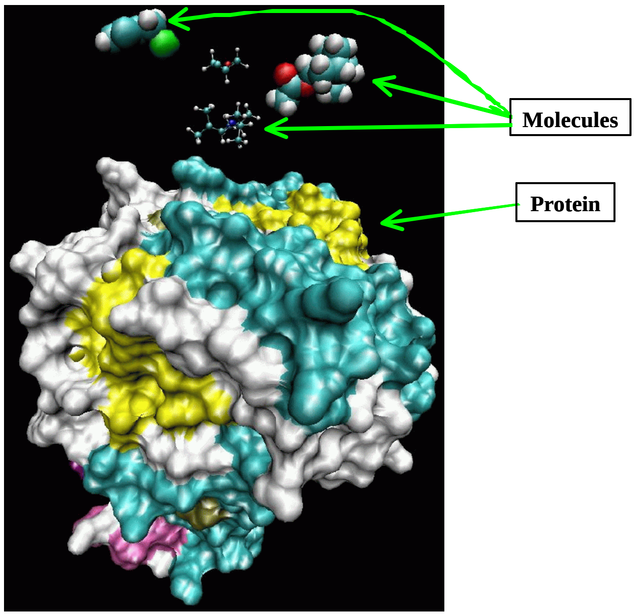


Figure 1: The X-ray crystal structure of a target receptor (surface rendered protein) and small molecules to be docked.

While performing Grid-based docking, the molecule structural information must be extracted from one of a number of large chemical databases located on different resources. Because the databases require storage space in the order of hundreds of megabytes to terabytes, it is not feasible to transfer the chemical database to all nodes in the Grid while processing. Therefore, access to a chemical database must be provided as *network service* (see Figure 2). Also, the chemical database needs to be selectively replicated on a few nodes within the Grid to avoid any bottleneck due to providing access to the database from a single source. Intelligent mechanisms (e.g., CDB broker) need to be supported for selecting optimal sources for CDB services depending on the location of resources selected for processing docking jobs.

Fundamentally, drug design is a *computational and data challenge* problem—it involves screening millions of compounds in chemical databases. Screening each compound, depending on structural complexity, can take from a few minutes to hours on a standard PC, which means screening all compounds in a single database can take years! For example, we are looking into one of the drug design problems that involve screening 180,000 compounds. Each job screening a compound is expected to take up to 3 hours of execution time on a desktop computer (e.g., Pentium-based Linux/Windows PC). That means, if we aim to screen all these compounds on a single PC, it can take up to 540000 hours, which is roughly equivalent to 61 years! If we use a typical cluster-based supercomputer with 64 nodes, we can solve this problem in one year. The problem can be solved with a large scale Grid of hundreds of supercomputers within a day. If we use a massive network of peer-to-peer style Grid computing infrastructure such as SETI@Home [20], the drug discovery problem could be solved within a few hours.

The Virtual Laboratory tools transform molecular modeling application into a parameter sweep application for executing jobs docking molecules in the CDBs in parallel on distributed resources. The parameterized application contains multiple independent jobs, each screening different compounds to identify their drug potential. These jobs are computational intensive in nature and only a small proportion of the execution time is spent on data communication (e.g., fetching molecules information on demand from remote databases). Applications expressed with this task-farming computational model have high *computation to communication* ratio. Hence, they can tolerate high network latency, which makes them suitable for executing in parallel on Internet-wide distributed resources.

In this Virtual Laboratory project, we have investigated the requirements of executing the molecular modeling application on the Grid. We have leveraged existing Grid technologies and developed new tools that are essential for Grid enabling the chemical database and the docking application on distributed resources. In this paper, we discuss a layered architecture of technologies and tools for creating the virtual laboratory environment for drug design application. We present results of scheduling molecular docking application for processing on the WWG (World Wide Grid) testbed.

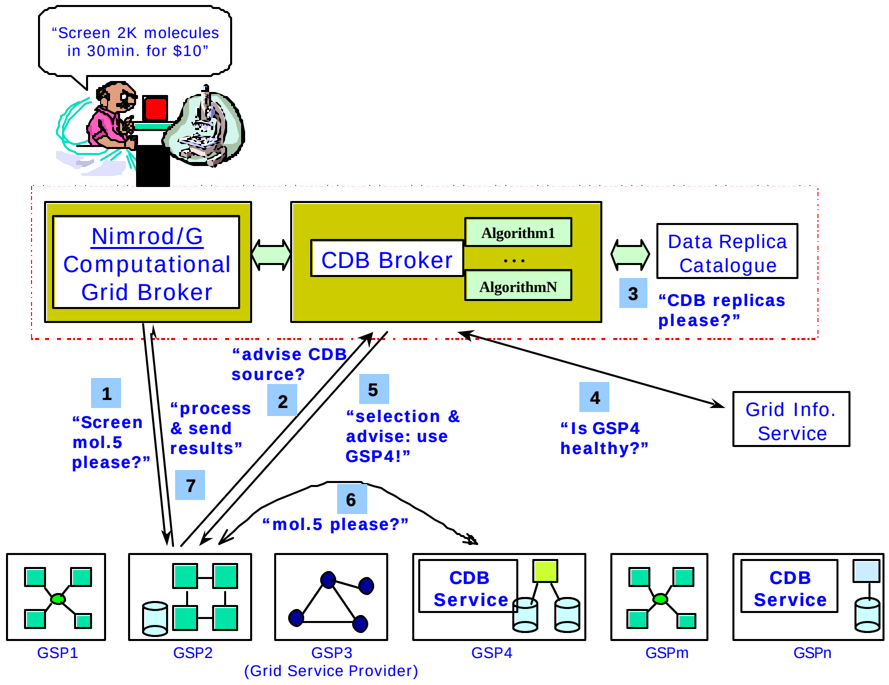


Figure 2: Resource brokering architecture for screening molecules on distributed resources.

2 Architecture – The Software Stack

The Virtual Laboratory builds on the existing Grid technologies and tools for performing data intensive computing on distributed resources. To provide a virtual computing environment for applications, depending on their requirements, we have developed the necessary tools. For example, to enable drug design application docking molecules in remote databases, we have developed tools for providing access to chemical databases as a network service. There are many scientific and commercial applications (e.g., molecular modeling, high-energy physics events processing, and financial investment risk-analysis) that explore range of scenarios. Instead of explicitly developing them as parallel applications using interfaces such as MPI, they can be composed as parameter sweep applications using tools such as Nimrod [3]. Such application jobs can be executed in parallel on distributed resources using the Nimrod-G resource broker (see Figure 2). A layered architecture and the software stack essential for performing molecular modeling on distributed resources is depicted in Figure 3. The components of Virtual Laboratory software stack are:

- The DOCK software for Molecular Modeling [15].
- The Nimrod Parameter Modeling Tools [3][17] for enabling DOCK as a parameter sweep application.
- The Nimrod-G Grid Resource Broker [4][5] for scheduling DOCK jobs on the Grid.
- Chemical Database (CDB) Management and Intelligent Access Tools:
    - CDB database Lookup/Index Table Generation.
    - CDB and associated index-table Replication.
    - CDB Replica Catalogue for CDB resource discovery.
    - CDB Servers for providing CDB services
    - CDB Brokering for selecting suitable (Replica Selection).
    - CDB Clients for fetching Molecule Record (Data Movement).

- The GrACE for resource trading toolkit [6].
- The Globus middleware for secure and uniform access to distributed resources [9].

The Grid resources (e.g., multiprocessors or clusters) at each location are generally presented as a single entity using resource management systems such as OS-fork, LSF, Condor, and SGE.

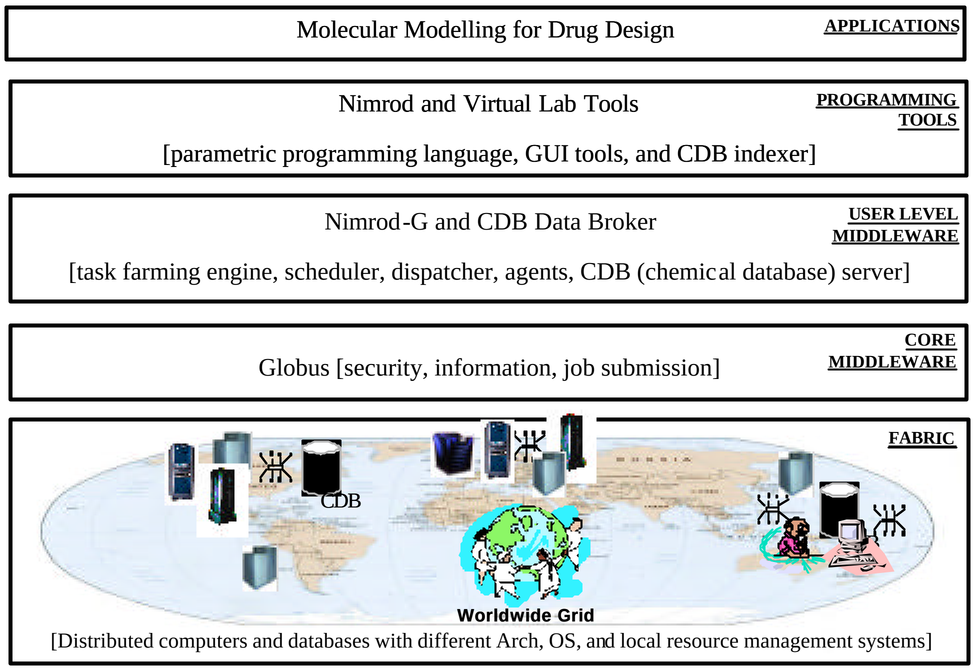

Molecular Modelling for Drug Design
APPLICATIONS
Nimrod and Virtual Lab Tools
[parametric programming language, GUI tools, and CDB indexer]
PROGRAMMING TOOLS
Nimrod-G and CDB Data Broker
[task farming engine, scheduler, dispatcher, agents, CDB (chemical database) server]
USER LEVEL MIDDLEWARE
Globus [security, information, job submission]
CORE MIDDLEWARE
FABRIC
CDB
Worldwide Grid
[Distributed computers and databases with different Arch, OS, and local resource management systems]

Figure 3: Layered architecture of Virtual Laboratory for drug design.

2.1 Docking Code

The original docking code developed by researchers at the University of California in San Francisco (UCSF) is one of the most popular molecular docking applications [10]. The program evaluates the chemical and geometric complementarities between a small molecule and a macromolecular binding site. It explores ways in which two molecules, such as a drug and an enzyme or protein receptor, might fit together. Compounds that might bind tightly to the target receptor must have complementary chemical and spatial natures. Thus docking can be seen as a three-dimensional puzzle searching for pieces that will fit into the receptor site. It is important to be able to identify small molecules (compounds), which may bind to a target macromolecule. This is because a compound, which binds to a biological macromolecule, may modulate its function, and with further development eventually become a drug candidate. An example of such a drug is the anti influenza drug Relenza which functions by binding to influenza virus attachment proteins thus preventing viral infection.

The relationship between the key programs in the dock suite is depicted in Figure 4. The receptor coordinates at the top represent the three-dimensional (3D) structure of a protein. The molecular modeller identifies the active site, and other sites of interest, and uses the program *sphgen* to generate the sphere centers, which fill the site [14]. The program *grid* generates the scoring grids [15]. The program *dock* matches spheres (generated by sphgen) with ligand atoms and uses scoring grids (from grid) to evaluate ligand orientations [14] [15]. It also minimizes energy-based scores [16] [6]. The focus of our work is on docking molecules in CDB with receptor to identify potential compounds that act as a *drug*. Hence, discussion in this paper is centered on the execution of the program *dock* as parameter sweep application on world-wide distributed resources.

The docking code is highly portable. We have been able to produce executables for Sun-Solaris, PC Linux, SGI IRIX, and Compaq Alpha/OSF1 architectures. For docking on heterogeneous resources, the Nimrod-G broker selects the right executable depending on the resource architecture.

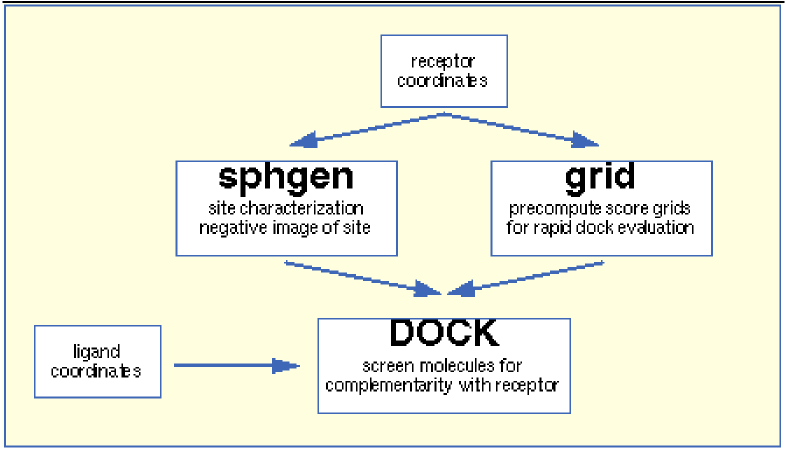

receptor coordinates
sphgen
site characterization negative image of site
grid
precompute score grids for rapid dock evaluation
ligand coordinates
DOCK
screen molecules for complementarity with receptor

Figure 4: Relation between key programs in the *dock* suite [10].

2.2 Nimrod-G Tools

Nimrod-G provides a suite of tools and services for creating parameter sweep applications, performing resource management, and scheduling applications. The Nimrod-G toolkit contains a simple declarative programming language and associated GUI tools for creating scripts and parameterization of application input data files, and a grid resource broker with programmable entities for processing jobs on grid resources.

Tools for Creating Parameter Sweep Applications

Nimrod supports declarative programming language and GUI-based tools that assist in the creation of parameter sweep applications [3]. They allow the user to: a) parameterise input files, b) prepare a plan file containing the commands that define parameters and their values, c) generate a run file, which converts the generic plan file to a detailed list of jobs, d) schedule jobs for processing on distributed machines, and e) control and monitor the execution of the jobs. The application execution environment handles online creation of input files and command line arguments through parameter substitution. The GUI tools supported by enFuzion, a commercial version of Nimrod, can also be used for parameterising applications. enFusion uses the same syntax as Nimrod for parameter specification [17]. Both Nimrod and enFuzion have been successfully used for performing parameter studies in a single administrative domain such as clusters. Nimrod-G [4][5] extends the capabilities of Nimrod and EnFuzion with the addition of powerful resource discovery, trading, and scheduling algorithms [7]. In Section 3, we discuss the capabilities of Nimrod-G tools by composing a molecular modeling program as a parameter sweep application for docking compounds in CDB databases and processing docking jobs on the Grid.

Nimrod-G Grid Resource Broker for scheduling DOCK jobs on Grid

The Nimrod-G Resource broker is responsible for determining the specific requirements that an experiment places on the Grid and performing resource discovery, scheduling, dispatching jobs to remote Grid nodes, starting and managing job execution, and gathering results back to the home node [5]. The sub-modules of our resource broker are, the task farming engine; the scheduler, a schedule advisor backed with scheduling algorithms, and a resource trading manager; a dispatcher and actuators for deploying agents on grid resources; and agents for managing execution of Nimrod-G jobs on grid resources.

The Nimrod-G grid explorer and dispatcher components are implemented using Globus services for resource discovery and initiating execution of Nimrod-G agents that take care of all the operations associated with the execution of an assigned job. The interaction between the components of the Nimrod-G runtime machinery and Grid services during runtime is shown in Figure 5. The Nimrod-G broker supports *deadline and budget constrained (DBC) scheduling* algorithms driven by a computational economy and user requirements [7]. In section 4, we discuss the results of Nimrod-G broker scheduling molecular modeling application on the Grid with DBC time and cost optimization scheduling algorithms.

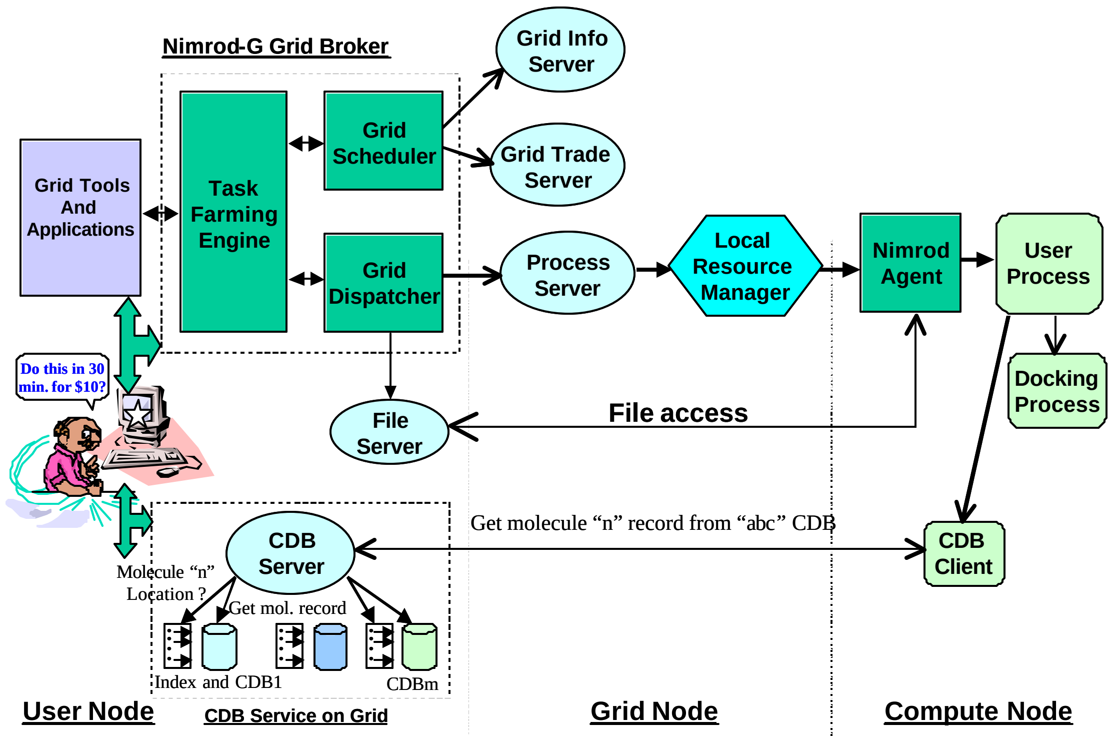


Figure 5: Deployment of Virtual Laboratory components at runtime and their interaction.

2.3 Chemical Database Management and Intelligent Access Tools

The chemical databases contain records of a large number of molecules from commercially available organic synthesis libraries, and natural product databases. The molecules are represented in MOL2 file (.mol2) format [12], which is a portable representation of a SYBYL [13] molecule. The MOL2 file is an ASCII file that contains all the information needed to reconstruct a SYBYL molecule. Each ligand record in a chemical database represents the three-dimensional structural information of a compound. The numbers of compounds in each CDB can be in the order of tens of thousands and the database size be anywhere from tens of Megabytes to Gigabytes and even Terabytes.

Instead of replicating the chemical database on every Grid node, we have developed a mechanism for replicating it on only a few selected nodes and then accessing ligand records in the database from remote machines. The CDB molecule server needs to provide serve requests of multiple users for molecule information. Hence, we have developed a multithreaded CDB server that can service requests from multiple users simultaneously. Instead of searching molecule records sequentially in a database represented using MOL2 format, we have built tools for creating index-tables for each CDB along with record size information. When a molecule record is requested, a CDB server first looks at the CDB index file to identify the record location and size and then directly reads the record from the CDB file. An interaction between a Grid node and a node running the CDB server while performing docking is shown in Figure 5. A detailed protocol level interaction between a client and the CDB server is shown in Figure 6.

It is possible to screen virtual combinatorial databases in their entirety. This methodology allows only the potential compounds to be subjected to physical (manual) screening and/or synthesis in laboratories, which is extremely time-consuming and resource-intensive.

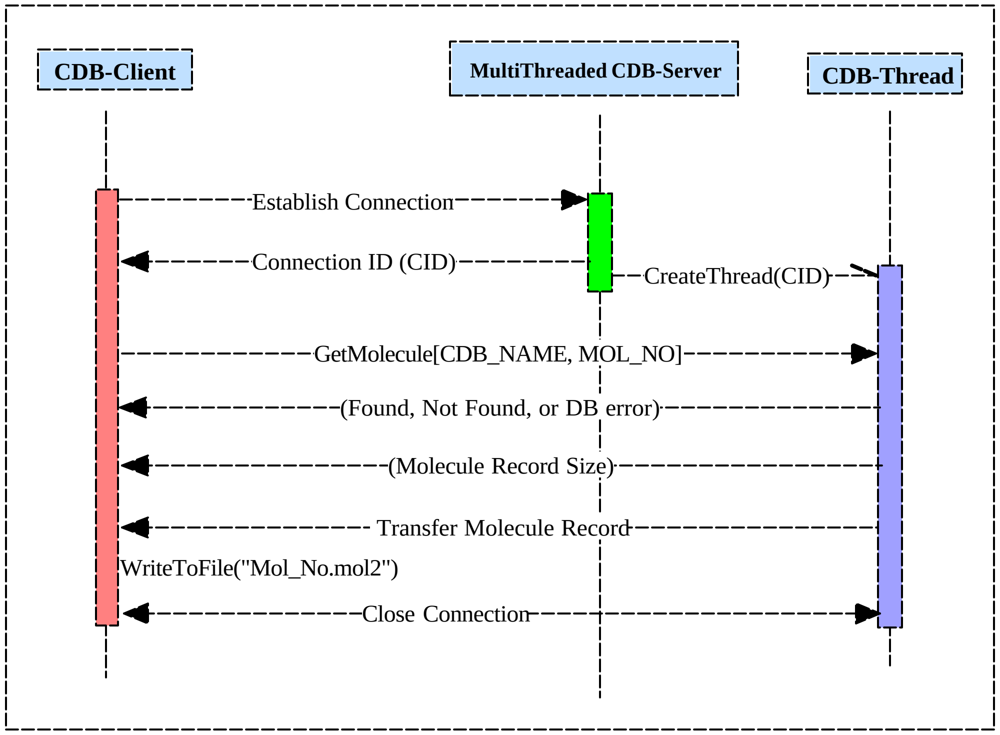


Figure 6: Protocols for Interaction between CDB clients and the server.

3 Application Composition

Nimrod/G experiments are configured from a plan file written using a simple declarative language. The plan file specifies the parametric tasks and the types of the parameters for these tasks. A parametric task consists of a script defined using a sequence of simp le commands, providing the ability to copy files to and from the remote node, execute certain programs, and perform parameter substitutions in input files.

Figure 7 shows an input file for the molecular docking program. This configures several aspects of the application, including the "ligand_atom_file" attribute, which indicates the sequence number of the molecule in the CDB to be docked. Nimrod/G will sweep over a range of molecules, so for each molecule it needs to place a different name at this position in the input file. To accomplish this, the user must replace the current value, naming a particular file, by a substitution placemarker. It consists of a dollar-sign ($) followed by the name of the parameter controlling the substitution, optionally surrounded by braces.

Figure 8 shows the input file with several attributes replaced by substitution place markers. The first part of the "ligand_atom_file" attribute has been replaced by a place marker for the parameter "ligand_number".

Figure 9 shows the parameter definition section of the plan file. Each parameter is defined by a keyword "parameter", followed by the parameter name, an optional label, and a parameter type. The remaining information on each line defines valid values for the parameter.

The parameter, for example, "database_name" has a label, and is of type *text*. Its valid values are listed, and the user will be able to select one of the values for the duration of the entire experiment. Most of the remaining parameters are single values, either text strings or integers, selected by the user, but with default values provided if the user does not wish to choose a value.

The range parameter, "ligand_number", used to select the molecule, is defined as an integer variable with bounds. For example, to process the first 2000 molecules in the CDB, this range parameter can vary from 1 to 2000 with the step size of 1.

The plan file is typically provided as input to a job generation tool, such as the EnFuzion Generator, in order to create a run file. The run file is similar to a plan file but contains specific instances of jobs to be run.

```
score_ligand                  yes
minimize_ligand               yes
multiple_ligands              no
random_seed                   7
anchor_search                 no
torsion_drive                 yes
clash_overlap                 0.5
conformation_cutoff_factor    3
torsion_minimize              yes
match_receptor_sites          no
random_search                 yes
    . . . . . .
    . . . . . .
maximum_cycles                1
ligand_atom_file              S_1.mol2
receptor_site_file            ece.sph
score_grid_prefix             ece
vdw_definition_file           parameter/vdw.defn
chemical_definition_file      parameter/chem.defn
chemical_score_file           parameter/chem_score.tbl
flex_definition_file          parameter/flex.defn
flex_drive_file               parameter/flex_drive.tbl
ligand_contact_file           dock_cnt.mol2
ligand_chemical_file          dock_chm.mol2
ligand_energy_file            dock_nrg.mol2
```

Molecule to be screened

Figure 7: A configuration input file for docking application.

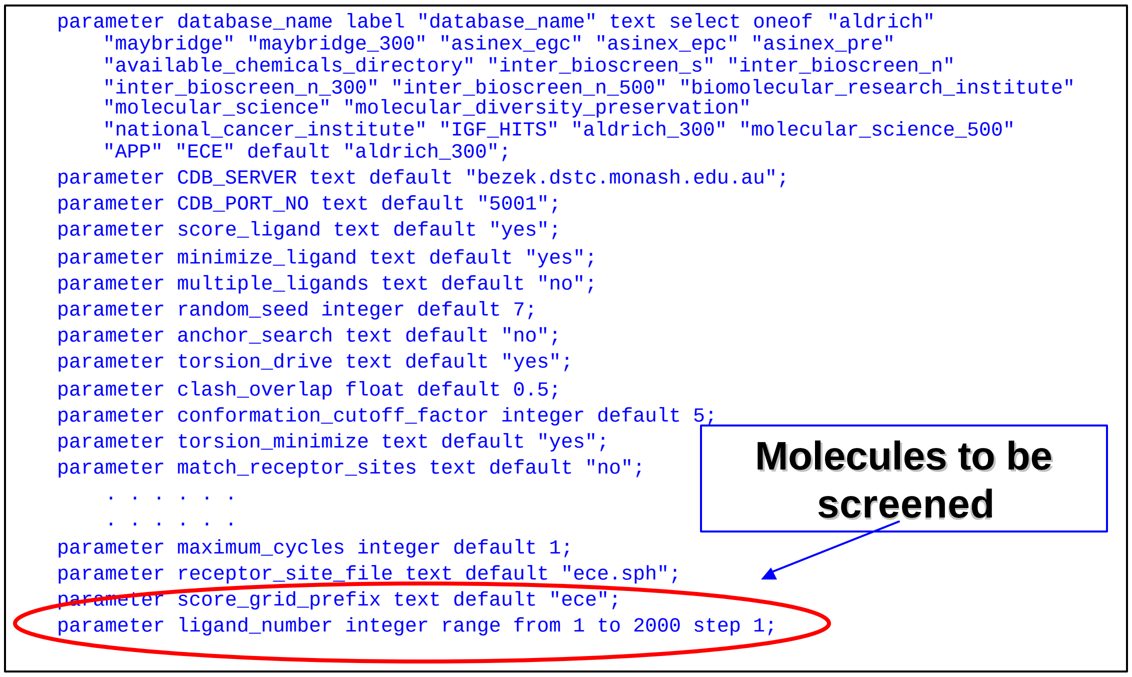
```
score_ligand                  $score_ligand
minimize_ligand               $minimize_ligand
multiple_ligands              $multiple_ligands
random_seed                   $random_seed
anchor_search                 $anchor_search
torsion_drive                 $torsion_drive
clash_overlap                 $clash_overlap
conformation_cutoff_factor    $conformation_cutoff_factor
torsion_minimize              $torsion_minimize
match_receptor_sites          $match_receptor_sites
random_search                 $random_search
    . . . . . .
    . . . . . .
maximum_cycles                $maximum_cycles
ligand_atom_file              ${ligand_number}.mol2
receptor_site_file            $HOME/dock_inputs/${receptor_site_file}
score_grid_prefix             $HOME/dock_inputs/${score_grid_prefix}
vdw_definition_file           vdw.defn
chemical_definition_file      chem.defn
chemical_score_file           chem_score.tbl
flex_definition_file          flex.defn
flex_drive_file               flex_drive.tbl
ligand_contact_file           dock_cnt.mol2
ligand_chemical_file          dock_chm.mol2
ligand_energy_file            dock_nrg.mol2
```

Molecule to be screened

Figure 8: Parameterisation of a configuration input file.

The parameters “receptor_site_file” and “score_grid_prefix” indicate the data input files. Their values indicate that data input files are located in the user home directory on Grid nodes. Instead of pre-staging, these files can be copied at runtime by defining necessary “copy” operation in the job’s “nodestart” or “main” task (see Figure 10). However, it is advisable to copy or “pre-stage” large input files in the beginning of application execution instead of copying them during execution of every job. This saves transmission time particularly when those files are going to be used for docking with many databases.

During the conversion, the user is able to set concrete values for each of the parameters. For the parameter "ligand_number", the user may choose not to select all values from 1 to 2000, but may select a subset of these values. By default, this generated 2000 jobs, each docking a single molecule.



```
    parameter database_name label "database_name" text select oneof "aldrich"
        "maybridge" "maybridge_300" "asinex_egc" "asinex_epc" "asinex_pre"
        "available_chemicals_directory" "inter_bioscreen_s" "inter_bioscreen_n"
        "inter_bioscreen_n_300" "inter_bioscreen_n_500" "biomolecular_research_institute"
        "molecular_science" "molecular_diversity_preservation"
        "national_cancer_institute" "IGF_HITS" "aldrich_300" "molecular_science_500"
        "APP" "ECE" default "aldrich_300";
    parameter CDB_SERVER text default "bezek.dstc.monash.edu.au";
    parameter CDB_PORT_NO text default "5001";
    parameter score_ligand text default "yes";
    parameter minimize_ligand text default "yes";
    parameter multiple_ligands text default "no";
    parameter random_seed integer default 7;
    parameter anchor_search text default "no";
    parameter torsion_drive text default "yes";
    parameter clash_overlap float default 0.5;
    parameter conformation_cutoff_factor integer default 5;
    parameter torsion_minimize text default "yes";
    parameter match_receptor_sites text default "no";
        . . . . . .
        . . . . . .
    parameter maximum_cycles integer default 1;
    parameter receptor_site_file text default "ece.sph";
    parameter score_grid_prefix text default "ece";
    parameter ligand_number integer range from 1 to 2000 step 1;
```

Figure 9: A plan file defining parameters type and their values.

The run file then contains a job for each combination of parameters. Hence the number of jobs is the product of the number of values chosen for each parameter. Since most of the parameters except "ligand_number" are single-valued, they have no effect on the number of jobs.

Figure 10 shows the task definition section of the plan file. The "nodestart" task is performed once for each remote node. Following that, the files copied during that stage are available to each job when it is started. The "main" task controls the actions performed for each job.

```
task nodestart
      copy ./parameter/vdw.defn node:.
      copy ./parameter/chem.defn node:.
      copy ./parameter/chem_score.tbl node:.
      copy ./parameter/flex.defn node:.
      copy ./parameter/flex_drive.tbl node:.
      copy ./dock_inputs/get_molecule node:.
      copy ./dock_inputs/dock_base node:.
endtask
task main
           node:substitute dock_base dock_run
           node:substitute get_molecule get_molecule_fetch
           node:execute sh ./get_molecule_fetch
           node:execute $HOME/bin/dock.$OS -i dock_run -o dock_out
           copy node:dock_out ./results/dock_out.$jobname
           copy node:dock_cnt.mol2 ./results/dock_cnt.mol2.$jobname
           copy node:dock_chm.mol2 ./results/dock_chm.mol2.$jobname
           copy node:dock_nrg.mol2 ./results/dock_nrg.mol2.$jobname
endtask
```

Figure 10: Task definition of docking jobs.

The first line of the "main" task performs parameter substitution on the file "dock_base", creating a file "dock_run". This is the action that replaces the substitution place markers in our input file with the actual values for the job.

As each docking operation is performed on a selected molecule in the CDB database, it is not necessary to copy such large databases on all Grid nodes. Hence, not only is the molecule file named in the configuration file, we also go to particular lengths to copy only the data for the molecule being tested. The executable script "get_molecule_fetch" (see Figure 11) is also created using parameter substitution, and runs the "vlab-cdb-get-molecule" executable, which fetches the molecule record from the *CDB molecule server* based on the parameter "ligand_number". The molecule record is saved in a file whose name is the same as integer value of the “ligand_number” parameter and “mol2” as its extension. For instance, if the parameter ligand_number value is 5, then molecule record will be saved in a file “5.mol2”.

```
#!/bin/sh
$HOME/bin/vlab-cdb-get-molecule.$OS $CDB_SERVER $CDB_PORT_NO ${database_name}.db $ligand_number
```

Figure 11: Parameterisation of script for extracting molecule from CDB.

The main code is the "dock" executable. Note that in the “execute” command, there are pseudo-parameters that do not appear in the plan file. These include environment variables, such as "HOME", as well as other useful parameters, such as "OS" indicating the operating system on the node. This allows us to select the correct executable for the node. If the “dock” executable files do not exist on Grid nodes, they need to be copied at runtime as part of the job’s “nodestart” task similar to copying input files.

The dock_run file created in the substitution step previously is now provided as the input configuration file for the docking process. The output files are then copied back to the local host, and renamed with another pseudo-parameter, the unique "jobname" parameter.

4 Scheduling Experimentation

We have performed scheduling experiments from a grid resource in Australia along with four resources available in Japan and one in USA. Table 1 shows the list of resources and their properties, Grid services, access cost or price in terms of Grid dollar (G$) per CPU-second, and the number of jobs processed on resources with deadline-and-budget constrained (DBC) time optimization (TimeOpt) and cost optimization (CostOpt) scheduling strategies.

Organization & Location	**Vendor, Resource Type, # CPU, OS, hostname**	**Grid Services and Fabric, Role**	**Price (G$/CPU sec.)**	**Number of Jobs Executed**	
				TimeOpt	**CostOpt**
Monash University, Melbourne, Australia	Sun: Ultra-1, 1 node, bezek.dstc.monash.edu.au	Globus, Nimrod-G, CDB Server, Fork (Master node)	--	--	--
AIST, Tokyo, Japan	Sun: Ultra-4, 4 nodes, Solaris, hpc420.hpcc.jp	Globus, GTS, Fork (Worker node)	1	44	102
AIST, Tokyo, Japan	Sun: Ultra-4, 4 nodes, Solaris, hpc420-1.hpcc.jp	Globus, GTS, Fork (Worker node)	2	41	41
AIST, Tokyo, Japan	Sun: Ultra-4, 4 nodes, Solaris, hpc420-2.hpcc.jp	Globus, GTS, Fork (Worker node)	1	42	39
AIST, Tokyo, Japan	Sun: Ultra-2, 2 nodes, Solaris, hpc220-2.hpcc.jp	Globus, GTS, Fork (Worker node)	3	11	4
Argonne National Lab, Chicago, USA	Sun: Ultra -8, 8 nodes, Solaris, pitcairn.mcs.anl.gov	Globus, GTS, Fork (Worker node)	1	62	14
		Total Experiment Cost (G$)		17702	14277
		Time to Finish Expt. (Min.)		34	59.30

Table 1: The WWG testbed resources used in scheduling experiments, job execution and costing.

We have performed a trial screening 200 molecules (from the *aldrich_300* CDB) on a target receptor called endothelin converting enzyme (ECE), which is involved in hypotension. The three dimensional structure of receptor is derived from homology modeling using related receptor structures whose three dimensional structures have been solved by X-ray crystallography experiments.

In this experimentation, for faster evaluation purpose, the range parameter "ligand_number" is defined with the bounds 1 and 200 and the step size as 1, which produces 200 jobs for docking molecules. As shown in Figure 12, *dock* program takes two different types of inputs files: a) *common input files*, the same files are required for all docking jobs and b) *ligand specific input files*, which vary from one job to another. The large common input files (receptor structure and pre-calculated score potentials) are pre-staged on resources instead of copying them at runtime. The files are copied using the *globus-rcp* command and stored in the directory location "$HOME/dock_inputs/" on resources as specified by the parameters "receptor_site_file" and "score_grid_prefix" (see Figure 8). The two application-specific executable files, "dock" and "vlab-cdb-get-molecule" invoked in the task scripts (see Figures 10 and 11) are also pre-staged. The executable files are stored in the "$HOME/bin/" directory on resources.

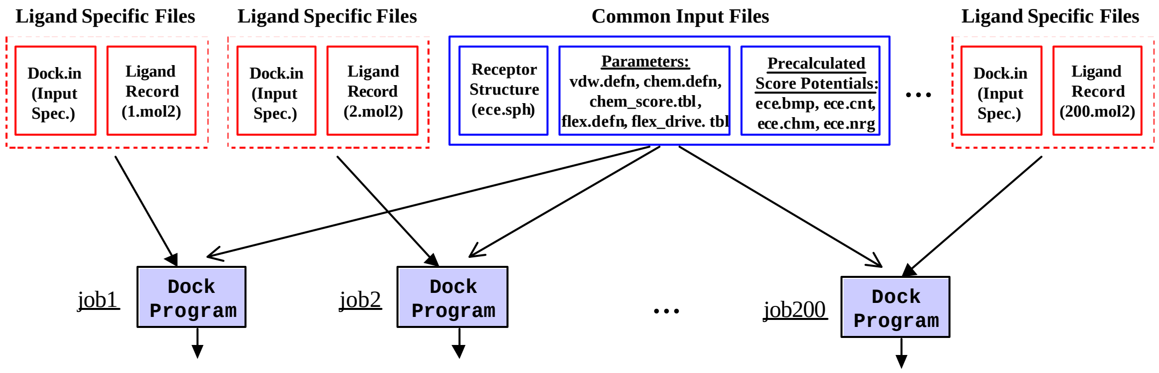
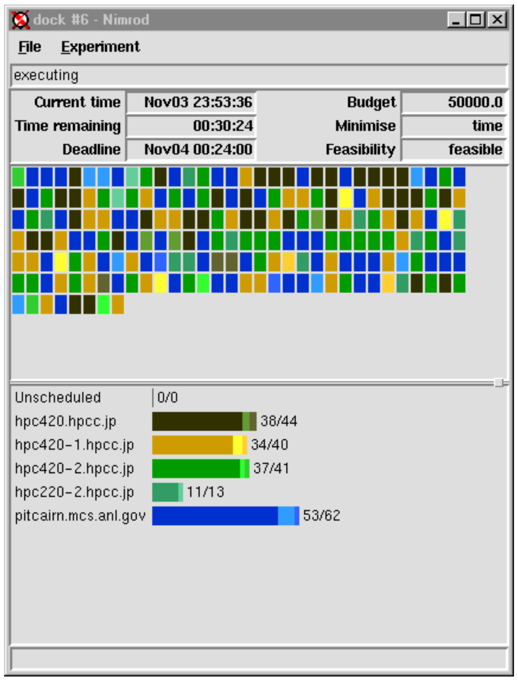


Figure 12: Static and Dynamic Input Files of Docking program.

In this scheduling experiment, we have set 60 minutes as the deadline limit and 50,000 G$ as the budget limit. We conducted experiments for two different optimization strategies [7]:

1. Optimize for Time - this strategy aims to produce results at the earliest possible time before a deadline, and within a budget limit.
2. Optimize for Cost - this strategy aims to minimize the cost of execution (spending from the given budget) and complete the experiment on or before the deadline.

The first experiment, *Optimize for Time* scheduling, was performed on November 3, 2001 at 23:23:00, Australian Eastern Standard Time (AEST), with a 60-minute deadline and finished on November 3, 2001 by 23:57:00. A snapshot of the Nimrod-G monitoring and steering client taken a few minutes (~5min.) before the completion of application processing is shown in Figure 13. This experiment took 34 minutes to finish the processing of all jobs using resources available at that time with an expense of 17,702 G$. Figure 15 shows the number of jobs processed on different resources selected depending on their cost and availability. Figure 16 shows the corresponding expenses of processing on resources. Figure 17 shows the number of jobs in execution on resources at different times. From the graphs it can be observed that the broker selected resources to ensure that the experiment was completed at the earliest possible time given the current availability of resources and the budget limitations. After 30 minutes, it discovered that it could still complete early without using the most expensive resource, hpc220-2.hpcc.jp.

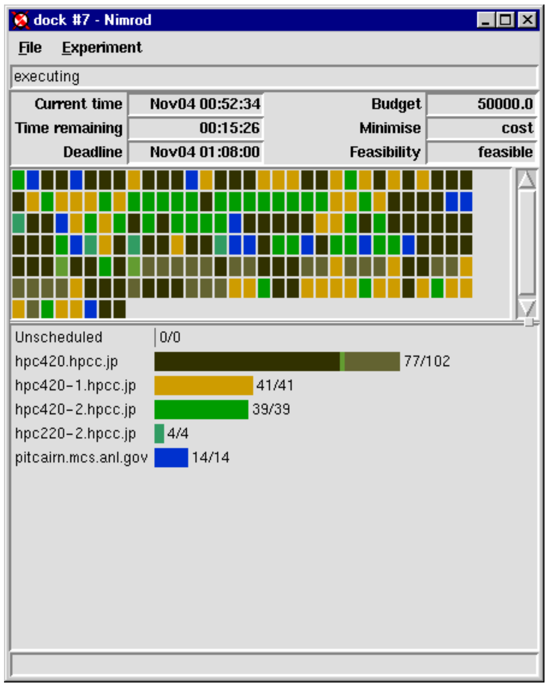

dock #6 - Nimrod

File Experiment

executing

Current time	Nov03 23:53:36	Budget	50000.0
Time remaining	00:30:24	Minimise	time
Deadline	Nov04 00:24:00	Feasibility	feasible

Unscheduled	0/0
hpc420.hpcc.jp	38/44
hpc420-1.hpcc.jp	34/40
hpc420-2.hpcc.jp	37/41
hpc220-2.hpcc.jp	11/13
pitcairn.mcs.anl.gov	53/62


dock #7 - Nimrod

File Experiment

executing

Current time	Nov04 00:52:34	Budget	50000.0
Time remaining	00:15:26	Minimise	cost
Deadline	Nov04 01:08:00	Feasibility	feasible

Unscheduled	0/0
hpc420.hpcc.jp	77/102
hpc420-1.hpcc.jp	41/41
hpc420-2.hpcc.jp	39/39
hpc220-2.hpcc.jp	4/4
pitcairn.mcs.anl.gov	14/14

Figure 13: A Snapshot of the Nimrod-G Monitor during "Optimize for Time" Scheduling Experiment.

Figure 14: A Snapshot of the Nimrod-G Monitor during "Optimize for Cost" Scheduling Experiment.

It should be noted that for each job scheduled for execution on Grid, the Nimrod-G runtime machinery (actuator) deploys Nimrod-G agents on remote resources. The Nimrod agents setup runtime environments (generally in scratch file space, "/tmp") on remote resources and execute commands specified in the task definition script (see Figure 10). The docking parameter files and ligand specific files are transferred from the home node, bezek.dstc.monash.edu.au in this case. The agent uses *http* protocols to fetch files via the http-based file server running on the home node. All parameter variables in the parameterized input files (see Figures 8 and 11) are substituted by their concrete values before processing. The ligand record is fetched from the CDB database server running on the home node. The agent then executes the *dock* program and stores output files in the scratch area. The required output files are then transferred to the home node and stored with the job number as their extension. All these steps involved in the execution of the *dock* program on Grid resources were completely hidden from the user.

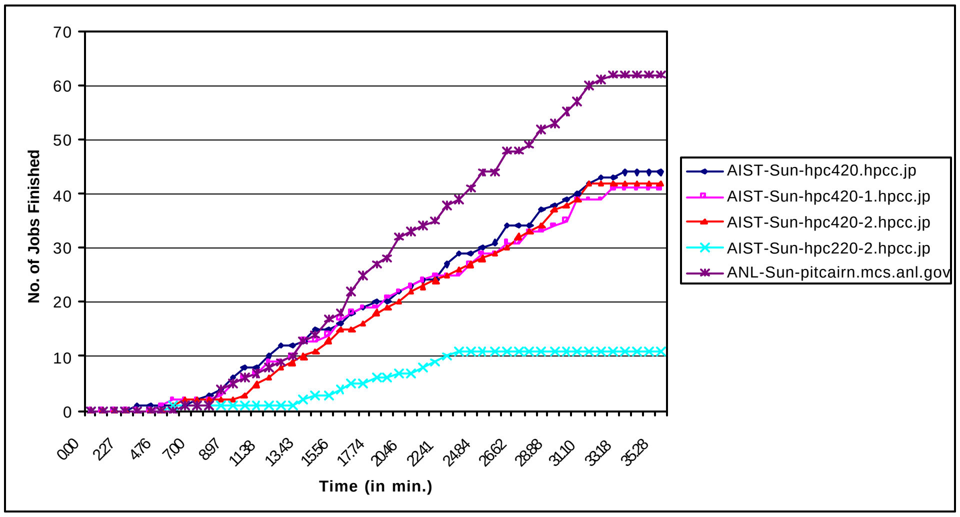


Figure 15: No. of jobs processed on Grid resources during DBC time optimization scheduling.

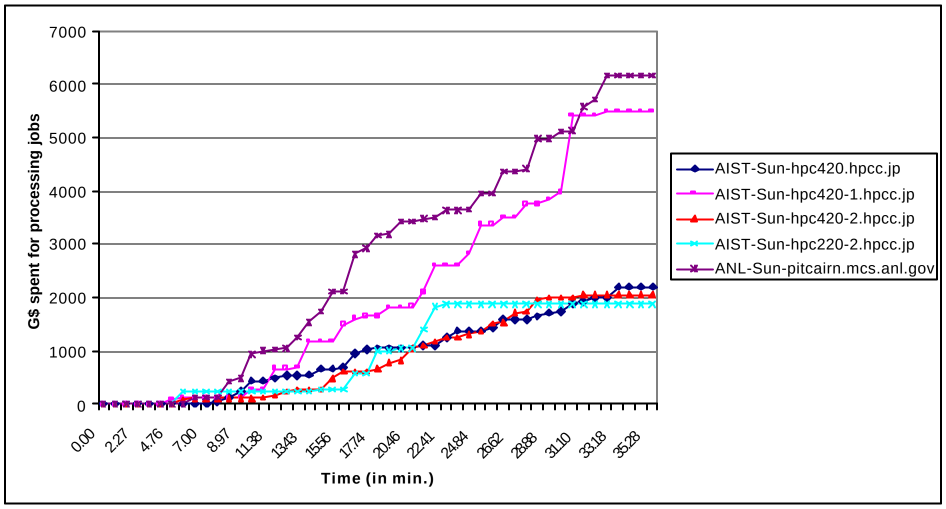


Figure 16: The amount spent on resources during DBC time optimization scheduling.

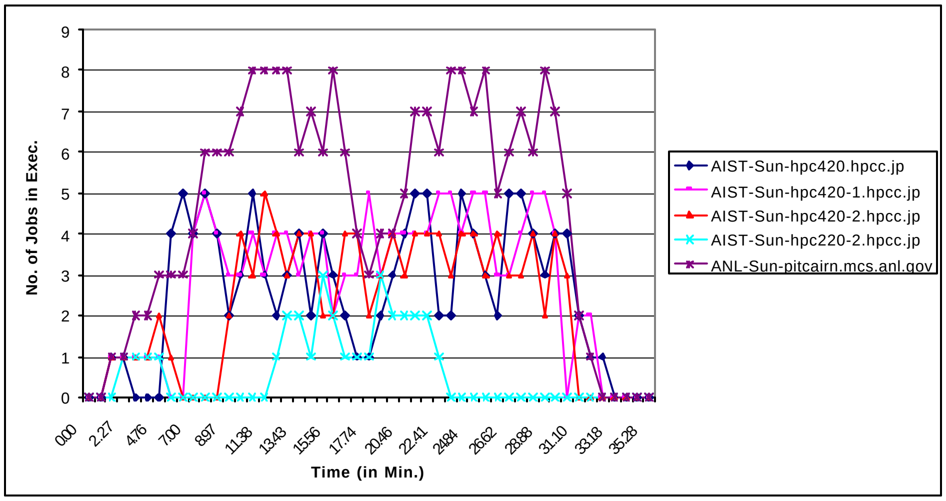


Figure 17: No. of jobs in execution on Grid resources during DBC time optimization scheduling.

The second experiment, *Optimize for Cost* scheduling, was performed on November 4, 2001 at 00:08:00, AEST, with a 60-minute deadline and finished on November 4, 2001 by 01:07:30. A snapshot of the Nimrod-G monitoring and steering client taken few minutes (~5min.) before the completion of application processing is shown in Figure 14. This experiment took almost 59.30 minutes to finish the processing of all using resources available at that time with an expense of 14,277 G$. It is interesting to note that the second experiment took extra 25.40 minutes, but saved 3,475 G$ in the process. Figure 18 shows the number of jobs processed on different resources selected depending on their cost and availability. Figure 19 shows the corresponding expenses of processing on resources. Figure 20 shows the number of jobs in execution on resources at different times. From the graphs it can be observed that the broker selected cheapest resources to ensure that the experiment was completed with minimum expenses, but before the deadline limit. In the beginning expensive resources are used to ensure that the deadline can be meet. If for any reason cheapest resources are unable to deliver expected performance, then the broker seeks the help of expensive resources to meet the deadline.

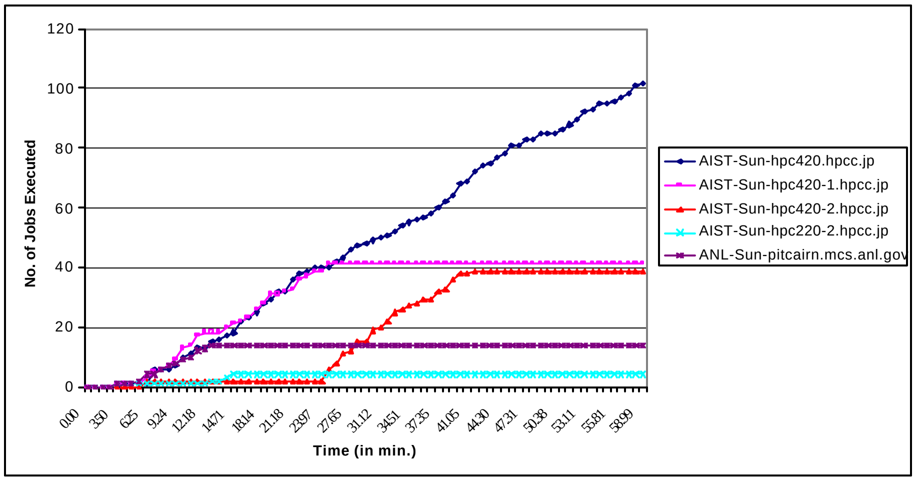


Figure 18: No. of jobs processed on Grid resources during DBC Cost optimization scheduling.

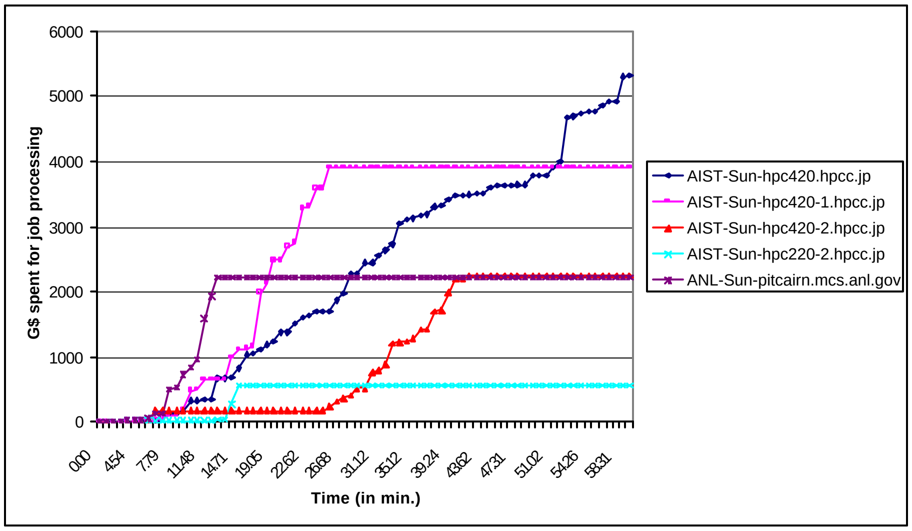


Figure 19: The amount spent on resources during DBC Cost optimization scheduling.

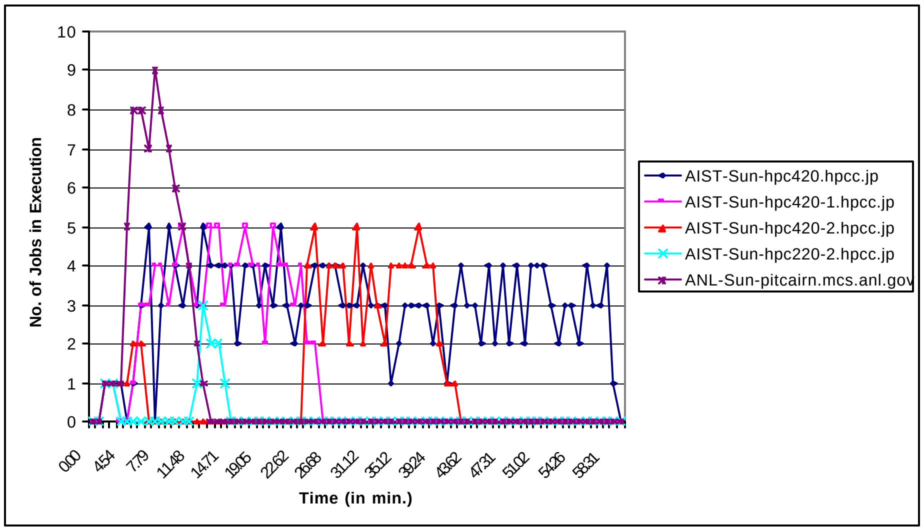


Figure 20: No. of jobs in execution on Grid resources during DBC Cost optimization scheduling.

5 Summary and Conclusion

Computational Grids enable the *sharing* and *aggregation* of geographically distributed resources for solving large-scale, resource and data-intensive problems faster and cheaper. However, application development, resource management, and scheduling in these environments is a complex undertaking. We have developed Virtual Laboratory environment and tools for formulating molecular docking for drug design as a parameter sweep application, chemical database management, and scheduling docking jobs for processing on a wide area distributed resources by leveraging existing Grid technologies. The new tools developed include chemical database indexer, CDB server for providing access to molecules in chemical databases as a network service, clients for accessing CDB services from a selected CDB service. We have used the Nimrod-G parameter specification language for composing an existing docking application as a parameter sweep application and the Nimrod-G grid resource broker for processing molecular docking jobs on distributed resources.

We have conducted deadline-and-budget-constrained scheduling experiments for parallel processing of docking jobs on the worldwide grid testbed under two different optimization scenarios. The results of this molecular docking application scheduling on a large-scale distributed resources show the potentials of the Virtual Laboratory tools for service oriented computing. They prove the effectiveness of computational economy and quality of services (QoS) driven scheduling as an efficient mechanism for the management of supply-and-demand for resources depending on the value delivered to the user. The economy driven service oriented computing encourages the users to utilize resources effectively by trading off between the deadline and budget depending on their QoS requirements.

Our experience with developing a prototype Virtual Laboratory environment for distributed drug design shows the potential and applicability of the Nimrod-G tools for data intensive computing. We are extending the current system to support adaptive mechanisms for the selection of the best CDB service depending on access speed and cost. We are also looking into applying the experience gained in this work to develop virtual laboratory environment for enabling high-energy physics events processing on distributed resources on a larger scale.

Acknowledgements

We would like to thank John Crossley (Monash U.) and Rob Gray (DSTC) for proof reading earlier drafts. They offered constructive comments and suggestions for enhancing the quality of the paper. We appreciate the efforts of Heinz Stockinger (CERN) for reviewing the chemical database server implementation. We thank National Institute of Advanced Industrial Science and Technology, Tokyo, Japan and the Argonne National Laboratory, Chicago, USA for providing access to the resources used in the experimentation.

References

[1] I. Foster and C. Kesselman (editors), *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.

[2] R. Buyya (editor), *High Performance Cluster Computing: Architectures and Systems*, Volumes 1 and 2, Prentice Hall, NJ, USA, 1999.

[3] D. Abramson, R. Sosic, J. Giddy, and B. Hall, *Nimrod: A Tool for Performing Parameterized Simulations using Distributed Workstations*, The 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.

[4] D. Abramson, J. Giddy, and L. Kotler, *High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?* International Parallel and Distributed Processing Symposium (IPDPS), IEEE Computer Society Press, 2000.

[5] R. Buyya, D. Abramson and J. Giddy, *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*, 4th Intl. Conf. on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), China.

[6] R. Buyya, D. Abramson and J. Giddy, *Economy Driven Resource Management Architecture for Computational Power Grids*, Intl. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), USA.

[7] R. Buyya, J. Giddy, D. Abramson, *An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications*, The Second Workshop on Active Middleware Services (AMS 2000), In conjunction with HPDC 2001, August 1, 2000, Pittsburgh, USA (Kluwer Academic Press).

[8] R. Buyya, H. Stockinger, J. Giddy, and D. Abramson, *Economic Models for Management of Resources in Peer-to-Peer and Grid Computing*, SPIE International Conference on Commercial Applications for High-Performance Computing, August 20-24, 2001, Denver, USA.

[9] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications*, 11(2): 115-128, 1997.

[10] T. Ewing (editor), *DOCK Version 4.0 Reference Manual*, University of California at San Francisco (UCSF), USA, 1998. Online version: http://www.cmpharm.ucsf.edu/kuntz/dock.html

[11] E. Lunney, *Computing in Drug Discovery: The Design Phase*, IEEE Computing in Science and Engineering Magazine, http://computer.org/cise/homepage/2001/05Ind/05ind.htm

[12] Tripos, Inc., *SYBYL Mol2 File Format*, http://www.tripos.com/services/mol2/, USA, Oct. 7, 2001.

[13] Tripos, Inc., *SYBYL Toolkit for molecular design and analysis*, http://www.tripos.com/software/sybyl.html, USA, Oct. 7, 2001.

[14] I. Kuntz, J. Blaney, S. Oatley, R. Langridge, and T. Ferrin, *A geometric approach to macromolecule-ligand interactions*, Journal of Molecular Biology, 161: 269-288, 1982.

[15] B. Shoichet, D. Bodian, and I. Kuntz, *Molecular docking using shape descriptors*, Journal of Compational Chemistry, 13(3): 380-397, 1992.

[16] E. Meng, D. Gschwend, J. Blaney, and I. Kuntz, *Orientational sampling and rigid-body minimization in molecular docking*, Proteins, 17(3): 266-278, 1993.

[17] TurboLinux, *EnFuzion Manual*, http://www.turbolinux.com/downloads/enf/man/enfuzion.htm

[18] R. Buyya, *World Wide Grid (WWG)*, http://www.csse.monash.edu.au/~rajkumar/ecogrid/wwg/

[19] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky, SETI@home: Massively Distributed Computing for SETI,

[20] W. Sullivan, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, D. Anderson, *A new major SETI project based on Project Serendip data and 100,000 personal computers*, Astronomical and Biochemical Origins and the Search for Life in the Universe—Proceedings of the Fifth International Conference on Bioastronomy. 1997.