Understanding and mastering dynamics in computing grids: processing moldable tasks with user-level overlay

Mościcki, J.T.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
User-level Overlay in action

In this Chapter\textsuperscript{1} we present examples of how the User-level Overlay may be used to efficiently support scientific user communities. Supporting multidisciplinary applications in distributed computing infrastructures is a collective effort in itself, as it brings together users, domain and computing experts and IT engineers. Such teamwork requires assuming multiple scientific and engineering roles: brainstorming of ideas, analysis and refactoring of applications, setting up and running experiments, delivering software components and guidance on how to use the existing ones. These various roles were assumed by the author of this thesis in the activities described in this Chapter.

5.1 Monte Carlo simulation with Geant4 toolkit

Geant4 [13] is a toolkit for the simulation of passage of particles through matter. Its areas of application include high energy, nuclear and accelerator physics, as well as studies in medical and space science. Geant4 uses Monte Carlo simulation model, which consists of a repetitive simulation of a large number of events. An event simulation consists of tracing of the trajectories of elementary particles as they are passing through matter and calculating their energies, momenta, energy depositions, decays and interactions. For example, an event may correspond to a collision of particle beams in the accelerator or the interaction of radioactive emissions with the tumors in oncological treatment. Task decomposition is straightforward for Monte Carlo simulations: tasks are independent and one task may correspond to any number of simulated events. Several approaches of task decomposition have been developed. They include static distribution of events to all processors using early binding [122] and redundant distribution of tasks using “N out of M strategy” where a subset of N tasks out of M submitted ones is needed to achieve the complete result [127]. Both approaches are problematic from a point of view of efficacy (the performance delivered to the application) and efficiency (the amount of wasted resources) [42]. Another approach to parallelizing Monte Carlo simulations is based on spatial parallelism where disjoint spatial domains are simulated simultaneously. Dynamic mapping of tasks to spatial domains was proposed in [70, 157]. These MPI-based approaches use semaphores with a distributed memory architecture and are not suitable for large-scale grids due to large latency between worker nodes and connectivity requirements which are not currently supported in grids.

In this section we present two examples of running Geant4-based simulations in distributed environments with task decomposition at the event level. The first one is a regression testing application which is a part of the Geant4 release process. In this case predictable and sustained delivery of partial results is particularly important for users. The second one is a group of medical physics and space science applications with varying computational requirements. We use this example to demonstrate two different ways of interfacing a Monte Carlo application to reduce runtime overheads. This example also illustrates the fact that scientists are often using mixed resources: local clusters, batch farms and the Grid.

5.1.1 Geant4 regression testing for release validation

Geant4 is a complex, object-oriented software package with more then $6 \times 10^5$ SLOC lines of optimized C++ code. Geant4 core software integrates a large number of modules which simulate different physics processes and which are contributed by members of a geographically distributed team. Therefore, careful testing of the Geant4 components is essential before the toolkit may be publicly released.

Statistical regression testing is performed before public releases which follow a 6-month release cycle. Previous public release is compared with a new release candidate using a test which consists of a simulation of a beam of elementary particles colliding

---

2Source Lines of Source Code - a metric measuring the size of a software program (generated using David A. Wheeler’s SLOCCount utility).
with a simplified model of a Tile Calorimeter. Several different types of particles are used, \( \pi^+ , \pi^+ , k^+ , k^- , p , n , e^- \), to cover electromagnetic and hadronic physics processes. The particle beams are configured with different energies from 1 GeV to 300 GeV. Several different types of materials (Fe-Sci, Cu-Sci,Cu-Lar, W-Lar, Pb-Sci, Pb-Lar, PbWO4) are used for the calorimeter setup to reproduce, in a simplified way, materials used in all LHC calorimeters. Finally, nine physics lists are tested. A physics list is a C++ library which provides models for interactions of elementary particles, and may provide more or less accurate approximation of reality depending on the particular simulation and energy range.

Geant4 regression testing for a single candidate release consists of \( \mathcal{O}(10^3) \) independent tasks, where a single task corresponds to a particular combination of particle type, energy, material type and physics list. Execution of a single task involves simulation of 5000 events. The simulation time of a single event depends on the energy and ranges from 0.03 s for 1 GeV to up to 10 s for 300 GeV on a standard 2008 PC. The total amount of CPU required for the testing of a single candidate release corresponds to few CPU years.

For management purposes and to easily re-run certain groups of tasks, the task space is subdivided in \( \mathcal{O}(35) \) runs, each consisting of approx. 250 tasks for the same physics list. The output of each task includes small test summary text file and small ROOT histogram files. Statistical testing is based on Kolmogorov tests to automatically compare the large number of physics observables. Statistically different distributions are visually examined by the testing team to understand the problem.

GANGA/DIANE framework was used for Geant4 regression testing since June 2007.

Task completion rate

Geant4 regression testing takes place in few weeks preceding a major public release. There is often a quick succession of candidate releases in the testing period and the testing may be partially or fully repeated multiple times. Therefore, during the testing period, a predictable and sustained processing throughput is essential for planning. Fig. 5.1 shows a comparison of task completion rate for the statistical regression testing application between late and early binding scheduling modes. The test run consisted of 207 independent tasks with average duration of around 400 seconds. In the early-binding scheduling mode (B) 207 jobs were submitted, each executing one simulation task. In the late-binding scheduling mode with DIANE (A), tasks were pulled by worker agents for processing. At least 170 EGEE Grid worker nodes were available for this test in the Geant4 VO. Therefore, the number of submitted worker agent jobs in the late-binding scheduling mode was fixed at 85 what corresponds to half of the number of all available worker nodes. In this way in both scheduling modes the same amount of grid resources was guaranteed.

Both scheduling modes were exercised simultaneously by submitting jobs to the EGEE Grid at the same time: 207 executable jobs in the early-binding mode (B1) and 85 worker agents in the late-binding mode (A1). Irregular job completion rate in the early-binding mode (B1 and B3) results from short-term variations of the grid dynamics. Moreover, job execution may be arbitrarily detailed what leads to incomplete runs
Figure 5.1: Comparison of task completion rate of late-binding scheduling based on DIANE (A) and early-binding scheduling (B). The figure shows three selected runs with typical behavior, for 85 submitted jobs. A number of effective workers in A is lower than the number of submitted jobs and is indicated in the figure. On the other hand, late-binding scheduling shields the user from such effects and provides a sustained task completion rate in the majority of cases (in our case the average number of tasks per worker agent was 2.5). It is worth mentioning, that the number of effectively used worker agents (indicated in Fig. 5.1) is typically smaller than the number of submitted ones: not all worker agent jobs start executing in time. However, the late-binding scheduler assures that, even if the number of effectively available resources is low and varying, the application output is produced in a steady and stable way.

5.1.2 Medical and space science studies

Geant4 toolkit is frequently used in the context of medical and space sciences. The GANGA/Diane user overlay has been used by the researchers since 2002 for such applications as dosimetric studies for brachytherapy [62], hadrontherapy and medical linear accelerator [37], radiation-protection of silicon detectors for aviation and space missions [84] and planetary astrophysics [124]. Depending on the application area, responsiveness and scalability requirements for these distributed simulations may vary significantly.

Medical applications, such as brachytherapy (a radiotherapy technique which consists of inserting several sealed radioactive sources directly inside or in close contact with the tumor), require quick response times - comparable with current standards in the
clinical practice. Usage of the Monte Carlo method is advantageous because of increased accuracy of the simulation, which is based on detailed geometrical and material models of the human body. However Monte Carlo simulation is much more CPU-intensive than standard approaches used in commercial treatment planning software in many clinics. For Monte Carlo simulation to be used as part of the treatment planning process, the acceptable response times should not exceed a few minutes as the simulation results should be available in a time window which covers the patient’s visit in a medical facility. Therefore, at least several tens of computing nodes are needed to provide enough computing power for a realistic brachytherapy simulation using Monte Carlo techniques.

Astrophysics applications are at the other end of the spectrum. For example, high-precision Monte Carlo simulations for LISA, a joint ESA-NASA experiment in space for measuring the gravitational waves, require few CPU-years in fully batch mode [90]. The critical issues in this case are reliable error recovery (to preserve the completed parts of the simulation), monitoring of the progress of the jobs and traceability of the failed worker tasks for debugging purposes. Typically, an order of 1000 CPUs is required to achieve acceptable execution times.

Structure of a typical application

The problem of running Geant4 simulations in distributed computing environments was addressed in a general way the GANGA/Diane User-level Overlay. This approach is suitable for Geant4 applications which produce analysis objects, such as histograms or tuples [37]. To achieve this a Diane application plugin was developed, which uses the AIDA [152] compliant analysis system and which may be interfaced with a Geant4 simulation application in two ways: via an executable or via a high-level API.

Running the simulation via an executable is simple: it involves spawning a subprocess, passing arguments and checking the exit status. It is also non-intrusive at the level of application source code. However, it requires that the simulation is reinitialized for every task what may incur additional overheads.

The high-level API allows to initialize the simulation once and then simulate an arbitrary number of events on-the-fly. This is possible if the application is built and loaded as a shared library directly into the running WorkerAgent process. The library defines an entry point – a factory function `createG4Simulation()` to create a simulation object which implements `IG4Simulation` interface shown in Fig. 5.2. As the simulation object is loaded and initialized once, the simulation of subsequent events may be done without this overhead. The interface between C/C++ libraries and Python modules is automated by the SWIG [23] wrapper generator.

This solution requires some changes to the simulation code – typically a simple refactoring of the `main()` function. The refactoring task is easy because the `main()` function of a typical Geant4 application is a small, very-high-level driver which performs three steps: it instructs the Geant4 kernel to load the user configuration actions via a macro file, sets the initial random seed and issues a simulation command. Fig. 5.3 shows the actual code of `main()` for brachytherapy application. Refactored application code may be compiled to an executable in a usual way.

One important requirement is the reproducibility of the simulation which is fully
User-level Overlay in action

```cpp
/// Application class prototype which implements the high-level API used
/// by the WorkerAgent process.

class BrachySimulation : virtual public DIANE::IG4Simulation
{
public:
    BrachySimulation(G4int);
    ~BrachySimulation();
    void setSeed(G4int seed);
    G4bool initialize(int argc, char** argv);
    void executeMacro(std::string macroFileName);
    std::string getOutputFilename();
    void finish();

private:
    G4int seed;
    G4RunManager* pRunManager;
};
```

Figure 5.2: High-level abstract interface of a Geant-4 simulation and an entry-point for dynamic component loading.

```cpp
int main(int argc, char **argv)
{
    // command-line arguments
    G4String macrofile = argv[1];
    G4int seed = atoi(argv[2]);
    BrachySimulation *simulation = new BrachySimulation(0);
    simulation -> initialize(argc, argv);
    simulation -> setSeed(seed);
    simulation -> executeMacro(macrofile);
    simulation -> finish();
    delete simulation;
}
```

Figure 5.3: Actual code of the main() for brachytherapy application refactored for use as DIANE plugin.
deterministic and depends on the initial seed for the pseudorandom number generator. To achieve this for multiple parallel tasks, an array of seeds, one seed per task, is first created from the master seed. Thus each task is executed with its own, predefined random seed. For larger number of tasks this approach requires a pseudo random generator with a long period, such as the Mersenne Twister [128]. From the point of view of a user there is a single master seed parameter to be handled in the run configuration.

Tasks produce analysis objects corresponding to their portion of simulation, which are stored in files in AIDA XML format. Analysis objects contain physics observables, such as spatial distributions of energy delivered by the radioactive source around the tumor in the case of brachytherapy. The final analysis objects are assembled by the RunMaster and are made immediately available to the user, alongside with the partial analysis objects. User may inspect the “live” analysis objects which are generated on the fly during the run.

**Results**

Several runs of Geant4 medical simulations in EGEE Grid were performed and are described in [37]. Some 50 brachytherapy simulation runs were performed over 3 weeks, using a pool of 40 worker nodes for each run. This is a relatively fast simulation, where each run consisted of $10^3$ tasks, simulating $10^7$ events each. Equivalent run with a single 2007 PC (sequential simulation) was estimated at $417 \pm 8$ minutes. Small error measure for the sequential simulation confirms that, due to a simple geometry setup, the CPU usage is very stable and not sensitive to the initial random seed. Thus it is reasonable to assume that the distribution of task duration, shown in Fig. 5.4, indicates the tail intrinsic to the distribution of processor speeds in the EGEE Grid.

Assuming ideal, linear speedup, with 40 identical CPUs, the simulation makespan should be approximately 10 minutes. Despite much wider distribution (due to additional job submission overhead) shown in Fig. 5.5, we may conclude that in relatively fast use cases, like brachytherapy simulation, calculation times close to the requirements in clinical practice may be achieved [37]. However, we also note that much larger use cases, such as medical linac, would require several days of computing with resources available to the Geant4 community at the time. Thus, addition of more resources in the Geant4 VO would be required for such use case to be supported in a production environment.

For smaller Monte Carlo studies users sometimes exploit local resources which are available for processing without job submission overhead. For example, a simulation of X-ray fluorescence emissions of rocks for the design study of BepiColombo ESA mission to Mercury [124] was performed in a cluster where a user had interactive access. The Remote backend of GANGA allowed to run WorkerAgents using a direct ssh connection to selected worker nodes. It is worth mentioning that from the user’s perspective the operation of the User-level Overlay remain identical, independently of the type resources used by the backend (Grid, batch or ssh).

The tests involved a small, interactive cluster of 15 worker nodes (with 20 nodes used for the largest simulation). The cluster was shared with other users. As shown in Fig. 5.6 DIANE was able to achieve between 75-95% of theoretical efficiency, defined as
Figure 5.4: Histogram of task duration for brachytherapy simulation on EGEE Grid.

Figure 5.5: Histogram of overall simulation time (makespan) for brachytherapy simulation on EGEE Grid.
where, $t_s$ is the elapsed simulation time on a processor exclusively used by a single user and $t_c$ is the elapsed simulation time on a processor shared simultaneously by many users. Expected performance of a parallel run is proportional to the number of available CPUs and inversely proportional to the execution time as shown in Fig. 5.7. The performance gain in this case could be even greater if multi-core nodes were exploited fully by allocating one WorkerAgent per processor (core) rather than one WorkerAgent per node.

### 5.2 Workflows for medical imaging simulations

The OpenGATE collaboration is a research group aiming at developing simulations for PET imaging [100]. The virtual laboratory developed in [147] provides grid-based services to support large-scale data storage, analysis and collaboration in medical imaging studies.

#### 5.2.1 Virtual laboratory with Ganga/DIANE components

The virtual laboratory environment, shown in Fig. 5.2.1, consists of user tools and the service backbone. Graphical user tools allow a user to easily manage data stored on the grid and help with preparation and launching of distributed GATE simulations. The service backbone integrates the **GANGA/DIANE** User-level Overlay and is used to execute job workflows in grid environments. The MOTEUR [76] workflow execution
engine provides a steering logic and GANPA/DIANE overlay serves as a task processing engine.

The workflow engine combined with GANPA/DIANE overlay allows dynamic partitioning of the simulations in such a way that each WorkerAgent runs a simulation program independently with a different initial random seed and periodically updates the number of simulated events to the RunMaster. The run terminates when a desired number of simulated events is reached. The output of the simulation is uploaded periodically to the output storage.

The MOTEUR engine is interfaced to the RunMaster as an external workflow engine which keeps track of the number of simulated events and generates new tasks if needed. The workflow engine effectively implements a control loop: tasks results are analyzed on-the-fly and the run is terminated when the simulation converges with a requested precision.

The DIANE RunMaster and the MOTEUR engine use a simple, file-based communication scheme implemented above a local file-system to keep track of the task status and exchange task input and output files. A special ApplicationManager plugin was developed to implement this scheme in a generic way and may be applied to any application managed by MOTEUR.
5.2.2 Results

The virtual laboratory service is operated at the Creatis research facility in Lyon\(^3\). From July 2009 to August 2010 more than 360 DIANE RunMaster instances were activated in the service backbone which handled \(58 \times 10^3\) worker agent jobs completing more than \(113 \times 10^3\) simulation tasks.

In addition, several targeted experiments were performed to assess the dynamic, workflow-based partitioning and simulation steering using GAN\(G\)A/DIANE overlay. Static splitting based on early binding was compared with dynamic splitting based on late binding [154]. Several scenarios were investigated: gLite-based file storage and transfer was compared with local file storage with DIANE FileTransferService. After processing several thousand tasks it was concluded that using late binding enables significantly more reliable operation than early binding as it allowed 100% of the results to be achieved in all tests. This conclusion turned out to be particularly important for grid-based output storage which was a source of a large fraction of runtime errors.

The late-binding approach also allowed a significant reduction of makespan. Additional overhead of the workflow engine was not penalizing in the terms of achieved performance. As an example, a simulation of \(20 \times 10^6\) events took 8.5 hours of a standard, 2008 dual-core PC. In the EGEE Grid with 100 jobs, the early-binding submission mode required up to 24 hours for all jobs to finish but yielded only 78% of successfully

---

\(^{3}\)Creatis is a part of Centre National de la Recherche Scientifique, France.
retrieved simulation results. On the other hand the GAN/GA/Diane overlay with 100 worker agents required 1.75 hour to complete all simulation tasks simulation, yielding 100% of successfully retrieved results despite 22% failure rate of the worker agent jobs.

This solution, developed in the context of the virtual laboratory project, integrates Diane/GAN/GA overlay into a workflow-enabled computing service above EGEE Grid. It allows to achieve consistently better performance than direct job submission based on early binding. At the same time, it provides a generic service for other workflow-based applications and is a demonstration of the flexibility and efficiency of the User-level Overlay approach.

5.3 Data processing for ATLAS and LHCb experiments

The ATLAS and LHCb experiments aim to make discoveries about the fundamental nature of the Universe by detecting new particles at high energies, and by performing high-precision measurements of particle decays. The experiments are located at the Large Hadron Collider (LHC) at the European Laboratory for Particle Physics (CERN), Geneva, with first particle collisions (events) observed in 2009. The LHCb experiment is dedicated to studying the properties of B mesons (particles containing the b quark), while ATLAS is a general-purpose experiment, designed to allow observation of new phenomena in high-energy proton-proton collisions.

Both experiments require processing of data volumes of the order of petabytes per year, rely on computing resources distributed across multiple locations, and exploit several Grid implementations, including EGEE, OSG and NDGF as well as locally available batch farms. The data from the experiments is distributed at computing facilities around the world and processed according to computing models specific to each experiment [57, 20]. The data-processing applications, including simulation, reconstruction and final analysis for the experiments, are based on the C++ Gaudi/Athena [22] framework. This provides core services, such as message logging, data access, histogramming, and a run-time configuration system.

An outstanding issue for LHC experiments is efficient management and access to very large volumes of data, often on-demand to allow rapid pre-filtering of data based on certain selection criteria so as to identify data of specific interest. This is implemented using VO-specific late-binding overlays which are coupled to distributed data management systems. In ATLAS, the usage of PANDA system [120] has been increasingly gaining importance as a analysis and production coordination system used in conjunction with the DQ2 [30]. In LHCb Grid jobs are routed through the DIRAC [176] workload management system (DIRAC WMS). In both experiments GAN/GA is used as the primary user interface and workload management is implemented by experiment-specific systems. Usage of GAN/GA in conjunction with Diane was reported in ATLAS for analysis clusters [68].

Preparation of application runtime environment for different distributed infrastructures in use in the LHC experiments is a challenging task. Application configuration typically involves several steps and complex preprocessing. Moreover, it differs between
distributed infrastructures. Therefore, applications components defined in GANDA play a key role in improving productivity of the physicist end users. At the same time multiple each distributed infrastructure defines a different access interface through middleware such as gLite or ARC, APIs such as DIRAC-API [150] or protocols such as HTTP (used in PANDA). Therefore, the important role of GANDA as application configuration and resource access interface.

One common use-case is easy switching between processing systems. Code under development by a user may contain bugs that cause runtime errors during job execution. The transparent switching between processing systems when using GANDA means that debugging can be performed locally, with quick response time, before launching a large-scale analysis on the Grid, where response times tend to be longer.

Results reported in [136] show that in 2008, more than $4 \times 10^5$ Grid jobs in ATLAS, and more than $3 \times 10^5$ Grid jobs in LHCb were submitted by end users with GANDA. Since 2007 1930 users in ATLAS and 630 users in LHCb were recorded. Fig. 5.9 shows the distribution of ATLAS and LHCb users in time.

Additionally, end-to-end testing of the distributed analysis models of the experiments is performed using Robots implemented above GANDA interface. Robots submit a representative set of analysis jobs on a daily basis, monitor their progress, and check the results produced. The overall success rate and the time to obtain the results is recorded and published on the web. Robots monitor this information, producing statistics on the long-term system performance.

Large user communities, such as ATLAS and LHCb, profit from encapsulating shared use cases as specialised applications in GANDA. In contrast, individual researchers or developers in the context of rapid prototyping activities may prefer to use generic application components. In such cases, GANDA still provides the benefits of bookkeeping and a programmatic interface for job submission. As an example of this approach, a small community of experts in the design of gaseous detectors use GANDA to run the GARFIELD [178] simulation program on the Grid. A GANDA script has been written that generates a chain of simulation jobs using the GARFIELD generator of macro files and GANDA’s Executable application component. The GARFIELD executables, and a few small input files, are placed in the input sandbox of each job. Histograms and

---

[136] data collected by automatic monitoring system counting user accounts with unique names
text output are then returned in the output sandbox. This simple approach allowed integration of GARFIELD jobs in GANGLA in just a few hours.

5.4 Massive molecular docking for Avian Flu

This section cites the results obtained by an independent team which used the GANGLA/DIANE overlay to perform massive molecular docking with the EGEE Grid resources in the search of Avian Flu cure. It provides an independent assessment of our research work on User-level Overlay.

*In silico* drug discovery is an increasingly important method to reduce costs and accelerate identification of molecules for treatment of viral diseases [173]. It allows to study the impact of mutations on drug resistance what became a particularly hot subject in recent years due to several outbreaks of influenza pandemic. One of the first demonstrations of usage of grid infrastructures for high-throughput virtual screening was performed at the time of the first outbreak of the Influenza Neuraminidase N1 (Avian Flu) in 2006 and was reported in [115]. The importance of the results obtained by this activity is twofold: 1) it addresses a problem with a possible immediate and enormous impact on everyday life, and 2) it provides an independent assessment of the User-level Overlay tools and techniques applied in EGEE Grid.

Virtual screening consists of several steps, including molecular docking as a key element. Molecular docking is a simulation method allowing calculation of the binding energy between a receptor (a molecule of a virus) and a ligand (a drug molecule). Docking process is performed on a large collection of drug molecules against a certain number of virus mutations. The results are ranked according to the efficiency of the binding. Selected molecules may then be post-processed by more accurate modeling techniques, such as molecular dynamics, and finally tested *in vitro* [45].

The usage of GANGLA/DIANE overlay for Grid-enabled high-throughput in-silico screening against Avian Flu was reported in detail in [115, 98]. It was compared to WISDOM [99] - a more traditional system based on early binding. More than 300 thousand chemical compounds where docked against 8 protein targets, generating 120 thousand output files. The docking space was divided into 8 disjoint parts and distributed for processing to a group of researchers. One researcher used GANGLA/DIANE for processing, all others used the WISDOM platform. Hence the DIANE-based system was assigned 1/8 of the total screening space. We cite the activity summary in Tab. 5.1, where crunching factor is defined as the speedup obtained by the system and distribution efficiency corresponds to the speedup divided by the worker pool size (maximum number of concurrent CPUs).

Several conclusions may be drawn for the late-binding DIANE-based system in comparison to a classical job submission approach applied by WISDOM. In DIANE one task corresponded to a single docking simulation which was estimated to take c.a. 30 minutes on a standard 2006 PC. In WISDOM docking was performed in batches of 40 simulations. Late-binding model allowed to use resources more efficiently and to double the processing efficiency as shown in Tab. 5.1. This was due to “the feature of interactively returning part of the computing efforts during the runtime (e.g. the output of each
Table 5.1: Summary of DIANE and WISDOM activity in 2006 virtual screening, including the “crunching factor” as a measure of task processing efficiency. The screening space was divided into 8 disjoint parts and assigned to DIANE (1/8) and to WISDOM (7/8). Source: [115]

<table>
<thead>
<tr>
<th></th>
<th>DIANE</th>
<th>WISDOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of completed dockings</td>
<td>308,585</td>
<td>$2 \times 10^6$</td>
</tr>
<tr>
<td>Estimated duration on 1 CPU</td>
<td>16.7 years</td>
<td>88.3 years</td>
</tr>
<tr>
<td>Activity duration</td>
<td>4 weeks</td>
<td>6 weeks</td>
</tr>
<tr>
<td>Cumulative number of Grid jobs</td>
<td>2585</td>
<td>54,000</td>
</tr>
<tr>
<td>Maximum number of concurrent CPUs</td>
<td>240</td>
<td>2,000</td>
</tr>
<tr>
<td>Number of used Computing Elements</td>
<td>36</td>
<td>60</td>
</tr>
<tr>
<td>Crunching factor</td>
<td>203</td>
<td>912</td>
</tr>
<tr>
<td>Approximated distribution efficiency</td>
<td>84%</td>
<td>46%</td>
</tr>
</tbody>
</table>

docking) which introduces a more economical way of using the Grid resources” [115]. A quick comparison based on data in Tab. 5.1 allows to conclude that, while average job duration in WISDOM was c.a. 20 hours, in DIANE it was 120 hours. It was also concluded that “a constant throughput can be effortlessly maintained for few weeks using the task pull model” [115]. This confirms the benefits of late binding for reducing operational effort required to manage large computing activities.

During the screening activity around 83% of jobs were reported as successfully completed but only 70% of jobs produced useful output. The difference was accounted to problems with file transfer to Grid storage elements. Since the same data management technique was used in DIANE and in WISDOM - similar success rates were reported. It was noted, however, that “the failure recovery mechanism in DIANE automated the re-submission and guaranteed a fully complete job” [115].

Scalability issues were reported with the bi-directional communication logic implementation of DIANE version 1.9 which was used at the time. In consequence this limited the crunching factor obtained from a single DIANE RunMaster. In the subsequent DIANE versions 2.x the Core framework interaction with the transport layer was refactored and this limitation was removed.

Finally, it must be noted that, in the context of virtual screening, Academia Sinica in Taipei developed a Grid Application Platform\(^5\) (GAP) which allows biologists and other researchers to use a web-based, domain-specific portal to perform screening in distributed environments. The platform embeds GANa and DIANE as components of a web service backend to access Grid resources and perform task management as shown in Fig. 5.10. This is another demonstration of a flexibility of the User-level Overlay.

\(^5\) available at [http://gap.grid.sinica.edu.tw](http://gap.grid.sinica.edu.tw)
5.5 Other examples of using DIANE/Ganga overlay

Several other application use-cases of User-level Overlay are worth mentioning. The Ganga/Diane system was used at CERN for Grid-based numerical evaluation of Feynman loops [107] as well as by other scientific groups in the context of task processing for microscopic image alignment using maximum-likelihood refinement [35]. Running distributed BLAST application for genomics research was described in [140]. Successful use of Ganga was reported in [174] in the context of automated analysis and recognition of image content for a novel, commercial search engine [175]. Simulation and analysis of alignment and statistical errors associated with measurements in MICE (Muon Ionizing Colling for Neutrino Factory) was performed on the Grid using Ganga what allowed “the study to be easily understood, repeated and modified by members of the collaboration who presently lack Grid experience”[63]. Minersoft [149] uses Ganga as a job manager to implement a software discovery service which uses Grid crawlers and harvesters to automatically locate, categorize and index application software available in large production grids. Ganga/Diane components are currently used to build a prototype for a distributed simulation and analysis system for environmental studies in the context of EnviroGrids project [116]. Ganga is also used to manage MPI jobs for numerical weather prediction for the Mediterranean Area [112].

Experimental use of the system has been reported in the context of Google Summer of Code 2009 project “Distribution of High Performance Computing Jobs among Multiple Computing Clouds”\(^6\). Ganga has also been interfaced to a general purpose

\(^6\)http://socghop.appspot.com/gsoc/student_project/show/google/gsoc2009/globus/
master-worker parallel computation PYTHON module called PyMW [87]. PyMW is intended to support rapid development, testing and deployment of large-scale master-worker style computations on a desktop grid or volunteer computing environment. In another context, reliability studies, performed with GANJA, have been reported as an MSc thesis in [80].

5.6 Summary

In this Chapter we demonstrated successful application of the User-level Overlay in a variety of use cases. The flexibility required to tackle scientific computing problems is needed at multiple levels, from an ability to link with external software components, such as workflow engines, and an ability to customize and adapt to existing application frameworks at the source-code level, to easy access and job configuration management for large user communities and legacy applications. A particular value presented in this Chapter comes from the fact that scientific and engineering ideas and tools developed in this thesis have been successfully extended and applied by others.