

File ID	212268
Filename	Thesis

SOURCE (OR PART OF THE FOLLOWING SOURCE):

Type	Dissertation
Title	Understanding and mastering dynamics in computing grids: processing moldable tasks with user-level overlay
Author	J.T. Mo#cicki
Faculty	Faculty of Science
Year	2011
Pages	viii, 178

FULL BIBLIOGRAPHIC DETAILS:

<http://dare.uva.nl/record/373814>

Copyright

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 then for strictly personal, individual use.



Jakub T. Mościcki is a researcher and software engineer at CERN, Geneva, Switzerland. He obtained the MSc in Computer Science from the AGH University of Science and Technology in Kraków, Poland. He is a lead developer of the Ganga project and creator of the DIANE framework, which are used to support very large LHC user communities as well as users of multidisciplinary applications in theoretical physics, medical and radiation studies, bio-informatics, drug design, telecommunications. His research interests focus on scheduling and management of distributed and parallel applications, large-scale computing infrastructures such as grids, and various forms of High Throughput and High Performance Computing.

Thousands of scientific users witness every day inherent instabilities and bottlenecks of large-scale task processing systems: lost or incomplete jobs and hard-to-predict completion times. They are struggling to resubmit failed jobs and get consistent results. And it is always difficult to catch up with latest deployed software environments or system configurations. In addition, the users have often more than one system to deal with: they continue to use locally available computing power (a desktop PC, a nearby computing center, a small cluster next door) while exploiting global resources such as grids. On top of this, grids use a large variety of middleware stacks, which are customized in different ways by user communities. **Quality of Service** and **usability** are the two keywords probably most frequently echoed in the corridors of many "grid-enabled" research labs.

This PhD dissertation presents scientific research from the problem statement, system analysis, modeling and simulation, to validation through experimental results. It captures and characterizes **complexity** and **dynamics** of global task processing systems using as an example the largest scientific grid to date - the **EGEE/EGI Grid**. A task processing model developed in this work allows to rigorously explain why the **late-binding** method is superior to traditional **task scheduling** based on early binding. A study of statistical properties of task processing times is complemented by Monte Carlo simulation.

This book is also **addressed to grid practitioners**: developers and users. Presenting several successful application examples from diverse domains, it explains how heterogeneity and dynamics of global task processing systems may be addressed and mastered in a cost-effective way directly by the users. It describes a **User-level Overlay**, based on two software packages, Ganga and DIANE, which are ready to use with little or no customization for **your application**. Advanced resource selection strategies and scheduling approaches developed in this book may be reused in **your environment**.



5 800054 694140

Understanding and Mastering Dynamics in Computing Grids

Jakub T. Mościcki

Understanding and Mastering Dynamics in Computing Grids

Processing Moldable Tasks with User-Level Overlay

Jakub T. Mościcki

Understanding and Mastering Dynamics in Computing Grids: Processing Moldable Tasks with User-Level Overlay

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. D.C van den Boom
ten overstaan van een door het college voor promoties
ingestelde commissie, in het openbaar te verdedigen
in de Agnietenkapel
op dinsdag 12 april 2011, te 12:00 uur

door

Jakub Tomasz Mościcki
geboren te Kraków, Polen.

Promotiecommissie

Promotor:	Prof. Dr. Marian T. Bubak
Co-promotor:	Prof. Dr. Peter M.A. Slood
Overige leden:	Prof. Dr. Hamideh Afsarmanesh Dr. ir. Alfons G. Hoekstra Dr. Juergen Knobloch Prof. Dr. ir. Cees Th.A.M. de Laat Prof. Dr. Krzysztof Zielinski
Faculteit:	Faculteit der Natuurwetenschappen Wiskunde en Informatica



This work makes use of results produced by the Enabling Grids for E-sciencE project, a project co-funded by the European Commission (under contract number INFSO-RI-222667) through the Seventh Framework Programme. EGEE brings together 91 partners in 32 countries to provide a seamless Grid infrastructure available to the European research community 24 hours a day. Full information is available at www.eu-egee.org and www.egi.eu

The book cover uses Stacy Reed's *Chaos From Order* artwork which she kindly shared with me for this purpose. Stacy is a diverse artist who enjoys exploring chaos through fractal applications. *Chaos From Order* represents the notion that in evolution, chaotic shapes and patterns, mutations, abnormalities and anomalies emerge over time, from what was once, in this case, a perfect mathematical form. More of her artwork can be viewed by visiting www.shedreamsindigital.net.

Author contact: jakub.moscicki@cern.ch
Printed by lulu.com
Copyright © 2011 Jakub T. Mościcki

Table of Contents

1	Motivation and research objectives	1
1.1	Distributed applications: common patterns and characteristics	2
1.2	Infrastructures for scientific computing	8
1.3	Higher-level middleware systems	9
1.4	User requirements	13
1.5	The research objectives and roadmap	15
2	Dynamics of large computing grids	19
2.1	EGEE – world’s largest computing and data Grid	19
2.2	Grid as an infrastructure	22
2.3	Grid as a task processing system	27
2.4	Summary	39
3	Analysis and modeling of task processing with late binding on the Grid	41
3.1	Introduction	41
3.2	Task processing model	42
3.3	Distribution of job queuing time	44
3.4	Simulation of task processing models	48
3.5	Summary	57
4	Development of the User-level Overlay	59
4.1	Vision	60
4.2	Functional breakdown and architecture	62
4.3	DIANE and Ganga software packages	63
4.4	Operation of the User-level Overlay	64
4.5	The DIANE task coordination framework	66
4.6	The Ganga resource access API and user interface	73

4.7	Heuristic resource selection	80
4.8	Adaptive workload balancing	85
4.9	Summary	89
5	User-level Overlay in action	91
5.1	Monte Carlo simulation with Geant4 toolkit	92
5.2	Workflows for medical imaging simulations	99
5.3	Data processing for ATLAS and LHC b experiments	102
5.4	Massive molecular docking for Avian Flu	103
5.5	Other examples of using DIANE/Ganga overlay	105
5.6	Summary	106
6	Capability computing case study: ITU broadcasting planning	109
6.1	Introduction	109
6.2	Broadcasting planning process	110
6.3	Compatibility analysis	111
6.4	Implementation of grid-based analysis system for the RRC06	113
6.5	Analysis of task processing	115
6.6	Summary	120
7	Capacity computing case study: LatticeQCD simulation	121
7.1	Introduction	121
7.2	Problem to be solved	122
7.3	Simulation model	123
7.4	Implementation and operation of the simulation system	125
7.5	Task scheduling and prioritization	130
7.6	Analysis of adaptive resource selection	137
7.7	Exploiting low-level parallelism for finer lattices	139
7.8	Summary	140
8	Conclusions and future work	143
8.1	Grid dynamics and its consequences for task processing	143
8.2	Contributions of this work	144
8.3	Open issues	146
8.4	Future work	147
8.5	Postscriptum	148
	Bibliography	164
	Summary	165
	Nederlandse samenvatting	167
	Streszczenie po polsku	169
	Publications	171

Acknowledgments	175
Index	177

List of Abbreviations

AWLB	Adaptive Workload Balancing
CDF	Cumulative Probability Distribution Function
CE	Computing Element
CERN	European Laboratory for Particle Physics
CORBA	Common Object Request Broker Architecture
CREAM	Computing Resource Execution And Management
DIANE	Distributed Analysis Environment
EGEE	Enabling Grids for e-Science
EGI	European Grid Initiative
HAF	Heuristic Agent Factory
HPC	High Performance Computing
HTC	High Throughput Computing
LHC	Large Hadron Collider
LQCD	Lattice Quantum Chromodynamics
MPI	Message Passing Interface
MPP	Massive Parallel Processing
MTA	Moldable Tasks Application

NDGF	Nordic DataGrid Facility
OpenMP	Open Multi-Processing
OPS	Operations VO
PDF	Probability Density Function
QoS	Quality of Service
RB	Resource Broker
SLA	Service Level Agreement
SMP	Symmetric Multi-Processing
VO	Virtual Organization
VOMS	Virtual Organization Management Service
WLCG	Worldwide LHC Computing Grid
WMS	Workload Management System
WN	Worker Node

CHAPTER 1

Motivation and research objectives

When we had no computers, we had
no programming problem either.
When we had a few computers, we
had a mild programming problem.
Confronted with machines a million
times as powerful, we are faced with
a gigantic programming problem.

E.W.Dijkstra

Scientific research is driven by two major forces: curiosity and utility. Pure science serves mainly the curiosity and sometimes generates utility as a by-product. On the other hand, applied sciences are *by definition* focused on utility: boosting progress in technology and engineering. The computer science is special in the sense that its main utility is to facilitate, support and sometimes even enable scientific progress in other fields. This is particularly true nowadays as science is done with computers and scientific communities use a growing number of computing systems, from local batch systems and community-specific services to globally distributed grid¹ infrastructures.

Increasing the research capabilities for science is the *raison d'être* of scientific grids which provide access to diversified computational, storage and data resources at a large scale. Grids federate resources from academia and public sector (computing centers, universities and laboratories) and are complex, highly heterogeneous, decentralized systems. Unpredictable workloads, component failures and variability of execution environments are normal modes of operation. The time cost to learn and master the interfaces

¹We use the lowercase term *grid* when referring to any grid infrastructure or to the computing grid as a concept. We use the uppercase term *Grid* when referring to the particular EGEE Grid infrastructure.

and idiosyncrasies of these systems and to overcome the heterogeneity and dynamics of such a distributed environment is often prohibitive for end users.

In computer science understanding of a system, which may include development of a methodology and mathematical modeling, is a prerequisite for extracting system's properties and modifying or controlling its behavior. Understanding patterns and characteristics of applications and distributed systems in current scientific context is a challenging task. Nonetheless it is essential for finding efficient methods, mappings and building appropriate support tools.

Moldable Task Applications are the majority of computational tasks processed in grids nowadays. In this thesis we analyze and develop strategies which allow user communities to overcome heterogeneity and dynamics of grids in the context of running such applications. Let's start with the review of common patterns and characteristics of computational tasks in this context.

1.1 Distributed applications: common patterns and characteristics

The three grand aspects of interest for today's large-scale scientific computing are the task processing, data management and distributed collaboration. In this work we address problems related to processing of computational tasks.

1.1.1 Moldability

Moldability is an important property of processing of computational tasks in parallel and distributed computing environments.

The concept of moldability has been first introduced in the context of tightly-coupled computing systems as a property allowing more flexible scheduling of parallel jobs [61, 40]. Moldable jobs have been defined in supercomputing as parallel jobs which "can execute on any of a collection of partition sizes" [41]. In tightly-coupled systems scheduling problems are of primary concern and therefore Feitelson and Rudolph [60] provided a detailed taxonomy to distinguish between rigid, moldable, evolving and malleable jobs. The different job classes define different models of interaction of application with execution environment, allowing for different scheduling policies. With moldable jobs, the number of processors is set at the beginning of execution and the job initially configures itself to adapt to this number. Evolving jobs may change their resource requirements during execution, such that it is the application that initiates the changes. If the system is not able to satisfy these requirements the job may not continue. The most flexible type of jobs are malleable ones that can adapt to changes in the number of processors during execution.

Rigid jobs in traditional parallel applications typically imply a fixed number of processors which depends directly on the decomposition of application domain, parallelisation algorithm and implementation strategy (message passing, shared memory, etc.). Such applications typically require simultaneous access to a fixed number of homogeneous resources (partitions). They often imply task parallelism which may not scale well

with the size of the problem. For example, numerous solvers in physics and engineering which are based on Finite Element Methods through domain decomposition are often implemented as rigid parallel jobs.

In the context of loosely-coupled, large-scale computing systems we define moldability as an ability to partition the computation into a number of parallel execution threads, tasks, jobs or other runtime components in a way which is partially or fully independent of a number of available processors and which may be variable in time. Therefore, our definition of moldability also embraces Feitelson's malleable and evolving jobs.

The degree of parallelism of moldable jobs may vary and may be flexibly adapted to the number of available resources at a given time. Moldable job application examples include Monte Carlo simulations, parameter sweeps, directed acyclic graphs and workflows, data-parallel analysis algorithms and many more.

Moldable jobs may execute on a variable set of heterogeneous resources. For example, many Monte Carlo simulations in physics consist of a large number of independent events which may be aggregated in work units of almost arbitrary size [46]. Granularity of task partitioning is often defined by the problem space. For instance, molecular docking in bioinformatics is a parameter sweep application where the processing of an individual point in the parameter space typically cannot be subdivided. Data analysis in High Energy Physics is an example of embarrassingly parallel application where task execution is constrained by the location and access to input data which may involve non-trivial execution patterns.

1.1.2 Communication

Communication requirements are fundamental for parallel and distributed applications where the data flows between physically separated processing elements. The amount and frequency of this communication depends on the granularity of parallel decomposition and, with some simplification, corresponds to the ratio between the amount of computation and communication required in a unit of time. Parallel applications which are typically executed in local, dedicated networks such as Myrinet or Infiniband or within internal supercomputer interconnects, may efficiently perform frequent and massive communication operations every few computing steps [171].

In large grids, however, the application elements are distributed over wide area, often public networks, where latency, and to some extent also bandwidth, are the limiting factors. Therefore, only applications with relatively low communication requirements may be efficiently implemented in grids. In practice, Grid jobs do not typically require communication more frequently than every several minutes or hours.

To fully exploit the scale and dynamic resource sharing in the grid, multiple grid sites are needed to be used at the same time by a single application. However, high-latency WANs impose serious communication constraints on applications such as using cross-cluster message passing. As a result the deployment of traditional tightly-coupled applications is often limited to clusters within a single grid site.

1.1.3 Data

Data-intensive science leads to *data deluge* [89] where storing, retrieving, transforming, replicating and otherwise organizing distributed data becomes a grand problem on its own. On global scale, issues such as efficient data management systems for applications which produce Petabytes of data within days prevail. A recent example in High Energy Physics was provided by STEP09 exercise at CERN, reported in more detail in [54]. One common strategy is to move processing close to data and it must coexist with strategies for asynchronous data replication across many sites to improve the degree of parallelism of data access. On local scale, the data access to storage systems is the key where storage solutions range from tape-based storage solutions such as Castor [156] to disk-oriented file-systems such as Hadoop [180] or Lustre [3]. Transactional databases and handling of metadata represent yet another dimension of complexity. Distributed data storage is also an additional source of application failures as many of the basic data handling tools, such as GridFTP [12], are known to have reliability issues.

From a processing perspective, distributed computational tasks are producers and consumers of data and the physical location and amount of input and output data is application-specific. For some heavily I/O intensive applications, such as Data Analysis for LHC experiments at CERN, the execution bottlenecks may be related to the performance and configuration of local storage systems [177]. Inefficiencies may arise due to particular ways in which the application interacts with the storage system. For example, *staging in* is a common technique to copy entire input files into a local disk from a mass storage system before processing effectively starts. On the other hand *streaming* uses specialized networks protocols to fetch portions of files on-demand as the task processing occurs. The time needed to access input data and store output data accounts for the wall-clock time of the task execution (as opposed to the CPU-time) and may be treated as an internal parameter of the task processing efficiency.

The efficiency of data access proves increasingly complex as applications often use rich software stacks for the I/O, including libraries which provide high-level data abstractions. In such cases, apparently sequential data access is in reality translated into complex random-access patterns. This is, for example, a case of ROOT [19] I/O Tree library, heavily used in High Energy Physics.

The distribution of input data affects task scheduling as it puts additional constraints which may result in non-optimal global usage of computing resources. For applications with very large data volumes, say $\mathcal{O}(100)$ TB or more, this issue requires special handling. However, a large fraction of applications running nowadays in grids has much smaller requirements on input and output sizes. For many applications, especially of Monte-Carlo type, not only the size of input data is negligible but also size of the output is small enough, say $\mathcal{O}(100)$ GB, to be returned directly to the user.

In some cases and up to a certain data size there is no real need of having permanently distributed data across a grid as the machine time-cost of making on-the-fly network copies is lower than the human time-cost of supporting the distributed data management system. Sometimes, especially in the medical applications, there are additional socio-political aspects which constrain arbitrary copying of data and enforce *de jure* specific data distribution and access models. Data transfer services such as TrustedSRB [146]

have been developed to meet the demands for data privacy and security.

In this research, we do not analyze nor model the interactions of applications with storage systems. Instead we assume the efficiency and reliability of data storage and retrieval to be internal application parameters. Hence, for example, an application which requires data transfers from remote storage elements at runtime will most likely be less reliable and less efficient than an application which uses local storage only. Reliability may simply be measured as a normalized² fraction of failed jobs and efficiency as a ratio of normalized execution times. Therefore, wherever possible, for certain types of applications, we will seek simple ways of accessing and storing data directly in the local storage space of the end user.

1.1.4 Coordination and synchronization

Some moldable jobs which result from embarrassingly parallel applications do not require coordination: jobs are executing fully independently and results are collected and merged out-of-grid by the user. If the merging step is done in the grid then this implies, however, some coordination mechanism. In this case an application may be represented as a Directed Acyclic Graph (DAG), where the merging job is launched after successful completion of all worker jobs. Workflows follow as an extension and generalization of DAGs and are used in medical simulation applications such as GATE [100] where jobs may be launched dynamically with the flow control defined by a workflow engine and according to criteria evaluated at runtime. Many applications follow other distributed programming abstractions [101] and skeletons [44] which include bags-of-tasks and task-farming in master-worker model, data-processing pipelines, all-pairs and map-reduce. For example, microscopic image processing [35] involves iterative lock-step algorithm, where in a single step a large number of images is analyzed in parallel. The results are combined and refined in the subsequent steps until a desired accuracy of the image alignment is reached.

1.1.5 Time-to-solution and responsiveness

For many traditional, massively distributed applications High Throughput Computing (HTC) or Capacity Computing [125] have been used as the terms to describe the use of many computing resources over long periods of time to achieve a computational task. In HTC applications time-to-solution is not a principal requirement. Conversely, High Performance Computing (HPC) or Capability Computing [125] are the terms used for applications where time-to-solution is important and which exploit large computing power in short periods of time. Traditionally HPC is associated with supercomputers or cluster environments. A recently coined term Many Task Computing (MTC) [158] addresses applications, which similarly to HPC require large computing power in short periods of time and, at the same time, similarly to HTC, may exploit many distributed resources to accomplish many computing tasks. Such applications become increasingly

²Reliability and efficiency of job execution is influenced by heterogeneity of grid processing elements. The purpose of normalization is to disentangle the application and grid features.

important, also outside of typical scientific context. For example, grid-enabled decision support application for telecommunication industry during ITU Regional Radio Conference 2006 required $\mathcal{O}(10^5)$ tasks to be completed under few hours deadline [141].

Responsiveness is an important feature of interactive and short-deadline application use cases. For example, grid-enabled medical image analysis requires timely (and reliable) delivery of the results for intervention planning or intra-operative support in a clinical context [73]. In medical physics, such as radiation studies and radiotherapy simulations [62, 34] precise estimation of the effects of radioactive doses may require a quasi-interactive response of the system when simulation parameters change.

For many end users one important ability is to follow the evolution of the computation and, if necessary, to take corrective actions as early as possible. Timely delivery of results, even if partial and incomplete, is especially important for the man-in-the-loop scenarios, where human interventions are implied during the computational process. This may include a very common use case of application development where users are testing (and maybe even debugging) the application code in a grid.

Interactive, or close-to-interactive and sometimes called interactive-batch, style of work is required in final steps of analysis of physics data. Such use cases are typically arising from visualization-oriented environments such as ROOT or RAVE [79]. The computational steering applications such as on-line visualization are at the extreme end of the spectrum and are fully interactive.

1.1.6 Failure management

Runtime failures of distributed application elements are inevitable in large systems. Simple failover strategies such as job resubmission may be applied in typical Monte-Carlo simulations, where some tasks may remain uncompleted provided that the total accumulated number of statistics is sufficiently high. Condor [170] was one of the first systems which successfully applied simple failure management strategies for processing of large number of jobs. For other applications, such as parameter sweeps, successful completion of all tasks is necessary and efficient failure management becomes more difficult, as it should take into account possible failure reasons. Grid-enabled in-silico screening against popular diseases using molecular docking [115] relies on reliable scanning of the entire parameter space. Failure management may become very complex for some applications which produce complex output stored in databases and which require one instance of each task to be executed exactly once at a given time. In such cases redundancy by parallel execution of many instances of the same task are not allowed.

From the end-user perspective, automatic recovery from failures without too high impact on the performance and throughput of the system is one of the key features. This applies for instance to Grid-enabled regression testing of large software packages such as Geant4 [13] where a few thousand test-cases with different configurations must be completed successfully.

1.1.7 Task scheduling and prioritization

Application-aware scheduling³ and prioritization⁴ of tasks is a non-trivial issue [159]. For example in the Lattice QCD thermodynamics simulation [142], the tasks are prioritized dynamically in the parameter space of the application to maximize outcome in terms of scientific information content. Scheduling in this context becomes a difficult problem because the sequential overhead in this application is very large, both in absolute terms and as a fraction of entire computation. Typical speedup analysis - solving problem of a fixed size faster (Amdahl's Law [14]) or solving a larger problem in fixed time (Gustafson's formulation [86]) - may not be easily applied because the number of available processors varies on much shorter time scales compared to the duration of the computation. Due to very large sequential execution overhead spawning new simulation tasks and adding new resources does not immediately increase the speedup.

Other scheduling difficulties may arise when it is not possible to control the granularity of tasks so that a user has not a priori knowledge and effective control on splitting of workload. For example, during ITU RRC06 processing, the task duration spanned three orders of magnitude according to a statistical distribution. Without a priori knowledge of task sizes, dynamic scheduling at runtime is required to correctly balance the workload.

1.1.8 Qualitative resource selection

Some applications require a combination of many different computing architectures, hence many *qualitatively* different resources, to accomplish computational tasks in a maximally efficient and cost-effective way. One such example is the previously mentioned Lattice QCD thermodynamics application, where supercomputing and grid resources may be used in different phases of the same computational activity as a best trade-off between cost and speedup. Such a mixed use is required due to scaling properties and internal structure of this particular Lattice QCD simulation, where the spatial size of the lattice is of a moderate size. For larger lattice sizes it would be advantageous to dynamically combine shared-memory parallel processing (such as the OpenMP standard) to use multicore resources available in the grid and the processing based on message-passing on clusters or supercomputers such as the Message Passing Interface (MPI) [166].

1.1.9 Summary

There is a large and growing class of important applications which we call *Moldable Tasks Applications (MTAs)*. MTAs share similar characteristics in the context of distributed processing environments: moldability, loose coupling of application elements, low communication requirements between processing tasks, non-trivial coordination and scheduling requirements. MTAs are often legacy, sequential applications (such as black-box FORTRAN executables) which may not be easily modified for porting into the

³Scheduling is a problem of mapping of a set of activities onto a set of limited resources over time.

⁴Prioritization is a problem of ordering of activities if time is limited.

distributed environment. This class comprises also I/O bound applications if data movements are not considered, according to the paradigm “move processing close to data”. Time-to-solution and reliability of processing is an important aspect for these MTAs. The scientific problems which we address in this thesis are related to processing of MTA applications.

1.2 Infrastructures for scientific computing

Scientists have at their disposal a wide and growing range of distributed computing infrastructures, from supercomputers, dedicated computing clusters, GPU computing processors and batch farms to grids, clouds and volunteer computing systems such as BOINC [15]. Growth in a variety of the systems themselves comes hand in hand with the growth in scale and complexity of each individual system.

Grids, which are in our focus, are a good example of how large and heterogeneous distributed computing environments add to the applications spatial and temporal dimensions of complexity. The portions of application are distributed across heterogeneous software and hardware systems and communicate over non-uniform, wide area networks. As the scale increases the component failures becomes a normal mode of operation⁵. The workloads are unpredictable as resources are not granted for exclusive usage in time and sharing with other users is subject to local and diversified mechanisms.

Last years have seen countless grid projects and initiatives, ranging from small experimental testbeds to large production infrastructures. Although the main goals for the major production grids remain similar and compatible with Foster’s checklist [64], the driving forces and following design choices were different despite the fact that most of the grids build on top of a subset of the protocols and services defined by the Globus project.

Supporting large-scale data processing in context of High Energy Physics was in the focus of projects derived from the EDG/gLite family of middleware such as DataGrid, WLCG, EGEE⁶ or projects such as NorduGrid/NDGF⁷ where major stakeholders are LHC experiments at CERN. Despite some design differences these HTC grids are federations of classical batch processing farms. They remain fundamentally batch-job oriented infrastructures where job execution times exceeding several days are common. Middleware services such as gLite Workload Management System (WMS) and CREAM CE [9] provide scheduling at the global and Virtual Organization (VO) levels⁸. As the round-the-clock operations and global service are in focus in these production grids, certain aspects of the infrastructure have been standardized, such as the worker node environment based on Scientific Linux operating system. In other projects, such as the OSG⁹,

⁵A simple back of the envelope calculation based on Mean Time Between Failure MTBF=1,2 million hours for typical disk drives yields roughly 1% of component failures annually. In a system with, say 100 thousand elements, this means a failure every 10 hours on average.

⁶<http://www.eu-egee.org>

⁷<http://www.ndgf.org>

⁸Virtual Organization is a group of users sharing the same resources. Members of one Virtual Organization may belong to different institutions.

⁹<http://www.opensciencegrid.org>

the resource providers have been the main driving force. This implies more heterogeneity of worker node environments as well as different policies for acceptable job durations (shorter jobs are more common).

HPC grids, such as the TeraGrid¹⁰, integrate diverse high-performance computers, SMPs, MPPs and clusters, via dedicated high-performance network connections. TeraGrid is a highly heterogeneous environment where uniform access to specialized computing resources is provided by GRAM services which are implemented above local batch systems such as LSF, PBS, LoadLeveler, Torque+Moab and Condor. TeraGrid provides access to high-performance architectures such as Cray XT5, IBM BlueGene/L, SGI Altix and a variety of 64-bit Linux clusters. DEISA (Distributed European Infrastructure for Supercomputing Applications) is a supercomputing grid infrastructure based on UNICORE middleware stack and WSRF standard. It enables access to a variety of architectures such as IBM BlueGene/P, NEC SX8/9 vector system or Cray XT4/5.

In large-scale distributed environments for scientific computing heterogeneity and variability prevail. The control of resources remains under multiple administrative domains. These domains are connected by external networks where latencies are high, topologies are complex and firewall restrictions subject to local policies. Software and hardware environments may greatly differ from one domain to another, as well as the reliability of the individual components of the system. The load of individual components changes dynamically and outside of user's control. The access to computing resources is typically implemented via job management systems where job waiting time and submission overheads may be very high.

Very large grid infrastructures are complex and dynamic systems due to a diversity of hardware, software and access policies [93]. Grids are also decentralized in nature – chaotic activity of thousands of users and many access- and application patterns make it difficult in practice to effectively predict global grid behavior despite existing models for job inter-arrival times [38], [117] and attempts of predicting job numbers in individual clusters [129]. In consequence, considerable operation efforts are needed to provide desired quality of service in virtually every large production grid such as EGEE, OSG, TeraGrid and NDGF. Independent observations and everyday user experience confirm that large and unpredictable variations in performance and reliability of the grids are commonplace [94].

1.3 Higher-level middleware systems

Large-scale grids do not provide efficient mechanisms to reinforce Quality of Service (QoS). The batch processing model induces large overheads which are not acceptable in many scientific applications. Therefore, the trend to provide higher-level, application-aware middleware above a set of basic grid services has been significantly increasing in the last decade. Main areas of work in this context include meta-scheduling techniques, middleware architectures and application execution environments.

¹⁰<http://www.teragrid.org>

1.3.1 Meta-scheduling with late binding

Late binding has been increasingly used as a meta-scheduling technique in recent years. This meta-scheduling technique is also known as pilot jobs, worker agents, infiltration frameworks [78] or placeholder scheduling [153]. Late binding is a scheduling and coordination method, where the work is assigned to a job at runtime rather than at submission time.

Historically, Condor glide-in [67] was the first late-binding technology used in the grid context. It has been subsequently reused for implementing higher-level services such as the glideinWMS [164]. However, distributed infrastructures are continuously evolving and early binding is now an established approach for grid and batch systems. In a typical early-binding case, a job carries one task which is specified when the job is submitted. In a late-binding system one or more tasks are provided dynamically to a job agent while it is running on a worker node.

A late-binding system may operate at several levels. At a virtual organization level it may be used to control and organize access to distributed resources by its members. At application or user level it may provide application-aware scheduling and help coping with variability and complexity of computing environments.

Specialized implementations of scheduling and task coordination mechanisms are often embedded in the applications themselves as it is a case for 3D medical image analysis [74], earthquake source determination [43] or MPI-BLAST [49] – a parallel implementation of the Basic Local Alignment Tool (BLAST) which is a de facto standard in genomic research and which is widely used for protein and DNA search and alignment in bioinformatics. In the case of MPI-BLAST the task management and book-keeping layer was implemented using a known HPC technology such as the MPI and then ported into grid environments using specialized implementations such as MPICH-G2 [102]. There are a number of problems with such an approach:

- support for inter-site MPI requires simultaneous execution of job arrays and such a capability is currently limited in large-scale grids, so the MPI jobs are effectively restricted to single clusters;
- the MPI was designed to build complex parallel applications rather than job management layers so the cost of development is relatively high;
- the management layer must be constantly maintained and adapted to the changing grid environment.

Moreover, it is impractical and inefficient to re-implement the same scheduling and coordination patterns for every application. Therefore, structured approaches to provide application-level scheduling have been investigated: AppLeS/APST [27] provides a framework for the parameter sweep applications and adaptive scheduling; the Condor M/W [165] provides a framework for master/worker applications. In this context an approach of selecting a static set of resources from an infinite resource pool was investigated. In practice, this is not sufficient as the resources are available only for a fraction of required time (queues and job lifetime limitations) and the problem must be reversed: resources are dynamically drawn from a pool and recycled over larger periods

of time. Additionally Condor M/W is Condor-specific technology and therefore is available only if the job is controlled by Condor scheduler. Advanced scheduling policies are not readily available for the EGEE Grid users, despite the fact that internally gLite WMS uses Condor-based components. One unanswered issue in the context of Condor M/W work was how to acquire dynamically changing environment parameters to overcome limitations of external monitoring tools such as Network Weather Service [183] which provide steady-state approximations for dynamically changing environment. It was concluded that integrating better information leads to better work schedules [183].

Living application [81] provides an interesting example of a method to autonomously manage applications on grids, such that during the execution the application itself makes choices on the resources to use based on internal state or autonomously acquired knowledge from external sensors.

Several late-binding systems acting at a level of virtual organization have been developed by the LHC experiments. Permanent overlay systems such as AliEn [161], DIRAC [176] or PANDA [120] proved to be successful over last years and enabled an efficient and fault-tolerant use of grid infrastructures [167]. The advantage of the VO-centric overlays is that they are developed and deployed within the VO boundaries, thus at a smaller scale and in shorter cycles, synchronized with the VO community needs. These systems implement many features such as centralized task queue, file and meta-data catalogs and data management services. Through the late-binding method they improve reliability and efficiency of task processing in grids which is necessary for large data productions. Due to a large scope they require significant, orchestrated efforts of the application community to develop and maintain central job management services and specialized services deployed in grid sites (so-called VO-boxes, where community power users have unlimited root access). Moreover, these systems, serving large communities, rely on dynamic sharing of large number of worker-agent jobs which raises the concerns about security, confidentiality and traceability of user jobs.

Despite the efforts to design them generically, VO overlays tend to be very domain-specific what makes it non-trivial to reuse them beyond their original area of application. For example, in HEP, centrally managed activities such as data production have been the main driver of the development of some VO overlays. In case of more diverse and chaotic end-user analysis tasks, these overlays require significant maintenance efforts and sometimes quasi-continuous refactoring and reengineering to meet the needs of dynamically changing application environments. For this reason mature VO overlays often tend to be domain-specific solutions and are hard to reuse elsewhere.

Late binding has been successfully applied and is now widely adopted in grids. However, to date the mechanisms why late binding is a more robust technique have not been rigorously explained, despite recently developed models [75]. An interesting attempt to match the performance of late binding using early binding and dynamic performance monitoring is presented in [169], however, it is not as robust as late-binding techniques. In [114] a model for scheduling of independent tasks in federated grids is presented, which requires implementation of meta-schedulers on each of grids and to run mapping strategies on them. An approach to scheduling taking into account data distribution is presented in [109]. A general model of scheduling of complex applications (workflows) on grids with multi-criteria is presented in [182].

1.3.2 Application execution environments

Application execution environments developed by the research communities vary from ready-to-use frameworks and portals to toolkits which allow to build specialized application environments.

Nimrod [6] was one of the first parametric modeling systems which used a declarative modeling language to automate task specification, execution and collecting the results with little programming effort. Nimrod/G was subsequently developed as an enhancement and includes dynamic resource discovery, task deadline control and grid security model. Soft QoS implementation based on economy-driven deadline- and budget-constrained (DBC) scheduling algorithms for allocating resources to application jobs was subsequently proposed in [33].

GridWay [91] aims at reducing the gap between grid middleware and application developers by providing runtime mechanisms for adaptive application execution. It has been increasingly used as an interface to Cloud computing resources.

SAGA [56] comes as a set of APIs which allow to build grid-aware applications using simple abstractions. The simplicity comes at a price, however: strict semantics may cause significant overheads and runtime dependencies may limit the existing middleware functionality [56].

Portals and graphical environments represent another trend in bridging the middleware gap by aiming at making the access to grid resources as easy as possible. A collaborative platform called GridSpace for system-level science integrates a high-level scripting environment with a grid object abstraction level hierarchy [31]. It allows building and running applications in a Virtual Laboratory. An approach for semantic integration of virtual research environments was examined in [85].

GENIUS web-based grid portal [18] has been developed as a problem solving environment with the aim that “scientific domain knowledge and tools must be presented to the (non-expert) users in terms of the application science and not in terms of complex computing protocols.”. However, it must be noted that web-based portals are mostly suitable for applications with standard, well-defined workflows which are not tightly integrated with the end-user environment. Therefore, in some user communities such as physics, command-line and scripting tools present a more natural and flexible interface. In practice flexible customization at the level of individual users is often as desirable as customization at the application level (the latter performed by community power users or domain experts).

1.3.3 Middleware architectures

A strategy to provide Quality of Service and missing capabilities through generic middleware has not yet fulfilled its promises due to deployment difficulties in large-scale grids. However, the research communities have developed several interesting approaches in this area.

One approach is to modify existing workload management systems by providing extensions such as the WMSX for gLite described in [24]. This enhanced version of the gLite WMS service allows to submit parameteric jobs and arbitrary workflows.

The authors also claim that it offers additional benefits to the users, such as improved debugging and management of jobs. However, it seems that the system remained at a prototype level.

An alternative approach consists of running parallel applications with topology-aware middleware [21]. In this case, a resource description schema is combined with meta-scheduler and topology-aware OpenMPI implementation which allows dynamic allocation of MPI processes, using colocated process communication groups, such that the communication and computation costs are balanced.

GARA [160] has been developed as an architecture that provides a uniform interface to varying types of QoS, and allows users to make advanced reservations. It allows to set QoS requirements and reservations for network but also for worker nodes, CPU time, disk space, and graphic pipelines. G-QoS [10] system aimed at achieving similar goals using Open Grid Services Architecture (OGSA). Despite being a promising solution, neither GARA nor G-QoS have not been widely adopted for practical use (in the latter case the OGSA standard became obsolete before the G-QoS was able to make impact). G-RSVP [179] represented an attempt to provide resource reservations using mobile agents.

As the QoS is effectively not implemented in large-scale grids, Service Level Agreements (SLAs) remain the primary guarantee for delivering the resources to user communities [4]. However, in contrast to other areas, such as the telecommunication industry [110], systems enforcing SLAs in scientific grids are not mature.

1.4 User requirements

Scientific communities have been the driving force for creation of large computing infrastructures and grids. Nonetheless, problems related to heterogeneity and dynamics in large computing infrastructures persist and are costly for large user communities and prohibitive for many smaller ones. It is so because enabling and supporting applications in distributed environments incurs high costs of time and manpower.

In this section we review main requirements which are addressed by this work. We focus on non-functional requirements which may be generalized and abstracted for a class of MTA applications as opposed to functional requirements which are defined in the scope of concrete applications and which are specific to concrete application domains.

1.4.1 Quality of Service

One of the most fundamental issues pertinent to the successful uptake of the grid technology by the users is the Quality of Service in the context of efficiency, dependability and variability of task processing. Users want to obtain scientific results in reliable and predictable manner even if execution of many hundreds or thousands of tasks is required to achieve these results. This observation is based on everyday experience in supporting users in different communities.

It may be useful to distinguish several typical cases. In parameter sweep or data analysis applications successful completion of *all* tasks is required “as soon as possible”.

Another class of applications includes Monte Carlo simulations, where completion of a large fraction of tasks, say 90% is also acceptable. In another typical scenario users want to see a small fraction of the results, say 10%, in a shortest possible time to verify the application setup. A sustained delivery of the results which allows to track application progress is often very important.

Users are interested in minimizing the time to produce the application output which may require completion of a set of tasks. Therefore, users seek to minimize the *makespan*, defined as a time to complete a given set of user tasks, which is a typical performance-related metric in scheduling research [159]. However, in complex and chaotic distributed systems such as grids the mastering of the *variation* is equally important. Reliable estimates of the makespan and decreasing its variance impacts directly the productivity of the end users and allow for planning of the end-user activities on the grid: “[...] *there is a socio-political problem that results when a user has an application with varying performance. It has been our experience that users want not only fast execution times from their applications, but predictable behavior, and would be willing to sacrifice performance in order to have reliable run times*” [162]. Therefore, it is interesting to describe task processing times in terms of probability distributions and analyze their properties such as average value and variance.

Another key quality metric is the *reliability*, i.e. failure rate, perceived by the user. Grids are built of unreliable elements and failures are frequent, however, the resources in grids are largely redundant and may be used to apply failure-recovery strategies such as rescheduling of failed jobs. If failures may not be correctly handled by the system in an automatic way then they are perceived as errors at the user level and require manual user intervention. The reliability at the user level is not synonymous with the system reliability: if the system is capable of taking corrective actions automatically (e.g. automatic resubmission of jobs) then the reliability perceived by the user may still be good (although at an expense of degraded performance). Efficient strategy of handling failures very often requires the inside application knowledge.

1.4.2 Infrastructure interoperability at the application level

One important aspect of large-scale scientific computing is the use of resources across existing computing infrastructures. This requirement stems from:

- a need for qualitative resource selection at the application level to manage cost and efficiency (section 1.1.8),
- a need for different working user environments (e.g. more efficient support for the application development process),
- a need for improving dependability of locally available resource in case of critical applications,
- a need for scaling out beyond locally available resources (there is no a single grid or system that unifies all possible resources).

Grids are only one of many computing environments for the everyday work of scientists. Development and testing of scientific applications is an intrinsic part of many types of research activities such as data analysis in particle physics. A typical method is to develop and debug a data processing algorithm locally, then to test it on a larger scale using on-site computing resources and locally available data before harnessing the full computational power of a grid. The transition between local and grid environments may also happen in another direction as the research process may involve multiple development phases and cycles.

Grids may be used for improving dependability of locally available resources and complementing them when peak demands arise. For example at ITU RRC06 the EGEE Grid delivered dependable peak capacity to an organization which normally does not require a large permanent computing infrastructure. In such a context grids may be seen as a competitive alternative to traditional procurement of resources.

Some user communities have access to local resources which are not part of a grid. Lack of human resources which are required to setup and maintain grid site services is one of the reasons of the conservative policy in embracing grid technology by many site administrators. It is clear that if users had a possibility to easily mix local resources with the grid ones it would be beneficial not only for them, but also, in the long term, for the whole grid community.

1.4.3 Application specific scheduling and coordination

Complex application coordination patterns, including application-aware scheduling, are not directly supported across multiple distributed computing environments. In case of large grid infrastructures such as EGEE, coordination layers are often considered application-specific and receive little support in the middleware. However, efficient coordination mechanisms are often the key element of enabling applications in the grids. Low-level communication technologies such as MPI used in MPI-BLAST inevitably incur a lot of development effort and expertise. Application porting *from scratch* is not cost-effective and may not be afforded by all communities. Therefore, bridging this coordination gap is an important requirement for many user communities.

1.5 The research objectives and roadmap

MTA applications are increasingly important and represent a majority of applications running nowadays in production grids. Understanding how the grid dynamics influence the processing of MTA applications is a challenging task. It includes understanding of spatial and temporal complexities of a large, heterogeneous, decentralized computing system. We attempt to grasp a fraction of this reality to extract fundamental patterns and processes from a largely chaotic system and ultimately to turn them into design patterns, methods and tools to boost productivity and research capabilities of the user communities.

The central research hypothesis may be formulated as follows: *cost-effective strategies for mastering dynamics in large-scale grids in the context of task processing of MTA*

applications may be efficiently addressed at a user level. Detailed research objectives are as follows:

1. quantitative explanation why late binding is advantageous as compared to the early binding by creating a task processing model which describes both binding methods,
2. identification of key mathematical properties which affect the dynamics of the late-binding method by characterizing the task processing makespan in terms of probability distributions and their parameters,
3. characterization and analysis of spatial and temporal dynamics in large grids based on available monitoring data,
4. development of an efficient strategy for MTA applications based on a late binding scheduler and a high-level task management interface,
5. demonstration of specific characteristics and properties of the strategy applied to selected scientific fields with particular emphasis on Capability and Capacity Computing.

The boundary conditions for our study are set by its *utility in the existing computing environments* – we explicitly choose this constraint to increase the impact of our work on real applications. We would have had more freedom if we conducted our research in a small experimental testbed, however, at a risk of reducing its significance. Hence, in our particular focus is the EGEE Grid - the largest scientific computing infrastructure built to date. In Chapter 2 we characterize the EGEE Grid as a distributed infrastructure and job processing system and we analyze the dynamics present in large grids.

One particular processing pattern of our interest is the late binding. Late-binding systems are sometimes praised by the end users to improve Quality of Service of task processing. In Chapter 3 we *quantitatively explain why late binding is advantageous* compared with standard job submission methods based on early binding, and what are the *key mathematical properties which affect the dynamics of the late-binding method*. The fundamental question is achieving acceptable makespans and reducing processing variability seen by the user on a system which is inherently variable, unstable and unreliable. From a general perspective this problem is similar to the provision of Quality of Service in the public TCP/IP networks, which by themselves are heterogeneous and do not provide unified QoS.

In Chapter 4 we describe a *strategy for efficient support of MTA applications* in large, distributed computing environments, including any combination of local batch farms, specialized clusters, clouds or grids. This strategy is implemented as a User-level Overlay: a set of tools for easy access to and selection of distributed resources and improved scheduling based on late binding. We demonstrate how this general strategy may be applied to a large, distributed system which is composed of many elements under separate administrative domains, and, in particular, independent job queues.

Successful use of the ideas and tools (often by independent teams and in diverse application areas) provides the best verification of the impact of our work. In Chapter 5 we

point out *specific characteristics and properties of our User-level Overlay system applied in selected scientific fields*. In particular we show how our strategy allows to achieve high task distribution efficiency and reduced operational efforts for large computational campaigns. We also demonstrate how the User-level Overlay makes it possible to obtain partial results in a predictable way, including quasi-interactive feedback at runtime.

A more detailed analysis of the system follows from a capability computing (high-performance) case study in Chapter 6, and a capacity computing (high-throughput) case study in Chapter 7. We conclude our work in Chapter 8.

CHAPTER 2

Dynamics of large computing grids

If it can't be expressed in figures, it
is not science; it is opinion.

Lazarus Long

In this Chapter¹ we present a descriptive analysis of spatial and temporal dynamics in large, production-grade grid infrastructures. The analysis is focused on computing services. It is based on publically available data and various monitoring sources in the EGEE Grid, and is complemented by the data from our specific observations of the system behavior. The aim of our analysis is to gain a better understanding how large grids work, what are the forces driving their development and functioning, and how they impact the end users and application communities. We use the EGEE Grid – the largest grid in operation today – as a representative example of a distributed system for processing of scientific tasks which exposes complex and dynamic behavior.

2.1 EGEE – world's largest computing and data Grid

The EGEE Grid² is a globally distributed system for large-scale processing and data storage. At present it consists of around 300 sites in 60 countries and supports more than 10^5 jobs a day. It offers more than 10^5 CPU cores and 20 PB of storage to 10^4 users in nearly 200 Virtual Organizations (VOs)³. EGEE is a multidisciplinary Grid,

¹A part of the results described in this Chapter formed the basis of the following paper: C. Germain-Renaud, C. Loomis, J. Mościcki, and R. Texier. Scheduling for responsive Grids. *J. Grid Computing*, 6:15–27, 2008.

²Enabling Grid for E-sciencE (EGEE), <http://www.eu-egee.org>

³Source: <http://technical.eu-egee.org>.

supporting users in both academia and business, in many areas of physics, biomedical applications, theoretical fundamental research and earth sciences, as summarized in Tab. 2.1. The largest user communities come from the High Energy Physics, and in particular the experiments active at the Large Hadron Collider (LHC) at CERN⁴.

Development of a large-scale, production grid is a long-term process which involves funding entities (e.g. national research councils or international agencies), resource providers (e.g. computing centers), resource consumers (e.g. user communities), hosting entities (e.g. large scientific laboratories) and external entities (e.g. cross-domain scientific or infrastructure projects). The process is complex and based on negotiation due to the distributed funding model and complicated relationships between involved parties. The EGEE Grid development started in 2002 as R&D at the Data Grid project, and was carried through several implementation and deployment phases of EGEE project to the currently ongoing consolidation phase and transition to European Grid Initiative (EGI). EGEE Grid emerged as a federation of three large infrastructures: WLCG which uses the gLite [113] middleware, OSG [155] which uses the Globus [66] middleware and NDGF which uses the ARC [58] middleware.

Application domain	Active VOs	Users
High-Energy Physics	43	5205
Infrastructure	28	2535
Multidisciplinary VOs	32	1825
Life Sciences	14	571
Computational Chemistry	4	448
Astronomy, Astrophysics and Astro-Particle Physics	21	362
Earth Sciences	11	325
Computer Science and Mathematics	6	28
Fusion	2	7
Others	36	1878
Total	197	13184

Table 2.1: Virtual Organizations and application domains in the EGEE Grid. Users may belong to multiple VOs and thus may be double counted. Data source: CIC-OP.

2.1.1 Middleware services

EGEE Grid is a distributed federation of computing, storage, information and monitoring services [145].

Information services aggregate static and dynamic information about the status of the Grid and provide it to other services. They allow discovering of types of available services and querying for the characteristics of each service. Information services also handle the authorization data which defines VOs allowed to use the services.

⁴General updated information on the LHC programme is available on CERN web site at <http://www.cern.ch>

Computing services include *Computing Elements* (CEs) which represent resources within a single cluster or batch farm. CE is the smallest unit for resource brokering and selection, which is realized in one of two models. Central resource brokering model, which is a default mode in gLite, uses *Workload Management System* (WMS) services, which perform job-resource matchmaking and routing of jobs to appropriate CEs. Client-side resource brokering is more common with new gLite CREAM CE services and also in NDGF with ARC middleware and OSG with globus/GRAM protocol.

Storage services include Storage Elements (SEs) and File Catalogues (FCs) which allow file-based access to data, file transfer and handling of multiple file replicas.

Existing monitoring projects and services allow to analyze the EGEE system at various levels. In particular, the GridObservatory [119] project is dedicated to providing datasets of traces reported by gLite services for scientific research⁵. Tab. 2.2 provides a summary of all data sources used in this Chapter.

Name	Full name and reference
CIC-OP	CIC Operations Portal http://cic.gridops.org/index.php?section=home&page=volist
GO	Grid Observatory http://www.grid-observatory.org
Gstat	GStat http://gstat-prod.cern.ch/gstat/geo/openlayers
GOCDB	Grid Operations Center Database https://goc.gridops.org
CESGA	EGEE Accounting Portal http://www3.egee.cesga.es/gridsite/accounting/CESGA/egee_view.php
GridView	Monitoring and Visualization Tool for LCG http://gridview.cern.ch/GRIDVIEW/job_index.php

Table 2.2: List of EGEE Monitoring Services used as data sources for analysis of EGEE Grid dynamics.

2.1.2 Conflicting forces: resource providers and consumers

Grid resources, including networks, are distributed among different administrative domains and controlled by different local policies. VOs bring subsets of resources into single virtual administration domains. This should allow setting of global access and resource sharing policies for the VO members, but in practice it is difficult to achieve due to incomplete and complex interfaces between grid services and local fabric layers.

The ultimate control of the resources is retained by the resource providers which define local policies to satisfy the demands of grid and local users – computing centers typically provide an important part of their resources to non-grid communities. For this reason system administrators in grid sites prefer to use well-established technologies, such as batch systems, to control the usage of their computing resources. Therefore,

⁵The Grid Observatory is a part of the EGEE-III EU project INFSO-RI-222667.

achieving additional capabilities above the ones already provided by the local systems is difficult.

Trust relationships are a delicate issue and resource providers require the accountability and traceability of the user activities as a part of the grid-participation agreement. This allows to exclude the misbehaving users from accessing the site services. The traceability requirements may be waived in well-understood, exceptional cases for restricted access of few power users who are mandated by the VO to submit jobs on behalf of other users. This is a case, for example, in HEP data production systems where centrally managed jobs are typically mapped to few management accounts. For some purposes, such as the data management, large VOs are also capable of negotiating the setup of VO-specific services, which are placed under direct VO control and deployed on dedicated physical nodes in grid sites. However, for smaller communities or individual scientists running their applications, such special arrangements are often impossible.

2.2 Grid as an infrastructure

2.2.1 Grid structures

The EGEE Grid infrastructure is heterogeneous and it integrates sites of varying sizes, computing capacity and internal structure, as shown in Fig. 2.1. The number of computing nodes per site spans four orders of magnitude. Nearly 5% of sites consist of above 2000 nodes with few exceptionally large sites with more than 10,000 computing nodes. On the other hand nearly 10% of tiny sites consists of less than 10 computing nodes and 40% of sites are below 100 nodes. Majority of the sites use up-to-date, mainstream hardware technologies as shown by the clear peak in the distribution of average computing node capacity (estimation of processing capacity of installed hardware is based on standard benchmarks, such as SPECINT⁶). However, tails on both ends of the distribution show the capacity variation of one order of magnitude for more than 10% of the sites. The absolute majority of sites expose one CE service, the remaining 10% of sites expose up to 8 CEs and, in extreme cases, even up to 25 CEs. A large number of CEs per site may be an indication that a site integrates several computing clusters for grid use, or that more CEs are needed to increase scalability of the service.

The EGEE Grid is structured in few ways, including high-level structures derived from computing models of heavy VOs, such as the MONARC [8] model, which is a hierarchical computing model for data processing at the LHC. The implementation details of the model differ for each LHC experiment but the hierarchical organization, shown in Fig. 2.2, is in common. In this model, sites are classified into Tiers, according to their size, provided level of service and operational support [26]. Tier0 is the data source for LHC experiments. Sites in Tier1 are large computing and storage farms which perform heavy data reprocessing. Sites in Tier2, typically research labs and universities, are medium-sized and support simulation and further stages of data analysis. Tier3 sites are small clusters, primarily used for end-user data analysis. The hierarchical model also applies to networking. Sites in Tier1 may be connected with high-speed, private

⁶Standard Performance Evaluation Corporation, <http://www.spec.org>.

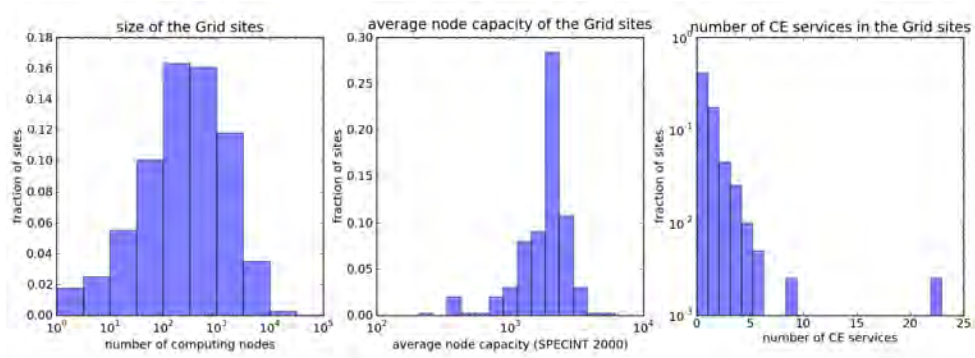


Figure 2.1: Histograms of basic characteristics of computing sites in EGEE Grid: number of worker nodes per site, average computing capacity of worker nodes and number of CE services per site. Y-axis shows a fraction of sites with a given value of X. Data source: GStat (gridmap).

optical networks [29], whereas most of other sites are connected using a public Internet infrastructure. Currently EGEE infrastructure includes 13 Tier1 sites (around 3% of all sites) and 123 Tier2 sites (around 30% of all sites).

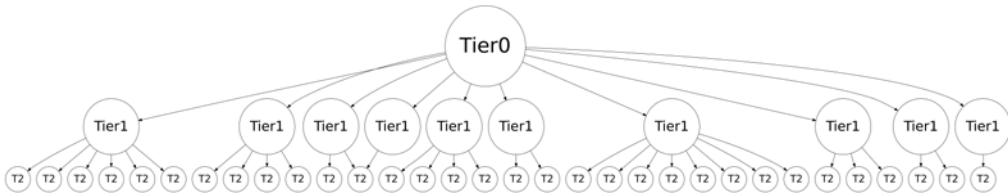


Figure 2.2: Tier model of the Grid is a basis of functioning for heavy user communities at the LHC at CERN.

Another structure is defined in the context of the development of the infrastructure itself. The EGEE sites are grouped into regions defined by Regional Operations Centers (ROCs) which provide operational support to the EGEE Grid sites. ROCs assist the sites in resolving operational problems, coordinate the infrastructure support, deploy middleware releases and provide testing and operational procedures in their region. Fig. 2.3 is a snapshot of the GridMap monitoring page which shows the structure of the regions, where the areas of regions and corresponding sites are proportional to the total computing capacity measured in SPECINT 2000.

2.2.2 Operational dynamics

Large-scale grids are living structures, where availability of resources changes dynamically due to infrastructure maintenance, hardware and software upgrades and random,

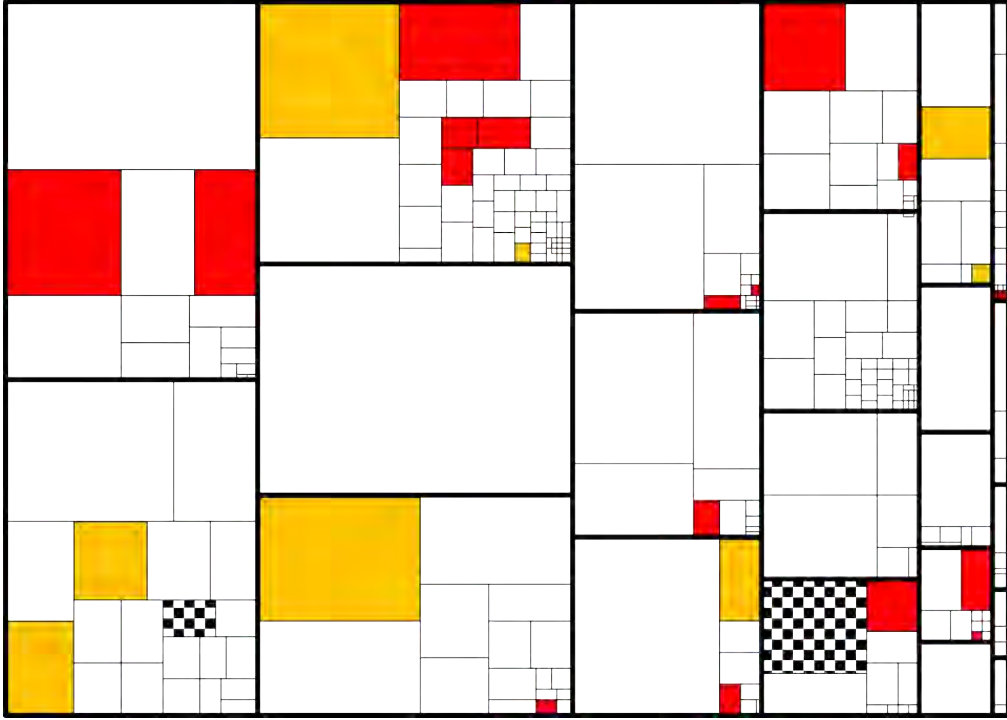


Figure 2.3: Snapshot of GridMap monitoring page showing the structure and relative size of EGEE Grid operation regions and associated sites. Sites are represented as rectangles within regions. The area of rectangles corresponds to the total computing capacity measured in SPECINT 2000. The map uses color-coding to show availability of services at the moment when the snapshot was taken. Dark (red) - site unavailable, light (yellow) - degraded operation, checkers pattern - maintenance.

unscheduled events. In the EGEE Grid, site availability is tracked by the operation centers, where sites may declare scheduled (planned) and unscheduled (emergency) downtime. During 7 years of operation, nearly 15000 site interventions (64% of planned and 36% unplanned) were recorded. From the point of view of the users, site interventions look as system failures, or service interruptions, which lead to temporary degradation of system performance. Hence, site interventions are an important source of observable instabilities in computing grids.

The weekly number of site interventions has been consistently increasing in the last years. It reflects the growth of the infrastructure and it currently oscillates around 100, as shown in Fig. 2.4 (c). Typical duration of interventions is below 3 hours, but as show in Fig. 2.4 (a), may take up to a week per site. Sites also tend to declare downtime in the unit of full days, what is visible as spikes in the histogram. The weekly accumulated downtime across all sites is above 100 days (Fig. 2.4 (d)), what indicates

that availability of resources is dynamically changing at rather large scale. A clearly visible plateau in Fig. 2.4 (d) reflects the efficiency improvement of operating a large grid: the operational experience of the support teams grows in time but it is balanced by the increased complexity of supporting of a larger system.

How much in advance the users know about the resources becoming unavailable? The notice period, calculated as a difference between the declared start of the intervention and the time of registration of the intervention in the operations database, is shown in Fig. 2.4 (b). Large majority of interventions is registered very late (close to the event time) or even after the event actually happened (in this case the notice period is negative). A majority of these interventions correspond unscheduled site downtimes due to failures, urgent security patches etc. The tails of up to 1 month in both directions indicate that, on one hand some interventions are planned well in advance, and on the other hand, that quite frequently the operational information does not propagate very efficiently in a large grid collaboration.

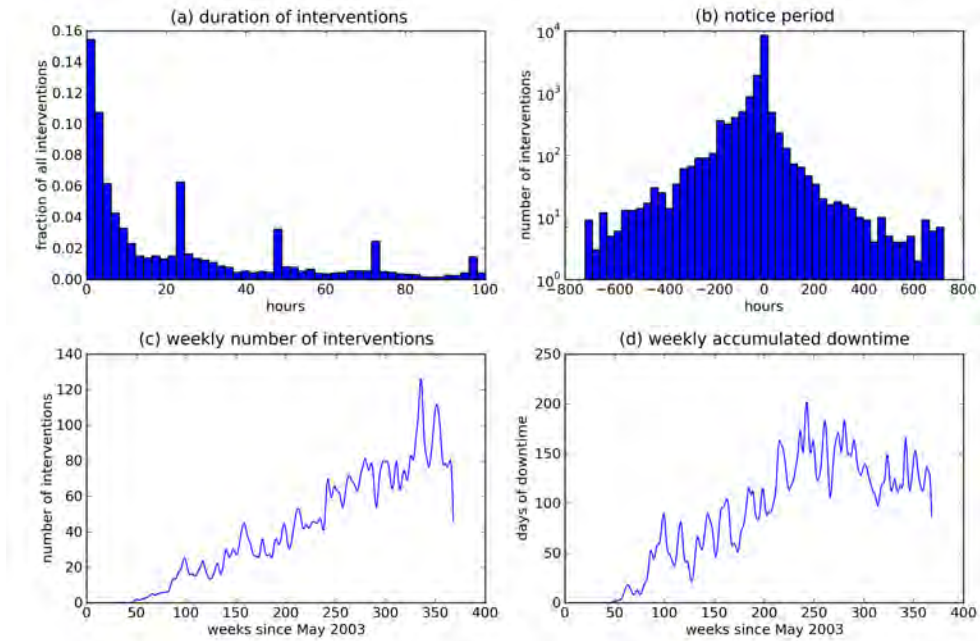


Figure 2.4: Operational parameters of the EGEE Grid: (a) frequency histogram of site unavailability duration, (b) histogram of notification delays, (c) evolution of weekly number of interventions in time and (d) weekly accumulated downtime for all sites. Plots (c) and (d) smoothed using a moving window with Gaussian smoothing. Data source: GStat and operations database.

2.2.3 Middleware and application deployment

Certification, quality assurance and testing of new middleware releases is performed centrally [69]. However, maintenance and deployment of the middleware in the EGEE Grid is not centralized because resource providers have local schedules and obligations towards their local (non-grid) communities. This affects the maintenance of existing grid protocols and services as well as the introduction of new ones. The middleware changes are propagated very slowly in a large grid and often take several months or years to complete. Fig. 2.5 shows how gLite middleware installations in the EGEE Grid sites evolve in time. Different curves correspond to different versions of the middleware and show the number of sites with a particular version installed. Multiple middleware versions exist in the Grid at any moment. The lifetime of a particular middleware version spans several years as many grid sites are free to arbitrarily delay the adoption of changes. Therefore, the infrastructure (thus the middleware) evolves quasi-independently of the evolution of the application functionalities. Moreover, efficient evolution of grid protocols in the current EGEE Grid is very difficult and mostly restricted by backwards compatibility. This makes it more difficult to improve existing functionalities in the middleware, especially if they are heavily used.

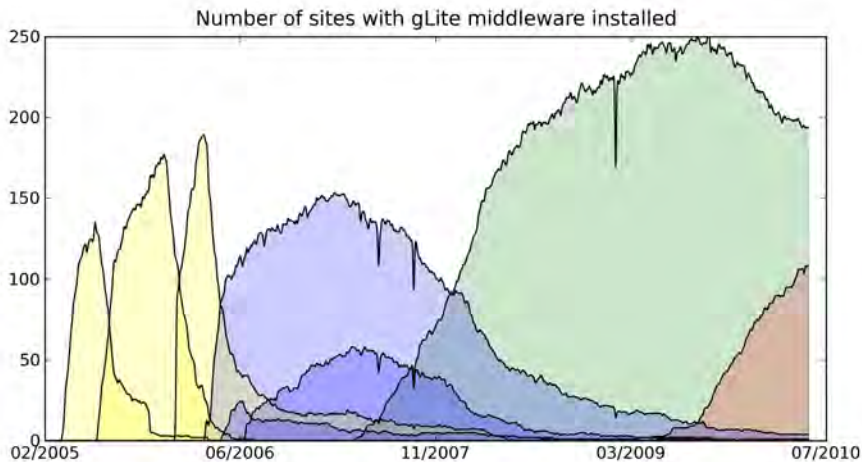


Figure 2.5: Evolution of deployment of gLite middleware in the EGEE Grid. The number of sites with a given version of gLite middleware installed is shown for all major middleware releases. Plots are semi-transparent and overlaid. Gaussian smoothing was applied with a 5-day moving window. Data source: GOCDB

The installation of application software is decoupled from middleware deployment. A standard approach is based on static distribution of software by the VO managers. The EGEE Grid Information Service may be queried for so-called software tags, which are published at the time of software installation and may be subsequently used for resource brokering and matchmaking of jobs. This centralized approach is not flexible

when users require new or locally modified software to be deployed in grid sites. It is also quite difficult and time consuming to achieve consistency of installed application versions across many sites. Therefore, sometimes users distribute application software dynamically, using storage elements (part of data management system) or using job file sandboxes. The drawback of this approach is that application distribution knowledge is not integrated in the Grid Information Service. Also, the use of network resources may be suboptimal if multiple, redundant transfers of software packages are needed, especially for large software packages. Sometimes software is also distributed out-of-the-grid, using HTTP servers (curl/wget) or even version control systems such as SVN/CVS to checkout latest software updates at runtime (this is the case of the ATLAS PANDA pilot-job framework).

2.3 Grid as a task processing system

2.3.1 Scheduling model

Similarly to classical batch systems, the bulk of EGEE Grid infrastructure is designed for optimizing of the throughput of long, non-interactive jobs. Access to computational resources is realized by the Workload Management System (WMS) using distributed queues in a multi-layer, multi-point, early-binding scheduling architecture presented in Fig. 2.6.

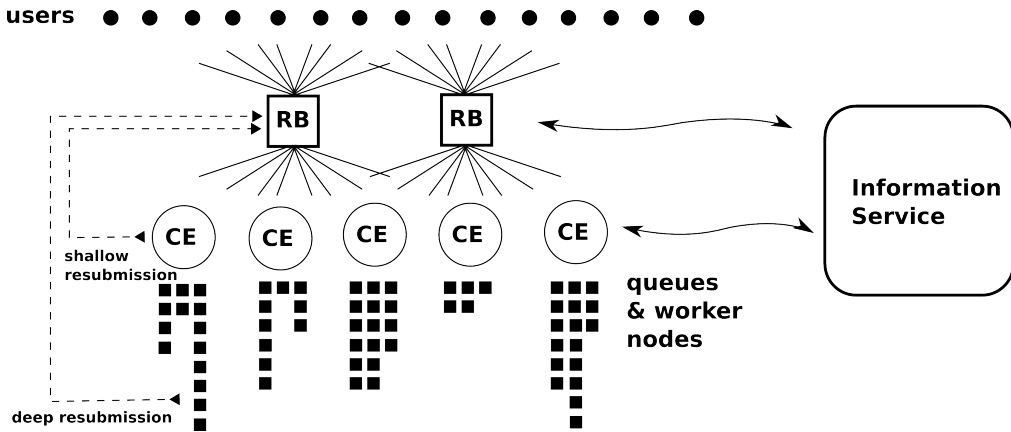


Figure 2.6: Workload Management System and job scheduling model in EGEE Grid. Storage services are deliberately omitted.

User jobs pass through three layers of computing services and associated job queues. The jobs are first transferred to Resource Brokers⁷ (RBs) which select suitable Computing Elements (CEs) based on the data acquired from the Information Service. CEs

⁷In this work we use the terms RB and WMS interchangeably to refer to the job brokering service in EGEE Grid. Historically, RB was used as a name for the first-generation of Grid middleware, while in more recent versions of gLite, WMS was introduced as a name for new service implementation. For

represent computing clusters within sites, and act as a common interface to local batch systems which queue jobs to be run on Worker Nodes (WN). The Information Service keeps a periodically updated status of grid resources and services. The scheduling is multi-point as different jobs of a single user may be routed to the same CE passing through different RBs, and conversely, may be routed to different CEs passing through the same RB. Both CEs and RBs receive load from multiple sources.

The model is based on early binding: users typically define and split the work before submitting the jobs and retrieve the results for successfully completed jobs by polling the WMS. Thus, one job carries one task which is specified when the job is submitted.

One distinctive feature of this scheduling architecture is the presence of resubmission loops. *Shallow resubmission* of jobs is done automatically by the system if jobs may not reach the batch system queue. Shallow resubmission occurs before the job execution on the worker node and may happen due to a number of reasons, such as configuration or operational problems of the computing elements, matchmaking problems due to outdated information on available CE resources, or authentication and authorization problems. *Deep resubmission* of jobs is done automatically by the system in case of problems with job execution and it is typically due to application-specific errors, such as data access problems, bugs in the application code, or runtime faults. Both types of resubmissions are configurable, and if the retry count is reached, the job is reported to the user as failed.

2.3.2 Spatial workload distribution

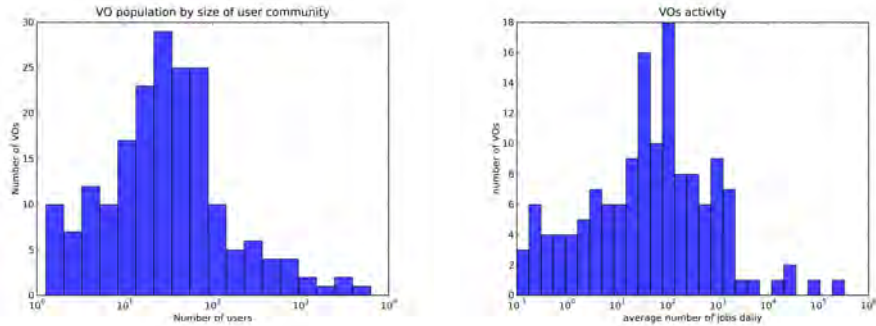
We expect that members of the same user community, thus of the same VO, generate similar workload patterns as they run similar applications and use similar task submission and processing tools. This should be true for spatial distribution of workload, for example, groups of scientists analyzing a single dataset which is stored in few grid sites. This should also be true for temporal use patterns, for example, when the task processing activities increase before an important community event such as a conference.

Majority of VOs comprise between 10 and 100 users as shown in Fig. 2.7(a). Few individual VOs, mainly LHC experiments, comprise more than 1000 users. The distribution of VO population differs from the distribution of VO activity, shown as an average, daily number of submitted jobs in Fig. 2.7(b). Again, few LHC VOs stand out with several tens, or even several hundreds of thousand jobs a day. The activity of a large fraction of VOs is below 100 jobs a day. LHC VOs tend to generate constant background of job traffic due to heavy, high-throughput reprocessing of data while for many smaller VOs job traffic is more erratic and irregular.

Spatial distribution of workload in the EGEE Grid may be represented as a graph, where vertices correspond to CEs and RBs, and edges represent job transfers from RBs to CEs. *Job traffic* is defined as a number of jobs flowing through a vertex or an edge. To construct workload distribution graphs we use a monitoring sample which covers a week of EGEE Grid activity in all VOs in April 2010⁸. This sample is representative

the purposes of this analysis we may safely assume that the goals and scope of RB and WMS services are the same.

⁸RTM data provided by Grid Observatory.



(a) VO population by size of user communities in EGEE Grid. Data source: CIC-OP
 (b) VO activity by number of submitted jobs (04.2009-04.2010). Data source: CESGA

Figure 2.7: Basic characteristics of the EGEE VOs.

for the typical workloads generated by all EGEE VOs: the distribution of the number jobs per VO in the sample is similar to the one recorded for a full year (shown in Fig. 2.7(b)). The monitoring sample contains some 2×10^6 job records covering 82 RBs and 614 CEs. The graph is sparse (density=0.035), however, it is difficult to visualize due to a large number of edges (8450). On the other hand, the distribution of edge traffic, shown in Fig. 2.8(a), indicates that the job traffic is small in a large fraction of edges. Therefore, it is possible to simplify the graph with little information loss about the job traffic. Simplification with a cutoff value v consists of removing all edges with job traffic smaller than v . For example, for $v = 1$ the simplified graph contains only 75% of edges but it represents 99.5% jobs, and for $v = 15$ graph contains only 35% of edges representing 95% jobs. The number of pruned edges, shown in Fig. 2.8(b), depends on the distribution of vertex degrees and the distribution of edge traffic.

Workload distribution graph⁹ for non-LHC VOs is presented in Fig. 2.9. The job flow reveals small clusters of RBs and CEs with high internal job traffic (A,B) and even dedicated RB-CE pairs (C). Opposite pattern (D) is also visible, where one high-degree RB generates low job traffic to many CEs. Another structure is visible as a shaded convex hull (E) which includes all RBs and CEs which are placed in one country domain.

Certain VOs may generate workload in very particular patterns. For example, in the OPS VO which performs infrastructure testing, one central RB service sends small number of jobs to more than 500 CEs (Fig. 2.10). Smaller “satellite” RBs send jobs to selected CE subsets and contribute to less than 50% of the total job traffic.

2.3.3 Information Service

Reliability and efficiency of task processing in the EGEE Grid strongly depends on quality of data provided by the Information Service, which is assumed to be accurate

⁹Graph analysis and visualization was performed using Guess [7].

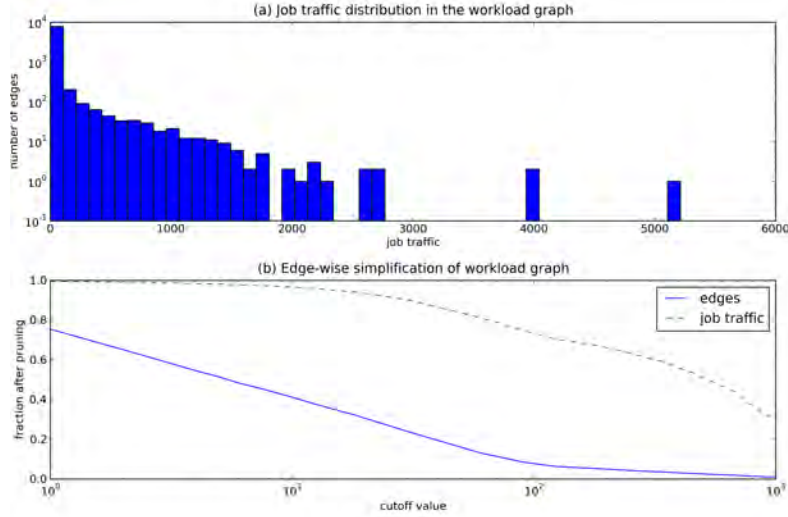


Figure 2.8: The workload distribution histogram (top) shows the edge traffic (number of jobs flowing in the edges). The edge-wise simplification results are shown in the bottom plot, as a fraction of the number of edges and total job traffic in the graph after pruning with a given cutoff value. For small cutoff values a significant number of the edges is pruned, however, the remaining, simplified graph still represents large majority of jobs observed in the sample.

and timely updated. In practice the Information Service is known to have significant delays in notifications due to its architecture [25] and it suffers from delays in information gathering and publishing and inaccurate data due to poor quality of implementation of middleware components.

Inaccurate or false data in the Information Service may create “strange” effects such as “black holes”. This may happen when a large number of jobs is sent to a CE in an instant of time in-between the CE status update in the Information Service, thus creating a sudden workload imbalance and increasing jobs waiting time. A misconfigured site, which is not properly black-listed, may be a reason for a consistent failure of all jobs sent to such a CE, which are then either reported as failed or resubmitted to other sites (thus increasing the job waiting time). Some resource selection parameters, such as runtime usage limits typical in batch systems (e.g. short and long queues) remain poorly implemented in the EGEE Grid because the mapping between the CE and the batch system is not consistently enforced, despite existing resource specification standards such as the GLUE [17] schema and JDL [113] attributes. Therefore, the traditional batch user logic – *send short jobs to fast queues and long jobs to slow queues* – is not readily available in the EGEE Grid.

Medium-term trends in the evolution of available resources, as published in the Information Service, are presented in Fig. 2.11 for three selected VOs. The data was

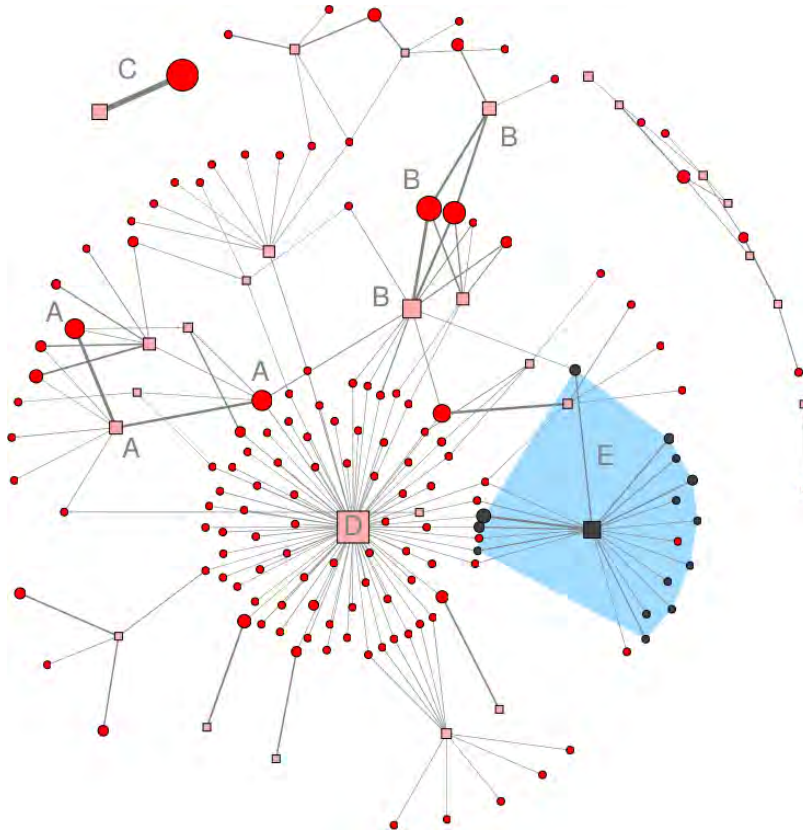


Figure 2.9: Workload distribution graph for non-LHC VOs, cutoff value $v = 60$, representing 46% of jobs. RBs are indicated as squares. CEs are indicated as circles. Visible structures: A - in2p3.fr, B - cnaf.it, C - csic.es, D - cern.ch. E - .nl. Shaded convex hull spans all services in the domain .nl (the corresponding vertices marked in dark gray). Vertex sizes and edge widths are scaled according to the number of jobs flowing through a given graph element.

collected by querying the Information Service using the `lcg-infosites` utility at regular intervals (every 10 minutes) for a period of seven months. Correlations of the number of available CPUs and CEs for different VOs are visible in both plots and may be due to a significant fraction of physical resources being shared between the VOs. Another interesting feature is visible for Geant4 VO around day 100: a relatively small perturbation in the number of available sites resulted in a large variation of the number of available CPUs. This may be explained by a CE with a large number of CPUs disappearing from the Information Service.

The EGEE Information Service exhibits large variations in stability of provided data. This is clearly visible in Fig. 2.12 which shows the histogram of daily and hourly

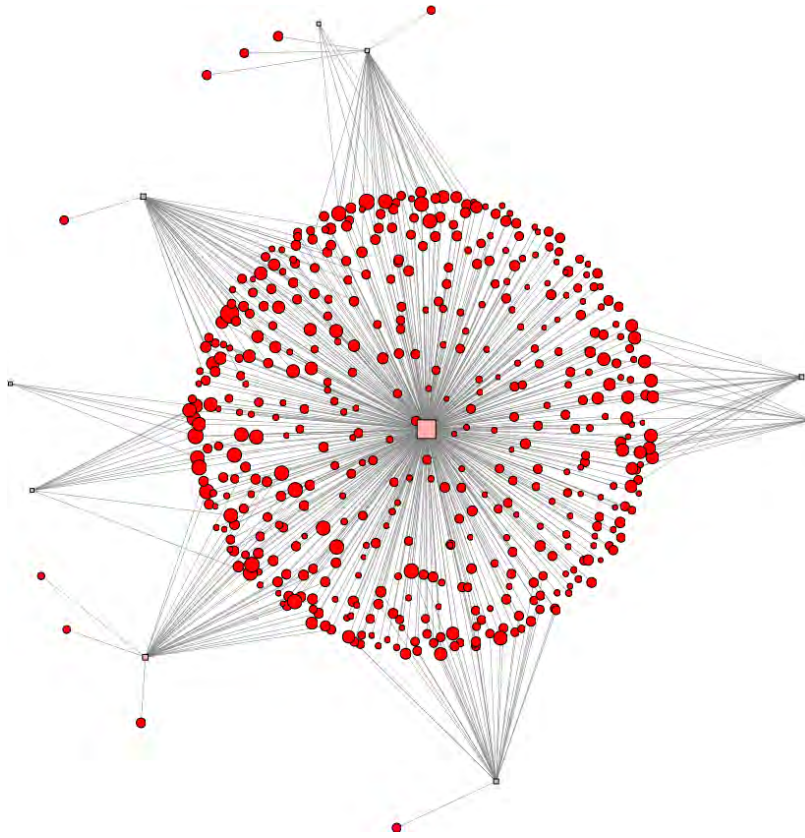


Figure 2.10: Workload distribution graph showing centralized job handling in OPS VO. RBs are indicated as squares. CEs are indicated as circles. The main RB is responsible for more than 50% of job transfers to more than 500 CEs. Vertex size and edge width are scaled according to the number of jobs flowing through a given graph element.

variations, calculated as a difference between maximum and minimum observed number of CEs in a given period. The hourly variations for a small Geant4 VO are much less pronounced than for a large Atlas VO. This may be related to poor scalability of the Information Service services. This effect is even more pronounced for daily variations: a large number of small perturbations in the Geant4 VO contrasts with a small number of large dips and spikes in Atlas VO.

Clearly, these short-term variations may not be fully accounted for the Grid maintenance and operations. They indicate limitations or bugs in the middleware itself. As an example, one particular problem of the Information Service was identified as an erroneous heartbeat implementation: a site which missed to send a heartbeat to the Information Service was automatically removed from the list of published sites. The system was later fixed to allow several heartbeats to be missed before the removal.



Figure 2.11: The number of Computing Elements and CPUs published in the EGEE Information Service. Data collected every 10 minutes by querying the Information Service. Day 0 corresponds to 23 Nov 2006, Gaussian smoothing with a 24 hours moving window.

A natural question is, if the problems with stability of provided information are intrinsic to grid architectures, or if they are only a “feature” of a particular implementation of the EGEE middleware. The answer is complex, but definitely the architecture of gLite middleware, and Information Service in particular, is prone to provide incorrect information, independently of the brokering model (centralized or client-based). In the case of centralized brokering, many RBs may be setup to handle jobs of a single VO for scalability and operational reasons, however, there are no clear mappings between a VO and a number, and location, of these RB services. In the case of client-side brokering, querying the Information Service for every submitted job would be very inefficient and would increase the job submission time. Therefore, the brokering information is typically cached locally to speed-up job submission, and the clients are responsible for refreshing of their caches. However, the local caches tend to be out-of-date and there are no consistent policies on how they are refreshed by the clients.

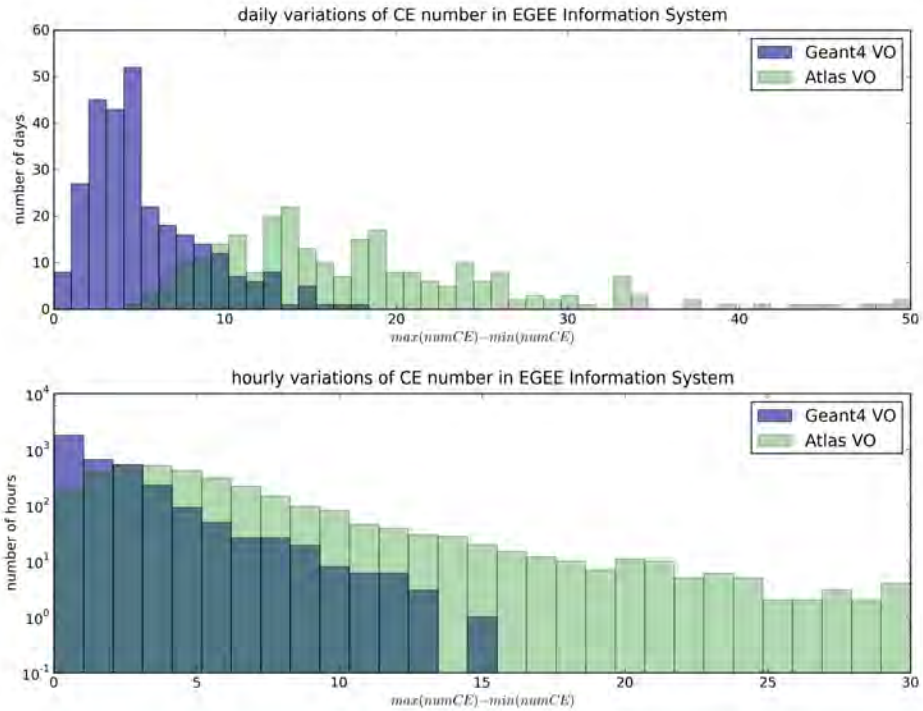


Figure 2.12: Histogram of the difference between highest and lowest number of CEs published in the EGEE Information Service in daily and hourly windows. The histogram bars are semi-transparent and overlaid. Data sample is the same as in Fig. 2.11.

2.3.4 Complexity of middleware architecture

The implementation of gLite middleware is based on a large number of middleware components from various providers, integrated into a complex architecture shown in Fig. 2.13. The detailed analysis and understanding of all components is not in the scope of our work, however, several features of the gLite architecture are critical for reliability and efficiency:

- large number of complex interactions between services implies less reliable operation of the system;
- alternative implementations of the CE service (LCG CE and CREAM CE) co-exist in the same system and increase the number of potential failure paths;
- Condor and Globus packages are integrated in several subsystems without a clear definition of component interfaces and responsibilities;

- multiple credentials management services at the user level (VOMS and MyProxy services) must interact with a complex proxy forwarding mechanisms in WMS, including identity switching at the CE level (with `glxec`).

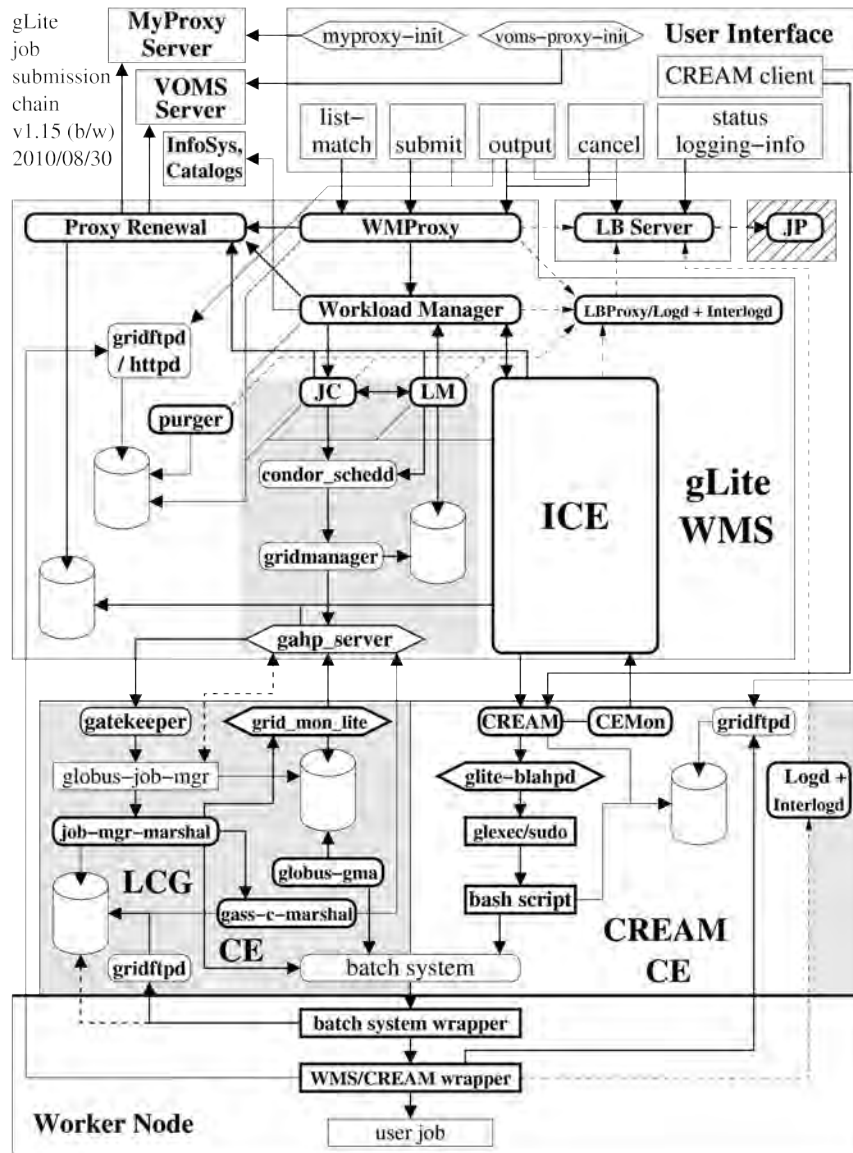


Figure 2.13: The actual gLite task processing architecture, showing the dependency and complexity of internal service interactions at WMS and CE levels. Diagram courtesy of M.Litmaath. Source: <http://cern.ch/twiki/bin/view/EGEE/EGEEgLiteJobSubmissionSchema>

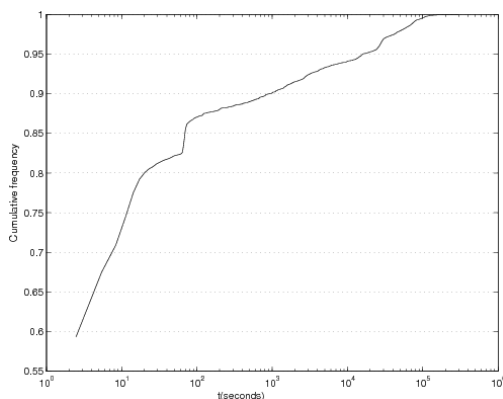


Figure 2.14: Cumulative distribution of execution times in Biomed VO.

Most of the complexity shown in Fig. 2.13 is not visible to the end users, however, it impacts the reliability and efficiency of processing of user tasks and is responsible for a perception of inadequate Quality of Service provided in grids. Common computing wisdom says that the central enemy of reliability is complexity.

2.3.5 Efficiency and reliability

Efficiency of task processing, defined as the ratio between the CPU time and the wall-clock time, strongly depends on applications. However, the submission and scheduling process before the job starts executing on a worker node is application-independent. Its efficiency may be studied as an intrinsic feature of a grid processing system. The time before the job reaches the worker node includes two main components: the middleware overhead s , which is the time for the job to pass from the user, through RB, to the CE, and the on-site queuing delay q , which is the time the job spends in a local batch queue (see Fig. 2.6). The turnaround time

$$m = s + q + t \quad (2.1)$$

is the total time from submission to notification that the job has completed, and it also includes t , which is the job wall-clock execution time.

Fig. 2.14 shows the distribution of execution times for 50×10^4 successful production jobs from 66 users in the Biomed VO, covering one year (October 2004 to October 2005). Due to limited monitoring capabilities at the time, the analysis is limited to one particular RB¹⁰.

The striking feature is the importance of short jobs: the 80% quantile is at 20 s. The second important point is the dispersion of t ; the mean is 2 s, but the standard deviation is of the order of 10^4 s. The very large fraction of extremely short jobs is partially due to the high usage of this particular broker by the EGEE Biomed VO. However, it was

¹⁰Monitoring traces for grid09.lal.in2p3.fr .

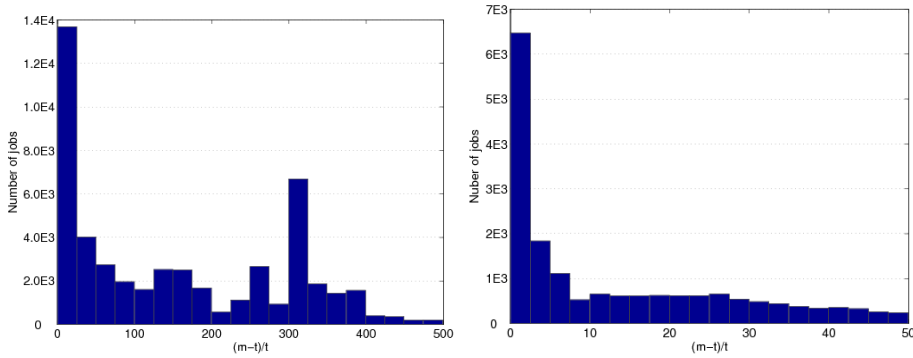


Figure 2.15: Distribution of the overhead factor. The left histogram is the distribution of the full sample, the right histogram is the distribution of the small overheads.

verified with other sources, that for more than 50% of the overall EGEE jobs in the same period, the execution time was less than 3 minutes.

Fig. 2.15 shows the distribution of the dimensionless *overhead factor*

$$o_r = (m - t)/t, \quad (2.2)$$

which is the overhead normalized by the execution time. Left histogram shows the distribution of the full sample: only 26% of the 53000 jobs are in the first bin, meaning that 74% of the jobs suffer an overhead factor larger than 25. Right histogram shows a close-up for small overheads and indicates that only 13% of the jobs experience an overhead factor lower than 2. It is clear that EGEE processing system is inefficient for shorter jobs.

The impact of the middleware and the queuing time on the global overhead is shown in Fig. 2.16, where distribution of q/s indicates that the queuing time is a significant component of the overhead. This behavior was exhibited at an early stage of EGEE usage, where the pressure on the resource was only starting to increase. Finally, the median queuing time is 91 seconds, and the median middleware overhead is 221 seconds.

Reliability of task processing experienced by the grid users is variable in time and is impacted by many factors, as the failures in the grid environment occur at application, system and network levels. A comprehensive overview of reliability and types of failures in the EGEE Grid is provided in [143] and probabilistic modelling of job failures and resubmission strategy is presented in [118]. The study on Grid reliability from end user's perspective [92] point at the *site autonomy* as one of the important sources of outages in resource and network elements in the Grid. At the same time job success rate strongly depends on the applications themselves. Despite application differences it is common for the user communities to experience varying job success rates as shown in Fig. 2.17.

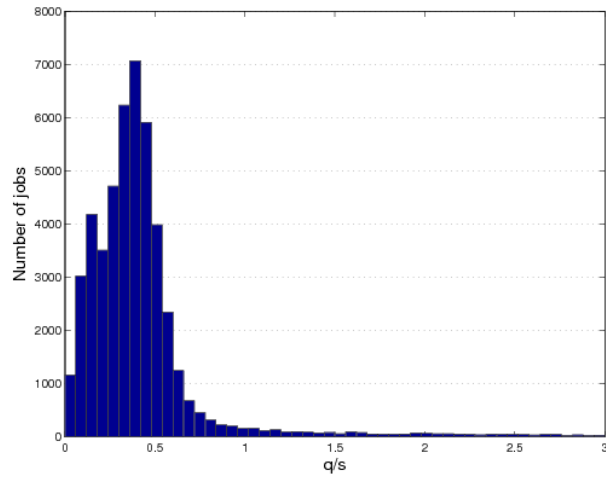


Figure 2.16: Distribution of a job queuing time as a fraction of middleware overhead, q/s .

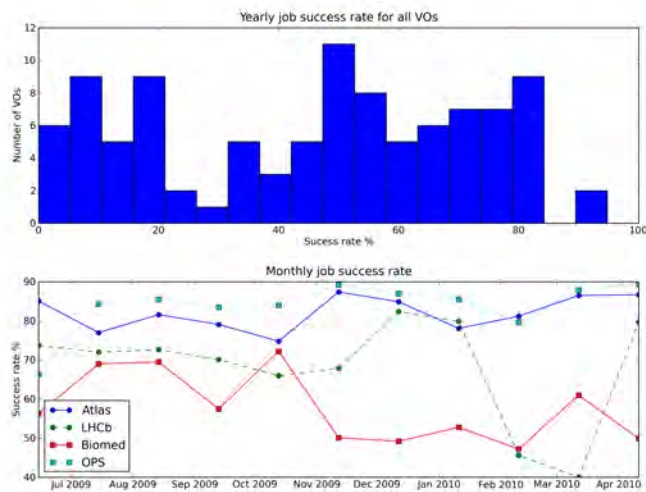


Figure 2.17: Distribution of job success rate from July 2009 to May 2010 and monthly reliability in this period for selected VOs. Data: GridView

2.4 Summary

Grids integrate computing resources on a large scale what provides a unique opportunity for many scientific communities. The large scale, however, comes with a cost: grids are structured in complex ways and are inherently dynamic in nature: resources are constantly reconfigured, added and removed. Long- and short-range changes in dynamics apply to use patterns, distribution of workload and processing reliability and efficiency. The delays in job execution, and the amount, quality and processing capacity of available computing resources at a given time, depend on the characteristics intrinsic to grids, such as scale and architecture of middleware services, and temporal activity of a global community of users.

CHAPTER 3

Analysis and modeling of task processing with late binding on the Grid

All models are wrong, but some are useful.

George E. P. Box

The limitations of the grid task processing model have forced application communities to seek for additional solutions to improve the Quality of Service. Amongst many different approaches, metascheduling with late binding has become one of the most successful task processing techniques. However, to date there is no strong theoretical explanation why this model is superior.

In this Chapter¹ we quantitatively explain why late binding is advantageous compared with standard job submission methods.

3.1 Introduction

Late-binding is a scheduling and coordination method, where work is assigned to a job at runtime rather than at submission time as shown in Fig. 3.1. In the case of standard, early-binding approach one job carries exactly one task which is specified at a time of job submission. Late binding scheduling involves three entities: a worker node (computing resource), a worker agent job (pilot) and an application work unit (task). The worker

¹The results described in this Chapter formed the basis of the following paper: J. Mościcki, M. Lamanna, M. Bubak, and P. Sliot. Processing moldable tasks on the Grid: late job binding with lightweight User-level Overlay. (*accepted for publication*) in *Future Generation Computer Systems*, 2011.

agent job is a placeholder for the actual application task, which is dynamically assigned to the worker node by an external scheduler. Hence the system does not involve process checkpointing and migration.

We assume that for every user there exists a scheduler with a private task queue which contains tasks belonging to that user only. We also assume that all worker agent jobs are submitted by and belong to the user. Thus the late-binding system is entirely self-contained from a single user perspective. The objective of the scheduler is to minimize the completion time (makespan) of a set of user tasks.

Analytical representations of such a task processing system are difficult because many of the parameters are random variables with empirical distributions for which clear models do not exist. In our approach we build a simple job-slot model to analyze the task processing QoS parameters of interest using Monte-Carlo simulation. Ultimately, the quantitative behavior of the system depends largely on specific distributions of system parameters. In this analysis the distribution of job waiting times in the EGEE Grid is the key component of the distribution of the task processing makespan.

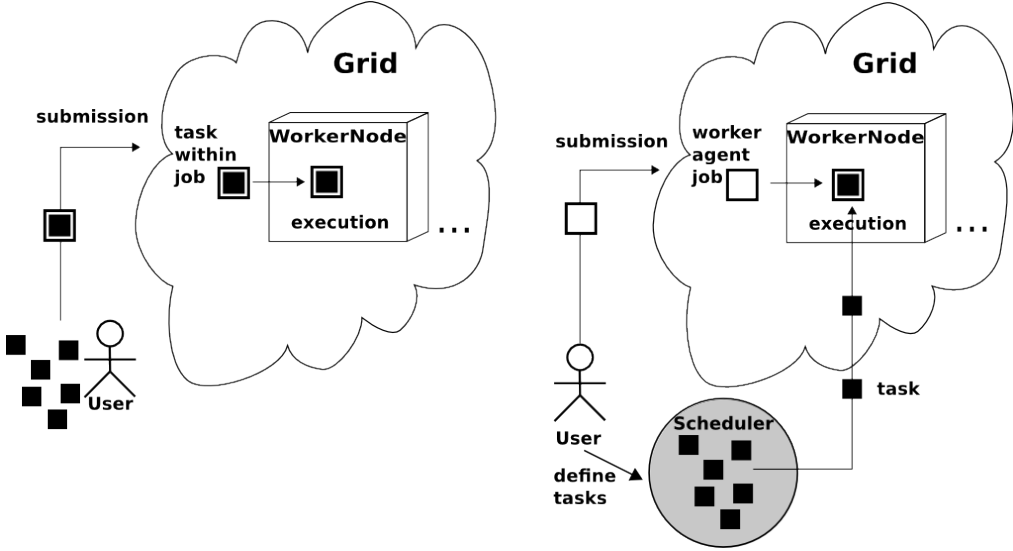


Figure 3.1: Early-binding (left): task is specified and assigned when a job is submitted. Late-binding (right): task is provided to a worker agent job at runtime.

3.2 Task processing model

To tackle the problem of comparing late- and early-binding scheduling methods we need a task processing model which is general enough to represent both approaches. We assume that a user workload is moldable and hence may be flexibly partitioned and executed as a set of tasks on a set of computing resources (processors or worker nodes).

Assuming no precedence relations between tasks (independent tasks) this model is also known as a Divisible Load model [28].

We look at the grid as a system of acquiring computing resources through job submission i.e. jobs allow the user to acquire computing slots for certain time. Application tasks are processed within computing slots which are characterized by:

- τ - time to acquire the resource;
- ν - usage time of the resource;
- $\varphi(t)$ - processing speed: amount of completed work (W) in time where $\varphi = \frac{dW}{dt} \geq 0$ and $\varphi(t) = 0$ for $t \notin [0, \nu]$

The work completed by K computing slots is given by:

$$W(t) = \sum_{i=1}^K W_i(t) = \sum_{i=1}^K \int_0^{t-\tau_i} \varphi_i(s - \tau_i) ds. \quad (3.1)$$

The makespan L is defined as the minimal time needed to complete the work W i.e. completion of all tasks defined by the user. Running tasks typically do not communicate and are independent². In case of early binding, tasks are assigned to jobs by a user at submission time, whereas, in the late binding case, a task is assigned and managed at runtime in a private queue by a User-level Overlay, and a user only submits worker agent jobs to acquire the computing slots.

Fig. 3.2 shows the workload distribution for K submitted jobs in the early-binding system G^3 and the user-level, late-binding system U^4 . Worker node i starts processing with the delay given by τ_i and stops processing after elapsed time ν_i . In late-binding model a crash of a worker node may result in premature termination of processing on that node. A smaller number of worker nodes, $k \leq K$, may be effectively used for processing. In the early-binding model worker nodes typically finish processing at different times as the load is not balanced at runtime. All $k = K$ submitted jobs are required to complete processing. Makespan L is determined by the completion of the last task.

Job queuing time τ is a random variable and corresponds to the time elapsed between job submission and the start of the execution on the worker node. Job execution time, ν , depends on the application-specific splitting of work, the number of available resources and their efficiency. Job execution time is bound by the maximum allowed time limit on the resource such as the batch queue limit, $\nu \leq \nu_{max}$. The distribution of ν_{max} is not uniform in grids and typically cannot be controlled. In this paper we assume that the average work per job fits in the allowed time on the worker node, such that $w = \frac{W}{K} \leq \nu_{max}$. This is a case of a user submitting a single bunch of jobs to be

²In principle, late binding is a more robust scheduling method also for tasks with precedence relations where output of one task may be needed as input of another task. However, for the sake of simplicity and clarity, this is not considered here.

³G stands for direct Grid submission.

⁴U stands for User-level overlay.

completed by a specified time rather than a continuous stream of jobs over a longer period. This applies to large fraction of end-user computational tasks, except for some applications of HTC-type which require continuous processing over long periods of time.

The processing speed φ_i is an application-specific capability of a worker node. It may be variable in time in the case of time-sharing with other users i.e. when other jobs are processed concurrently on the same worker node. It may depend on specific application requirements, such as the amount of local I/O, the actual load on the processor and runtime failures of the application. In case of the late-binding system φ_i depends on a number of hidden variables which relate to the implementation of scheduling algorithm, granularity of task splitting and properties of the communication network. The overheads of late-binding scheduling may result in processing gaps, where $\varphi_i = 0$.

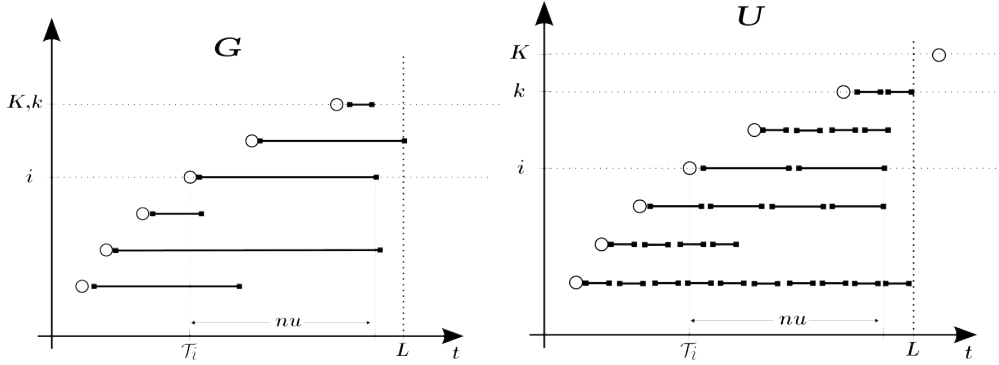


Figure 3.2: Workload distribution in early- and late-binding systems for K submitted jobs. Circles indicate worker nodes becoming available for processing. Line segments indicate worker nodes busy with application execution and a segment length corresponds to task duration.

3.3 Distribution of job queuing time

Job queuing time τ is an important component describing the variability in the largest, production-grade grid systems. We collected a large number of independent observations of single probe jobs submitted frequently and at regular time intervals to the EGEE Grid. The underlying assumption was that a single probe job causes negligible perturbations in the system i.e. it does not influence the overall state of the Grid in an observable way. Therefore, probe jobs have been designed for minimal resource and system footprint: very small, near zero execution time and minimal number and size of input and output files. Probe jobs could execute on *any* grid resource, as automatically decided by the Grid Workload Management System when no matchmaking constraints have been defined. By construction, the execution of a probe job could not fail. The experiment consisted of sending probe jobs every 30 minutes to the EGEE Grid during

the period of one year in the three Virtual Organizations, Geant4, ATLAS, LHCb which represent different use-patterns. Geant4 is a small VO using shared resources and low user activity. ATLAS is a large VO with a substantial amount of dedicated resources and high user activity. LHCb is a medium-size VO with high user activity which is managed by DIRAC [176], a community-specific workload management system.

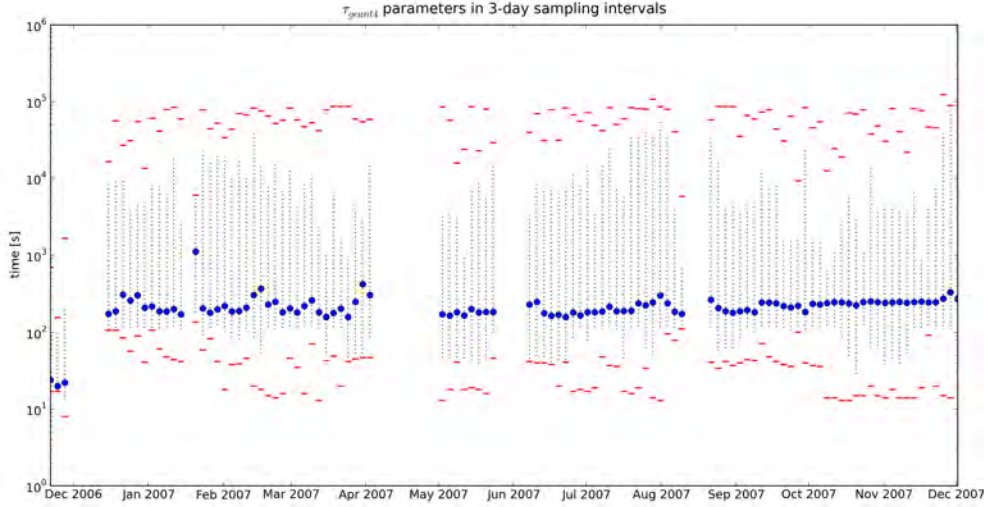


Figure 3.3: Evolution of τ_{Geant4} distribution in time, covering all 3-day sampling intervals. Y-axis shows τ in seconds. Vertical bars correspond to the 0.05-0.95 quantile range of τ distribution. Median is shown with blue circles. Extreme values are shown as horizontal red bars.

The job queuing time distributions $\tau_{geant4}, \tau_{ATLAS}, \tau_{lhcb}$, which correspond to the delay between job submission and beginning of job execution, are dispersed and heavily tailed. Fig. 3.3 shows the 0.05-quantile, 0.5 (median) and 0.95-quantile of τ_{geant4} of 3-day sampling intervals. The sampling interval was chosen such that we may reasonably assume that the process underlying the τ distribution is stationary within the sampling interval and that the sample is large enough. There are on average 120 measurements in each sample after the data cleanup to remove samples with less than 10 measurements. A shorter interval would lead to very small samples while with a longer interval the stationarity assumptions would become problematic. The same technique has been applied to τ_{ATLAS} and τ_{lhcb} .

If $\tau_q = t$ is a q -quantile of the distribution in the sample interval then a fraction q of all jobs submitted in that interval start no later than t . We consider $\tau_q = t$ at 0.95 confidence level if in 95% of sample intervals $\tau_q \leq t$. That is probability $P(\tau_q \leq t) = 0.95$.

In the Geant4 VO at 0.95 confidence level, the first 5% of jobs start in less than 2

$[s]$	$\mu_{0.05} \pm \sigma_{0.05}$	$\mu_{0.5} \pm \sigma_{0.5}$	$\mu_{0.95} \pm \sigma_{0.95}$
τ_{Geant4}	90 ± 35	215 ± 60	10900 ± 13000
τ_{ATLAS}	85 ± 70	310 ± 950	8100 ± 13600
τ_{LHCb}	53 ± 80	1300 ± 5300	15100 ± 25000

Table 3.1: Mean μ and standard deviation σ of selected quantiles of τ distribution for Geant4, ATLAS and LHCb VOs.

minutes, up to 50% of jobs start within 5 minutes, while starting 95% of all jobs takes up to 10 hours and may take as long as 34.5 hours for the late-arrival jobs. In the ATLAS VO at 0.95 confidence level, the first 5% of jobs start in less than 3 minutes, up to 50% of jobs start within 5 minutes, while starting 95% of all jobs takes up to 11 hours and may take as long as 29.5 hours for the late-arrival jobs. In the LHCb VO at 0.95 confidence level, the first 5% of jobs start in less than 4 minutes, up to 50% of jobs start within 14 minutes, while starting 95% of all jobs takes up to 20 hours and may take as long as 45 hours for the late-arrival jobs.

Across all sampling intervals the τ distributions are rather dispersed. Standard deviation of $\tau_{0.05}$, $\tau_{0.5}$ and $\tau_{0.95}$ quantiles between all sampling intervals is often higher than the mean as shown in Tab. 3.1. This indicates that the processes producing τ distributions are rather chaotic, heavily tailed and not stationary in long term. These general observations are consistent with other studies which try to model and predict the job waiting times. Models based on Markov chains have been proposed in [129]. In [38] a 2-phase lognormal distribution to model CE queuing time is proposed. However, in our further study we may use τ as an empirical distribution without assuming any exact model.

Fig. 3.4 shows the cumulative probability distribution (CDF) of job waiting times. Despite small differences, the distributions of probe job queuing times are very similar for all three VOs. Similar distribution, shown with crosses in Fig. 3.4, may also be observed for jobs running real data analysis and simulation applications in the CMS VO. This distribution has been obtained from the Dashboard monitoring service⁵ and it contains 44 K observations from a single day in October 2009. It illustrates the fact that long tails in the job queuing time distributions are present for real applications and have not significantly changed for several years (2006-2009). It also indicates that long tails are intrinsic in the job timing data independently of the monitoring method used: monitoring callbacks (τ_{CMS}) or EGEE Information System (τ_{ATLAS} , τ_{Geant4} , and τ_{LHCb}); and that they are compatible with other studies [25].

Similarity of obtained distributions may be explained by the fact that in each VO there is a fraction of “bad” computing sites where queuing times are long. There are

⁵<http://dashboard.cern.ch> and private communication with J. Andreeva (CERN)

$\mathcal{O}(300)$ computing sites involved in the job brokering process which follow a Tier model, depending on size, operational expertise and hardware, software and human resources available for EGEE Grid operations and middleware maintenance. A small number of large sites belong to Tier 1 (high-availability), while majority of smaller sites belong to Tier 2 or 3 (best effort). It should be noticed that a certain, non-negligible fraction of physical resources (computing nodes) is shared between different VOs, typically via the fair-share policy-based scheduling implemented in the fabric layer (batch systems) of the EGEE Grid sites. Intrinsically inefficient push-based job brokering based on outdated information system may also contribute to observed distributions.

The corresponding probability distribution (PDF) of job waiting times resembles an exponential distribution and follows a straight line on a log-log scale plot, which is a characteristic property of complex systems. We plan to explore this observation in the future to complement the ongoing research which tries to characterize grids as complex systems [93].

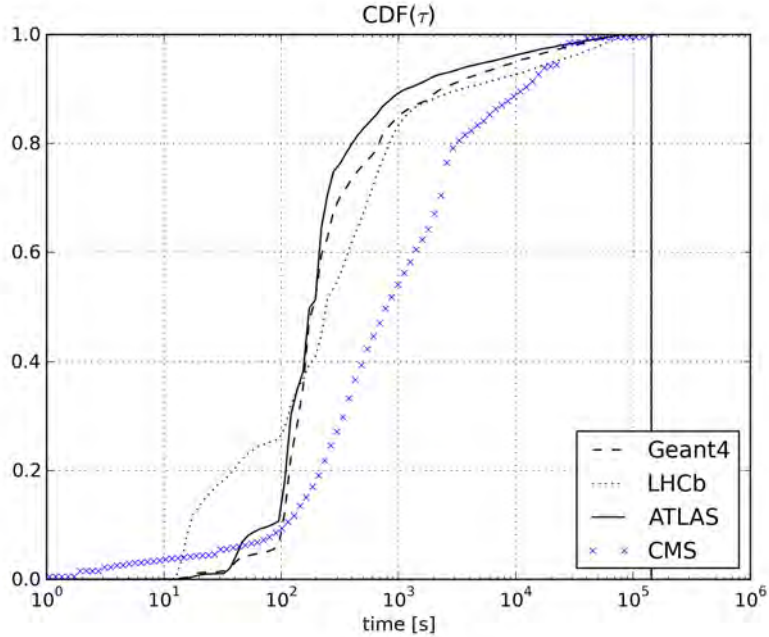


Figure 3.4: Cumulative probability distribution (CDF) of job waiting times τ_{Geant4} , τ_{ATLAS} , τ_{LHCb} , τ_{CMS} . Y-axis shows probability.

The long tails in the τ distribution have a deep impact on the efficiency of job processing. If $P(t \leq \tau)$ is a probability of starting a job at most at time τ described by the $CDF(\tau)$ function, then the probability $P_K(t \leq \tau)$ of K jobs starting not later than at time τ , corresponds to random sampling with replacement of K jobs from the general population of jobs. Each job is drawn independently and starts not later than

at time τ with probability $CDF(\tau)$. Therefore,

$$P_K(t \leq \tau) = \prod_{i=1}^K P(t \leq \tau) = [CDF(\tau)]^K. \quad (3.2)$$

P_K drops sharply when K increases even for large waiting times. For example, the probability $P_{100}(t < \tau_{0.95})$ of starting all 100 submitted jobs not later than at the time corresponding to 0.95 quantile of the distribution τ is only 0.06%. This corresponds to the probability of 100 jobs starting execution not later than 10, 11 and 20 hours after submission in Geant4, ATLAS and LHCb VOs respectively.

3.4 Simulation of task processing models

We analyze the properties of makespan as the QoS metric of interest to a typical end user for late-binding models (U) and classical Grid early-binding models (G).

Equation 3.1 involves independent random variables (system parameters) and functions with unknown analytical representations. Therefore, an analytical approach is possible only for the simplest models U_0 and G_0 where we study the processing on a set of homogeneous resources with grid-specific job waiting time τ . The U_0 model is a continuous approximation of a discrete task model U_1 . In the U_1 model, independent tasks of the same size are executed on a set of homogeneous resources in self-scheduling mode. For U_0 and U_1 models we performed a Monte-Carlo based simulation where system parameters are simulated by sampling with replacement from a population of observations to reflect the empirical distributions characteristic in the EGEE Grid environment. More specifically, we simulate an execution of a set of tasks using a set of grid jobs with queuing time given by the empirical distribution τ , obtained from the real job traces observed in the EGEE Grid. By repeating the simulation multiple times we obtain a probabilistic distribution of makespan L and study its average value μ_L and standard deviation σ_L .

In more complex models U_2 and G_2 the resources are heterogeneous with variable processing efficiency, and in U_3, G_3 models job submission and execution failures are considered. In this thesis we focus the discussion on the effects of the τ distribution and therefore we leave these more complex models for future work. All models are summarized in Tab. 3.2.

We treat simultaneous submission of K jobs by a single user as equivalent of randomly drawing a sequence of K probe observations from τ distribution. In this approach an implicit assumption is that τ is stationary with respect to the activity of a single user. Assuming that the grid is an “infinite, redundant resource” the normal activity of a single user does not change the overall state of the system and the processes underlying the τ distribution remain stationary. This is an obvious approximation as the grid has a finite size, and additionally, the resources are partitioned at the level of Virtual Organizations. An application may also require a particular subset of resources due to execution requirements (hardware constraints, available data, etc), shrinking the effective size of the grid. Although the underlying processes may not be stationary in

model	resources	splitting	job failures
G_0	<i>homogeneous</i>	<i>discrete</i>	no
U_0	<i>homogeneous</i>	<i>continuous</i>	no
U_1	<i>homogeneous</i>	<i>discrete</i>	no
G_2	<i>heterogeneous</i>	<i>discrete</i>	no
U_2	<i>heterogeneous</i>	<i>discrete</i>	no
G_3	<i>heterogeneous</i>	<i>discrete</i>	yes
U_3	<i>heterogeneous</i>	<i>discrete</i>	yes

Table 3.2: Summary of grid submission models with early binding (G) and DIANE user-level scheduler models with late binding (U).

such boundary cases the makespan may only be worsened by the user activity. In this case the tails of the τ distribution may only get longer which implies that our probing methodology gives the best-case of grid behavior because it corresponds to a single user having no effective impact on grid processes.

The assumptions about stationarity of the underlying τ distribution may not hold if K is large. The total number of computing slots in the EGEE Grid as of 2010 is $\mathcal{O}(10^5)$ and with a large number of submitted jobs, say $\mathcal{O}(10^4)$, we may expect the distribution tails to grow due to suddenly increased number of queuing jobs in the system. Therefore, simulation results for large values of K are only used to indicate general trends and must be interpreted correctly.

3.4.1 The basics: G_0 and U_0 models

In this section we consider the most basic models G_0 and U_0 for which we derive the makespan L as a function of a random variable τ . We assume that the processing efficiency $\varphi_i = p_i = \text{const}$ is a single parameter which directly corresponds to the processing power of the worker node. We ignore runtime errors and grid submission failures. We also ignore any correlations between p and τ e.g. grid sites with consistently faster processors and shorter job waiting times.

Assuming p_i is constant in time for a given computing slot then Equation 3.1 becomes:

$$W(t) = \begin{cases} \sum_{i=1}^K p_i \times (t - \tau_i) & \text{if } t \in [\tau_i, \tau_i + \nu_i] \\ 0 & \text{otherwise} \end{cases} \quad (3.3)$$

To simplify equations but without losing generality we take p to be equal, $p_i = p$, for all worker nodes.

First, we consider a model of classic grid submission G_0 where a user submits K independent jobs at the same time. The total workload W is split before the submission in K equal parts. One part is processed by one job. The makespan is then bound by

the last starting job:

$$L_{G_0} = \frac{W}{pK} + \max(\{\tau_1, \dots, \tau_K\}) = \frac{W}{pK} + \tau_K \quad (3.4)$$

Thus, for large number of jobs the G_0 model realizes the “worst-case scenario”:

$$\lim_{K \rightarrow \infty} \mu_{L_{G_0}} \rightarrow \max(\tau) \quad (3.5)$$

$$\lim_{K \rightarrow \infty} \sigma_{L_{G_0}}^2 \rightarrow \sigma_\tau^2 \quad (3.6)$$

In late-binding models U a user also submits K independent jobs at the same time, however, the workload is distributed at runtime. In U_0 model we assume that the workload is freely divisible such that the scheduler dispatches arbitrarily small quanta of work to the worker nodes and the workload is processed with the speed proportional to p and delivered to the end user without latency or other communication overheads. We also assume that each quantum of work is assigned exactly once to the given worker node. The U_0 is a theoretical, continuous model which allows to focus the study on the impact of τ distribution on the Quality of Service. It may also be analyzed analytically in a straightforward way.

The amount of work completed by such system follows from Equation 3.3 to be

$$W(t) = \sum_{\tau_i < t} p \times (t - \tau_i) = p \sum_{\tau_i < t} (t - \tau_i) \quad (3.7)$$

If k computing slots are used then by rewriting the previous equation we obtain the makespan for total workload W :

$$L_{U_0} = \frac{W}{pk} + \frac{\sum_k \tau_i}{k}, \tau_k < t. \quad (3.8)$$

As k grows the $\frac{W}{pk}$ converges to zero and the remaining component contains a sum of identically distributed random variables which, following the Central Limit Theorem converges to a normal distribution with the variance decreasing proportionally to the number of used computing slots:

$$\lim_{k \rightarrow \infty} \mu_{L_{U_0}} \rightarrow \mu_\tau \quad (3.9)$$

$$\lim_{k \rightarrow \infty} \sigma_{L_{U_0}}^2 \rightarrow \frac{\sigma_\tau^2}{k} \quad (3.10)$$

In the U_0 model a user does not only wait shorter for the completion of workload but also the variance of the waiting time is smaller. In the G_0 model all K submitted jobs must be successfully finished for U_0 a smaller number of effectively running jobs $k \leq K$ may be sufficient to complete the workload. This is demonstrated by the empirical distributions of L_{G_0} and L_{U_0} with the effective number of jobs (k) shown for three problems sizes, small workload in Fig. 3.5, medium-size workload in Fig. 3.6 and large

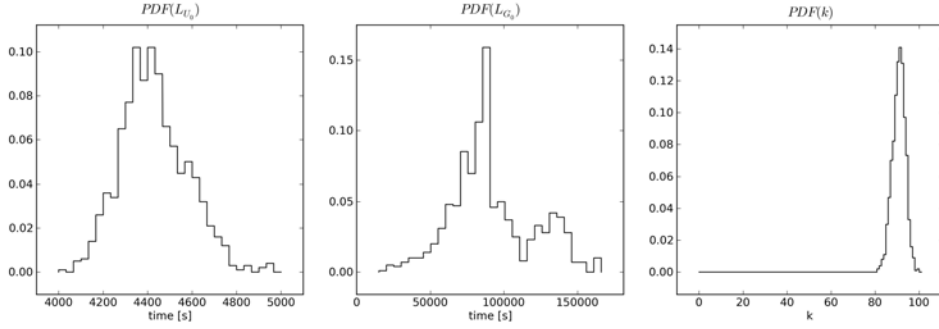


Figure 3.5: Small workload. Probability distribution (PDF) of L_{G_0} , L_{U_0} and k for τ_{LHCb} , $W = 100$ CPU-hours, $K = 100$. This corresponds to a total workload of 4 CPU-days and an average load of 1 CPU-hour per job.

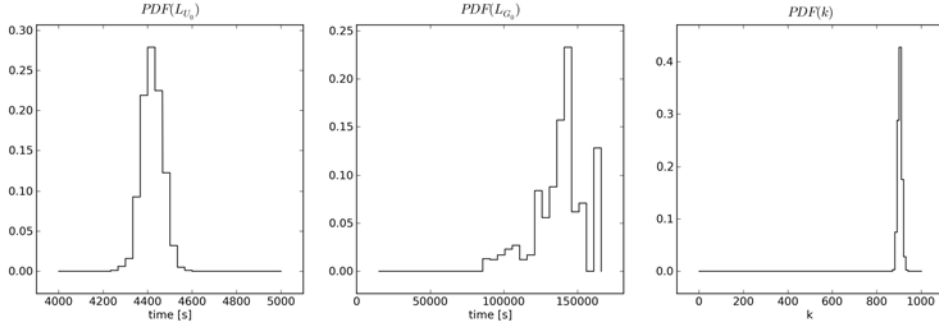


Figure 3.6: Medium-size workload. Probability distribution (PDF) of L_{G_0} , L_{U_0} and k for τ_{LHCb} , $W = 1000$ CPU-hours, $K = 1000$. This corresponds to a total workload of 40 CPU-days and an average load of 1 CPU-hour per job.

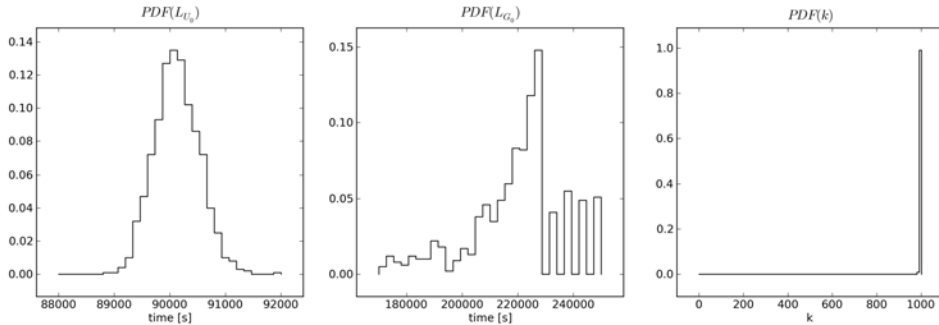


Figure 3.7: Large workload. Probability distribution (PDF) of L_{G_0} , L_{U_0} and k for τ_{LHCb} , $W = 24 \times 10^3$ CPU-hours, $K = 1000$. This corresponds to a total workload of 3 CPU-years and an average load of 24 CPU-hour per job.

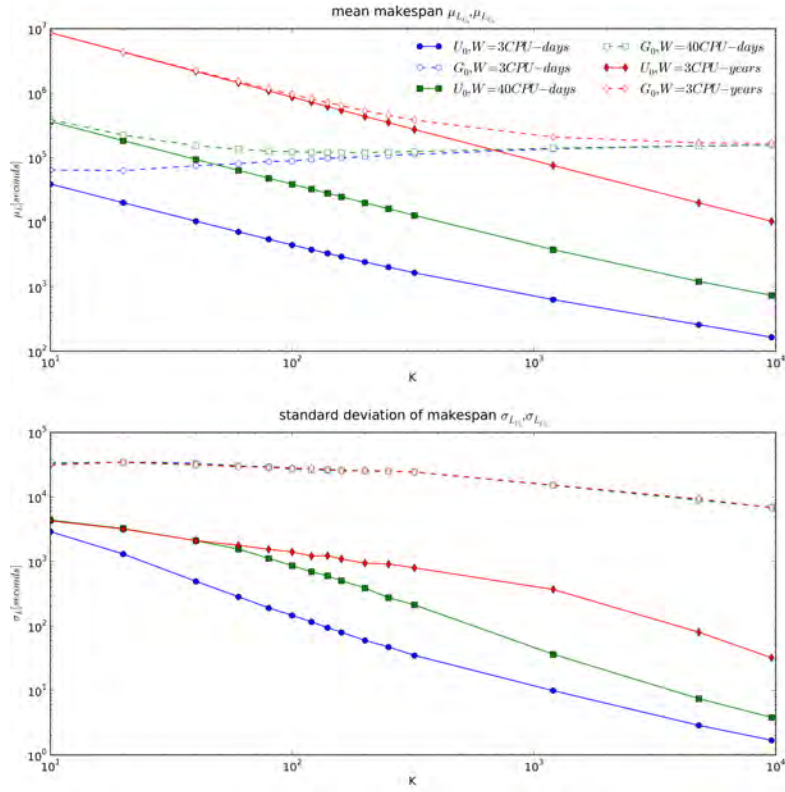


Figure 3.8: Mean makespan and standard deviation for τ_{LHCb} as a function of K submitted jobs for three problem sizes. Some low values of K may be unrealistic in practice: e.g. if $K = 40$ and $W = 3\text{CPU-years}$ then $w = 25$ CPU-days which is out of bounds for typical batch systems present in the EGEE Grid sites.

workload in Fig. 3.7. For clarity we choose the units such that with $p = 1$ the amount of workload directly corresponds to the execution time needed to process the jobs, e.g. 1CPU-day workload is equivalent to one day of execution of one job.

Fig. 3.8 shows the mean and standard deviation of makespan for the three problem sizes as a function of K submitted jobs. The efficiency of the U_0 model is higher when the average workload per job $w = \frac{W}{K}$ is small and its effect is more pronounced for the mean execution time. The standard deviation is consistently lower for U_0 model by one to four orders of magnitude in all considered cases. Tab. 3.3 shows the speedup

$$S_{U_0, G_0} = \frac{\mu_{L_{G_0}}}{\mu_{L_{U_0}}} \quad (3.11)$$

for selected configurations.

W	K	$w = \frac{W}{K}$	S_{U_0, G_0}
4 CPUd	10	10 CPUh	1.6
	60	1.6 CPUh	11.3
	100	1 CPUh	21.3
	9600	37.5 CPUs	1000
40 CPUd	10	4 CPUd	1.06
	60	16.5 CPUh	2.11
	320	3 CPUh	9.7
	9600	375 CPUs	214
3 CPU _{yr}	≤100	≥10 CPUd	≤ 1.1
	320	75 CPUh	1.4
	1200	20 CPUh	2.8
	9600	2.5 CPUh	16

Table 3.3: Task processing speedup achieved in U_0 model. Higher values are better.

The quartile coefficient of dispersion,

$$c = \frac{\tau_{q75} - \tau_{q25}}{\tau_{q75} + \tau_{q25}} \quad (3.12)$$

corresponds to the slope of the middle part of τ distribution with $c_{ATLAS} = 0.426$, $c_{Geant4} = 0.56$, $c_{LHCb} = 0.796$. The dispersion coefficient of L_{U_0} for all three τ distributions consistently decreases by a factor $\mathcal{O}(25)$ for small workload and factor $\mathcal{O}(250)$ for the large workload. Thus the U_0 model gives a consistent reduction of variance despite differences in the underlying τ distributions. The makespan in the late-binding model is not sensitive to changes in the underlying distribution τ , thus for practical purposes the exact knowledge and modeling of this distribution has less importance.

In the early-binding G_0 model the overhead of grid job scheduling and queuing time results in high-variability of makespan, which negatively impacts the Quality of Service of the EGEE Grid perceived by the end users. As identified in [73] early binding is not suitable for certain applications which require responsiveness and interactivity and is unacceptable in high-granularity job splitting scenarios where many thousands of short jobs must complete in short time.

The U_0 model explains the intrinsic advantage of the late binding model over classic job submission in case of heavy-tailed distribution of job waiting times. It represents an idealized system to provide upper bounds for the QoS metrics for the subsequent, more realistic models. It proves that the late binding by construction allows to dramatically reduce the variance and substantially reduce the average makespan even if the underlying system does not provide strong guarantees for job execution waiting times.

3.4.2 U_1 model for discrete tasks

In the U_1 model the work is split into N tasks which are handled by the late-binding system. Due to discrete nature of the model $L_{U_1} > L_{U_0}$ is expected.

If scheduling overheads are ignored and processor speed is constant then $\varphi_i = p = \text{const}$ and

$$\lim_{N \rightarrow \infty} \mu_{L_{U_1}} \rightarrow \mu_{L_{U_0}} \quad (3.13)$$

$$\lim_{N \rightarrow \infty} \sigma_{L_{U_1}} \rightarrow \sigma_{L_{U_0}} \quad (3.14)$$

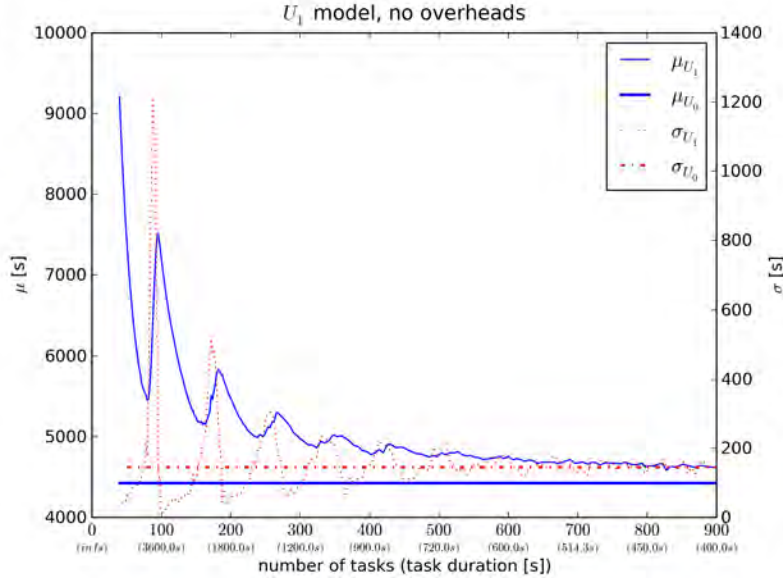


Figure 3.9: Simulated mean makespan and its standard deviation in discrete task model U_1 for small workload $W=100$ CPUh and $K=100$ submitted worker agents.

Fig. 3.9 shows the simulated mean makespan and its standard deviation for small workload $W=100$ CPUh and $K=100$ submitted worker agents in the function of N equal tasks. As expected U_1 model converges to U_0 as N increases. The values of μ_L and σ_L oscillate in a complex way due to *task paging* described below. Task paging is a load-balancing effect which in a grid depends on the shape of τ distribution.

For $N \ll K$ few workers process few large tasks thus the makespan μ_L is large. However, the variance of makespan σ_L^2 is small because only a small subset of K submitted workers processes the tasks. As the dispersion coefficient of the τ distribution is smaller in the inter-quartile range then in the tails, the effective variance of τ for a subset of quickest workers is smaller then the variance of general population of workers. As N increases the variance grows as the workers drawn from the tail of τ distribution start processing. The makespan reaches a local minimum at the point when the workers which arrived first to the system start processing more than one task, thus increasing the makespan. As N increases further the system flips between the state in which all

workers process exactly one tasks and some workers processing more than one. The boundary between the two states is determined by the dispersion of the τ distribution in the point corresponding to the current value of makespan. The variance suddenly drops when the makespan becomes bound by the workers processing two tasks. Again these workers are the quickest ones with the smallest variance of τ . As the task duration decreases as N grows, the makespan reaches a local maximum and starts decreasing. The process is repeated periodically according to the characteristic value of τ in the function of average task duration W/N . The amplitude of μ_L is bound by the average task duration.

From a user perspective, the U_1 model yields consistently lower values of σ_L than plain grid G_0 model, however, it is hard to predict exact bounds for oscillating σ_L , especially for $N \approx K$.

For the realistic U_1 model it is necessary to consider a time-cost to schedule and execute each task, what results in the worker being periodically idle. In this case $\varphi_i(t)$ is a step function with a value in p when the worker is active and 0 when the worker is idle. Average time spent on such scheduling and communication in the late-binding system for each task is denoted by t_{idle} . We may safely assume that per-task time spent by the master itself is negligible for simple algorithms such as self-scheduling and that t_{idle} is bound by the task transfer time i.e. time needed to transfer task input and output parameters between the master and the worker.

If the amount of data associated with task input and output parameters is small then communication overheads are dominated by network latency between grid sites. In this case $t_{idle} \approx 1s$ as the typical network latencies in wide area networks are in the order of few hundred milliseconds.

Sometimes larger data structures, including entire files, may be part of task input or output parameters. The details are application-dependent but estimation of t_{idle} based on applications already deployed using a late-binding scheduler may be used as a reference. One such example is the Lattice QCD Thermodynamics simulation [142] where the input and output data of each task amounted to 10 MB. Around 300 K task transfers were measured over several weeks in 2008 between DIANE master at CERN and DIANE workers distributed among 110 EGEE Grid sites. Average task transfer time was 5 seconds, independently of the transfer direction (from or to master server). 50% all transfers took less than 2 seconds and 95% of all transfers less than 17 seconds. This indicates a range of typical values of t_{idle} for such task transfer to be between 5 and 20 seconds.

Fig. 3.10 shows average makespan for a range of t_{idle} values for $W=100$ CPUh and $K=100$ submitted worker jobs. For $t_{idle} \neq 0$ makespan L_{U_1} reaches global minimum when $\frac{W}{N} \approx 10 \times t_{idle}$. Beyond that point constant overhead t_{idle} becomes predominant and makespan L_{U_1} grows linearly to reach $L_{U_1} = 2 \times L_{U_0}$ for $W/N \approx t_{idle}$. The optimal choice of task granularity requires a subtle balance between undesirable task paging effects when number of tasks is comparable to the number of available computing slots (worker agents) and the task execution overheads when the task duration becomes too short.

The t_{idle} analysis is also applicable for other fixed overheads which may be associated with task execution. For example, certain Monte Carlo simulations may require a non-

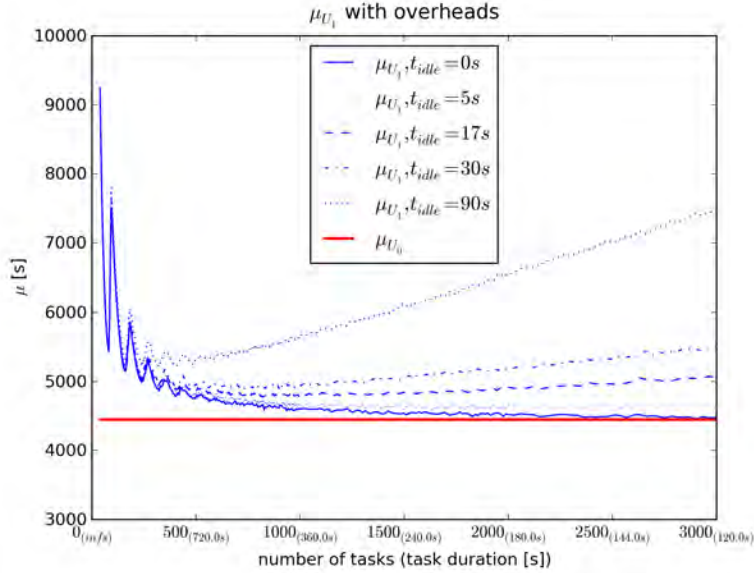


Figure 3.10: Simulated mean makespan in discrete task model U_1 with overhead $t_{idle} \leq 100s$ for small workload $W=100$ CPUh and $K=100$ submitted worker agents as a function of number of tasks. In parenthesis the corresponding single task duration is indicated (in seconds).

negligible initialization time regardless of the number of subsequently simulated events. Data analysis applications often require interaction with external storage which often is preceded by a non-negligible initialization time and may involve other significant overheads related to the data-management. Such applications are penalized when the tasks become too short.

3.4.3 Over-submission of jobs

Late-binding task scheduling models allow to reduce unwanted effects of a heavy tailed τ distribution, if a number of tasks executed by a single computing slot (job) is sufficiently large. The *effective* waiting time for a computing slot is a truncated τ distribution where a subset of first k computing slots, out of K requested ones, is active. The over-submission factor $f_o = 1 - \frac{k}{K}$ is a proportion of submitted requests for computing slots which are unused. For the three typical workloads presented in Fig. 3.5, 3.6, 3.7 the over-submission factor f_o is smaller than 10% and further decreases for larger absolute values of makespan.

For some applications, early binding with heavy oversubmission of jobs may be used to reduce the makespan. Out of K submitted jobs only first $k \ll K$ are kept and the remaining ones are canceled. The main problem with this is that for such

redundant scheduling to be efficient one must involve an excessive number of canceled jobs ($f_o \approx 1$) which may overload or disrupt the queuing systems leading to longer waiting times [36]. Redundant job submission with early binding is also limited to certain specific applications, where it is not required to execute all tasks to complete the computation. For example, in some Monte Carlo simulations the goal is to simulate a certain number of events independently on which set of actual tasks achieves this goal. In many other applications, however, such as parameter sweeps, data mining or simulation based on spatial or temporal division of problem domain the goal is to process each defined task exactly once, and late binding is the only practical option to reduce the makespan.

3.5 Summary

Late-binding scheduling is a technique to improve the Quality of Service of large distributed systems such as EGEE Grid where processing of user tasks which is done on a best-effort basis. Large distributed grids federate hundreds of computing sites and independent job queues. Hence heavily tailed distributions of job waiting times and frequent failures at various levels are inevitable. We have studied and compared standard task processing technique with a late-binding model with private user task queues managed by a User-level Overlay. We identified theoretical and practical reasons why late binding outperforms the standard approach for moldable task processing applications:

- late-binding scheduling model allows to reduce undesirable effects of tails of the τ distribution on a basis of Central Limit Theorem;
- for discrete tasks mean and variance of makespan depend in a complex way on the number of tasks and their total duration;
- late binding allows to achieve speedups which are often greater than an order of magnitude compared to early binding;
- the optimal choice of task granularity depends on the time-cost to schedule and execute each individual task which should not exceed 10% task execution time for optimal performance;
- in extreme cases of very large workloads (in order of CPU-years) and relatively small number of worker nodes (less than hundred) the early-binding and late-binding systems are equivalent.

CHAPTER 4

Development of the User-level Overlay

Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it. And to make matters worse: complexity sells better.

E.W. Dijkstra

Make everything as simple as possible, but not simpler.

A. Einstein

In this Chapter¹ we explain the development strategy and architecture of a User-level Overlay which implements the late-binding task processing model described in the previous Chapter. The User-level Overlay consists of loosely-coupled, reusable and customizable services and tools based on GANGA and DIANE software packages. Such a strategy allows to easily develop high-level, application-specific functionalities such as heuristic resource selection and adaptive task scheduling.

¹The results described in this Chapter formed the basis of the following papers: J. Mościcki, F. Brochu, J. Ebke, U. Egede, J. Elmsheuser, K. Harrison, R. Jones, H. Lee, D. Liko, A. Maier, A. Muraru, G. Patrick, K. Pajchel, W. Reece, B. Samset, M. Slater, A. Soroko, C. Tan, D. van der Ster, and M. Williams. Ganga: A tool for computational-task management and easy access to Grid resources. *Computer Physics Communications*, 180(11):2303 – 2316, 2009; V. Korkhov, J. Mościcki, and V. Krzhizhanovskaya. The user-level scheduling of divisible load parallel applications with resource selection and adaptive workload balancing on the Grid. *IEEE Systems Journal*, 3:121–130, March 2009.

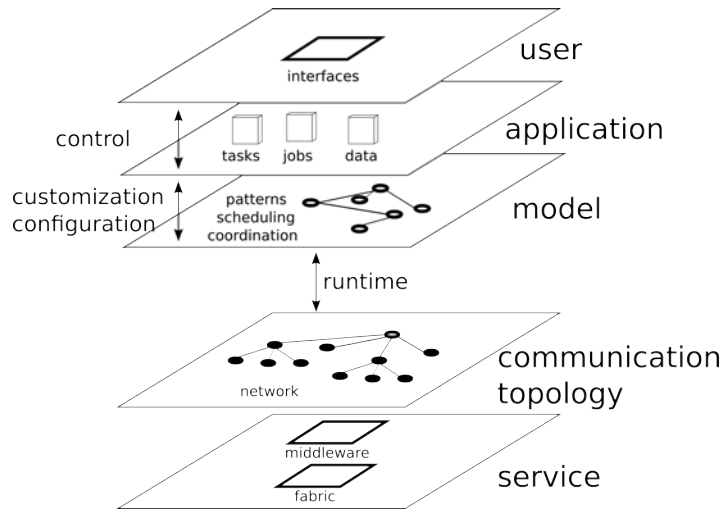


Figure 4.1: Layered model for grid computing includes both system and application domains. Separation between application and model is to emphasize the fact, that many applications follow common scheduling patterns and coordination methods and thus may take advantage of tools which support generic application execution models.

4.1 Vision

Emergence of large-scale production grids has changed the way computing capacities and capabilities are provided to the applications, simply because resources are not placed in a unique administrative domain and the scale and complexity of these systems tends to be overwhelmingly large. For example, traditional techniques developed for clusters, such as administrative fine tuning of scheduling, or prioritization of access to resources on a basis of individual applications, are not anymore practical. Instead, grid environment naturally falls into two somewhat distinct domains: a system layer which includes infrastructure and generic middleware services and an application layer which includes user jobs, applications and tools (Fig. 4.1). An attempt of mastering grid dynamics inevitably shifts our focus onto the application layer as a most likely place where such issues may be addressed, also because there are many grids and many middleware stacks, and we may sometimes need to use them simultaneously.

We use the term *overlay* to emphasize the fact that we build a layer above existing middleware services and across available grids and other distributed infrastructures without modifying them. Overlay in this context refers to a model layer in Fig. 4.1, which consists of interconnected software elements which span *above* the system domain. Layering is a key for overlays: grid users do not control resources and they have little influence on choice, life-cycle, upgrades and availability of middleware services. In many cases it is important that application-specific needs are accommodated at timescales convenient to the application community. These timescales are often not correlated at all with the timescales convenient to the resource and middleware providers. Moreover,

deployment and operation of large-scale *production* grids is radically different from practices in grid computing *research* which does not make it easier to apply new scientific concepts in production environments².

With grids being multi-institutional federations of resources, trust relationships between resource providers and consumers are a delicate issue. Resource providers are not easily convinced to install application-specific services in their sites, as it often requires privileged access and special procedures negotiated as Memoranda of Understanding or Service Level Agreements [4]. Therefore, we use the term *user-level* to emphasize the fact that tools and services provided by the User-level Overlay live entirely in the user-space and do not require special access rights for installation and operation. Moreover, we require that all components of the User-level Overlay should use single user credentials. This is to be completely compliant with the grid security model which assumes accounting and traceability at the user level.

For this strategy to be effective only minimal assumptions on available grid services and infrastructures should be made. We envisage the User-level Overlay as a self-contained bundle of tools and services which travel together with the application code to be run by remote execution services. At runtime the User-level Overlay transforms into a layer of interconnected components, spanning multiple grid sites. To achieve this we rely on outbound network connectivity to be available on the worker nodes in a grid. Outbound connectivity is the least common denominator of virtually all distributed infrastructures which are currently in production.

Late-binding overlays raise concerns of resource owners about traceability and fair-share use of the system. For example, VO-centric overlays sometimes allow mixing of worker agents and tasks of different users, which may result in problems of distinguishing who is the owner of the application running on a particular resource. On the other hand, this feature allows the VOs to manually control resource sharing policies for their members, however, it requires a special agreement between a VO and resource providers which is only feasible for large user communities. A User-level Overlay is free of this limitation as the worker agents may only execute tasks which belong to the same user. In addition, running another set of tasks typically requires another set of worker agents jobs to be submitted and therefore the User-level Overlay does not bypass the fair-share mechanisms provided by grid middleware and underlying batch/fabric layers.

It is clear that no single turnkey solution may fit all applications across all distributed infrastructures. Support of flexible and incremental development of application-specific functionalities and abstractions and easy adaptation and customization by a user is the key. Flexibility is needed at the application level as well as at the user level to support increasingly complex use-cases and scientific workflows. Therefore, in our vision the system becomes an application-hosting environment where software plugins may be composed to best fit the purpose of a specific application. At the same time default

²Anecdotal evidence of this has been given by one of our operations/deployment colleagues at a coffee break. He was once talking to a known researcher in grid computing who claimed that “to solve your problems with the middleware component X you should *upgrade your testbed* to take advantage of this entirely new middleware architecture that we recently developed”. The only possible answer was “Well, the fact is that we don’t have a testbed but a production system with 10^5 CPUs, 300 sites, 3×10^5 jobs and several thousand users every day.”

plugins should be provided off-the-shelf to the end users for standard computing tasks.

With the appearance of modern scripting languages such as PYTHON or Ruby the choice of implementation language becomes an essential part of the design [123]. As observed in [148], “with additional basic tools, PYTHON transforms into a high-level language suited for scientific and engineering code that’s often fast enough to be immediately useful but also flexible enough to be sped up with additional extensions”. PYTHON primarily acts as glue, or steering layer, for extension modules which allow to use high-performance compiled code. On the other hand, PYTHON features, such as dynamic typing, runtime decoration, introspection and extension of objects allow to achieve enormous degree of flexibility. Clear syntax allows to express ideas and algorithms as human-readable programs even with limited prior knowledge of the language. Therefore, this particular choice of the implementation language is an important part of our strategy. To emphasize this fact we chose to present algorithms using PYTHON syntax.

4.2 Functional breakdown and architecture

The requirements to be addressed, presented in Sec. 1.4, include: 1) improvement of the Quality of Service, 2) cross-infrastructure interoperability, 3) easy access and transition between different distributed environments for end users and 4) application-specific scheduling and coordination. These requirements may be mapped into the following high-level functional areas:

1. *late-binding task scheduler* for optimization of workload distribution and task coordination,
2. *resource selection algorithms*,
3. *resource access interface* for uniform access to remote resources,
4. *end-user environment* for easy application configuration and management of user jobs.

Fig. 4.2 shows the high-level outline of these functional areas where late binding is a key element. Resource access interface enables to allocate computational resources and to create dynamic resource pools which may span multiple distributed infrastructures. A resource pool consists of processes running on remote worker nodes which are directly controlled by the task scheduler. Resource selection component allows to control the number and quality of the resources in collaboration with the task scheduler. End-user environment provides easy ways for application configuration and management.

A relation between an application and the late-binding scheduler is one of the key issues: a scheduler may be embedded into the application or external to it. A scheduler embedded into the application is developed and optimized specifically for a given application, typically by re-factoring and instrumenting the original application code. It allows fine tuning and customizing the scheduling according to the specific execution patterns of the application. Such a scheduler is intrusive at the application source code

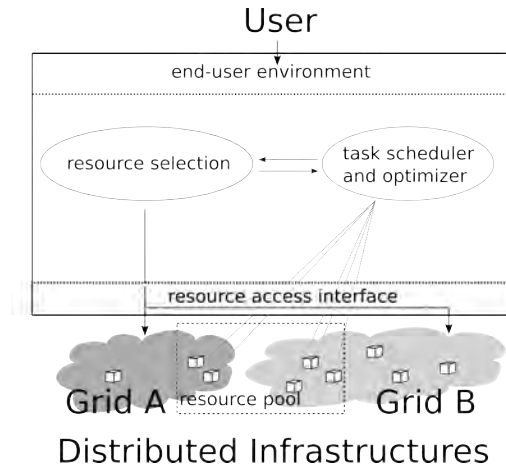


Figure 4.2: Main functional components of User-level Overlay. Resource selection component is used to construct a resource pool through a uniform resource access layer. Task scheduler and optimizer uses the late-binding method to operate directly on a dynamic resource pool. Upper layer provides end-user environment shielding users from underlying complexity of distributed infrastructures.

level which means that the code reuse of the scheduler is reduced and the development effort is high for each application. A scheduler external to the application relies on the general properties of the application such as a particular parallel decomposition method (e.g. task parallelism, iterative decomposition, geometric decomposition or divide-and-conquer). An application adapter connects the external scheduler to the application at runtime. Depending on the decomposition method, the application re-factoring at the source code level may or may not be required. The disadvantage of external schedulers is that it may be very hard to generalize execution patterns for irregular or speculative parallelism. In this case, which occurs in various situations ranging from medical image processing to portfolio optimization [172], a development of a specialized embedded scheduler may be necessary. Any particular solution is a trade-off between usability, flexibility and efficiency. Our User-level Overlay adopts an approach which is mid-way between a generic scheduling service and an embedded scheduler tightly-coupled with the application code.

4.3 DIANE and Ganga software packages

The User-level Overlay consists of loosely-coupled components which have been developed as two separate software products:

- DIANE – providing a task scheduler,
- GANGA – providing a resource access interface and an end-user environment.

DIANE [133, 134] is a task coordination framework which exploits the late-binding method. DIANE provides an application-aware scheduler which may be extended by a system of plugins to support master/worker workloads such as task farms and bag of tasks. Plugins for DAGs [82] and data-oriented workflows [76] have been implemented as third-party contributions by interested user communities. The framework also supports customized, application-specific processing methods and failure-management strategies.

GANGA [136] is a job submission interface to access distributed computing resources in an easy and uniform way. It is an open and extensible framework which allows to submit jobs through a system of plugins and it currently supports Portable Batch System (PBS) [88], Load Sharing Facility (LSF) [163], Sun Grid Engine (SGE) [72], Condor [170], gLite [113], ARC [58], Globus/GRAM [48], GridWay [91] and the SAGA API [77] standard. GANGA has a double role in the User-level Overlay. Firstly, it provides a scripting API to programmatically access remote resources and may be used as a job management abstraction layer. Secondly, GANGA provides a user-oriented working environment for application configuration and management of computational tasks.

4.4 Operation of the User-level Overlay

In the GANGA/DIANE overlay user workload is split dynamically in a number of tasks which are scheduled for execution to a set of worker agents (Fig. 4.3). A scheduler is a part of the master server which is managed by the user as a personal, transient service. Worker agents communicate directly with the master which, by default, automatically stores the results in the local storage area directly accessible by the user. When processing is terminated the overlay is automatically destroyed and resources released.

GANGA interface is used to control the worker agents which may be submitted, killed and removed as simple batch jobs. When started, worker agents pull the tasks from the master which controls the distribution of work. The system is fault-tolerant and may run autonomously: a worker agent which fails to complete the assigned calculations is replaced by another worker agent. Late-binding allows to reduce the scheduling overhead and to dynamically adapt to changing workload and evolving pool of available resources.

Resource selection is controlled by the user with the scripting interface of GANGA. Several simple submitter scripts are provided which allow the user to submit a specified number of worker agent jobs directly from the system shell. Worker pool may also be controlled automatically by Agent Factories – specialized submitter scripts which provide streams of worker agent jobs over longer time periods. Agent Factories may implement advanced resource selection schemes based on application feedback and in response to the performance requirements of the application which are provided by the task scheduler. At the conceptual level, loose coupling between resource selection and task scheduling allows the quality, provenance and number of computational resources (worker agent jobs) to be controlled independently of the number of tasks at the application level.

At the software level, loose coupling between GANGA and DIANE is also beneficial

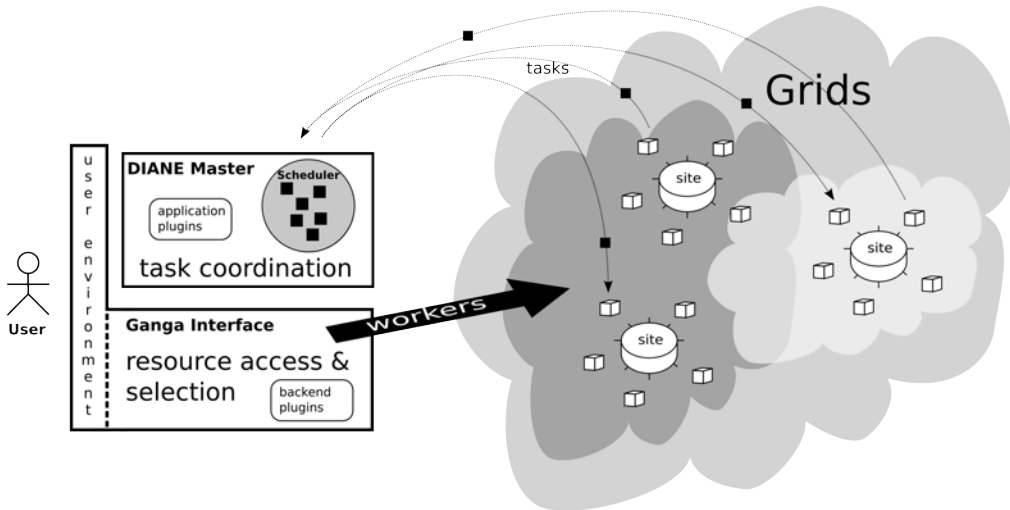


Figure 4.3: Architecture of the User-level Overlay operating on multiple grids at the same time. Task coordination in a Master/Worker model is provided by DIANE. Resources are acquired by sending worker agent jobs using GANGA interface. Multiple clouds represent different computing environments and grids. User has a direct control over GANGA and DIANE components and plugins.

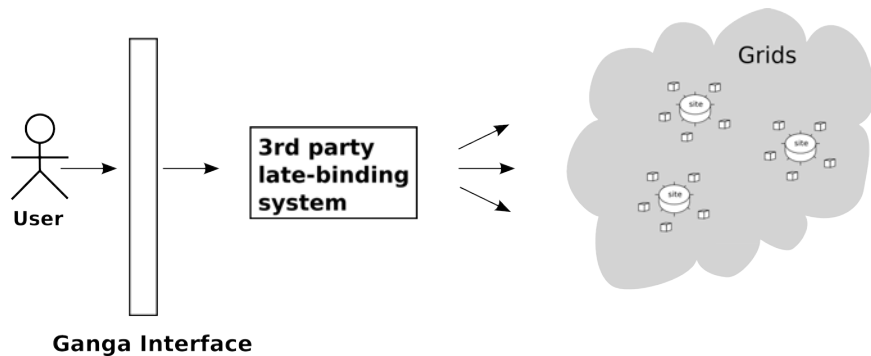


Figure 4.4: User-level components in an alternative configuration: third-party late-binding task management system replacing DIANE. GANGA provides an interface which hides complexity of actual implementation of task management system and underlying distributed infrastructures.

as it allows to replace specific functionalities with other implementations, thus making integration into existing domain-specific frameworks more easy. A user may choose a component to be used separately or to be replaced by third-party solution as shown in Fig. 4.4 where DIANE is replaced by a third-party task management system. For example, this configuration is used in large High Energy Physics communities, where third-party task management is implemented as VO-centric overlay (discussed in the next Chapter). In this case users are interacting with the task management system via GANGA.

4.5 The DIANE task coordination framework

The DIANE task coordination framework is based on a simple distributed programming model which follows the Master/Worker pattern. The implementation uses PYTHON and a high-performance Object Resource Broker, omniORB³.

DIANE model relies on an ability to partition the computational problem into parametric tasks which are executed independently as presented in Fig. 4.5. Tasks are defined as basic execution blocks which convert a set of input parameters into a set of output parameters. Execution block may be a sequence of programming instructions, an invocation of an executable, a set of OpenMP threads or MPI processes to accomplish the task. The tasks may be created dynamically at runtime or arranged in more complex structures to express data flow, execution dependency or specific ordering in the task execution model.

The ensemble of computation consists of many tasks and it is called *a run*. The system consists of many worker processes which communicate with one master process (the worker processes do not need to share the filesystem nor memory). A task operation is specified by a set of parameters which are produced by the RunMaster (running on a master node) and consumed by the WorkerAgent (running on a worker node). The worker produces the output and sends it back to the master. The RunMaster keeps track of tasks in order to react to failures in the task execution, spawns and coordinates new tasks.

A built-in FileTransferService provides a simple but reliable way of transferring files between RunMaster and the WorkerAgents. It provides a simple PYTHON-based API which allows to download and upload files without file-size restrictions and may be readily used in the DIANE component plugins. FileTransferService may be used to overcome limitations of certain middleware implementations, such as gLite WMS service which typically restricts a single file size to 10 MB.

4.5.1 Layered architecture of DIANE

The framework consists of three layers, as shown in Fig. 4.6:

1. application and scheduling layer,
2. core framework,

³<http://omniorb.sourceforge.net>

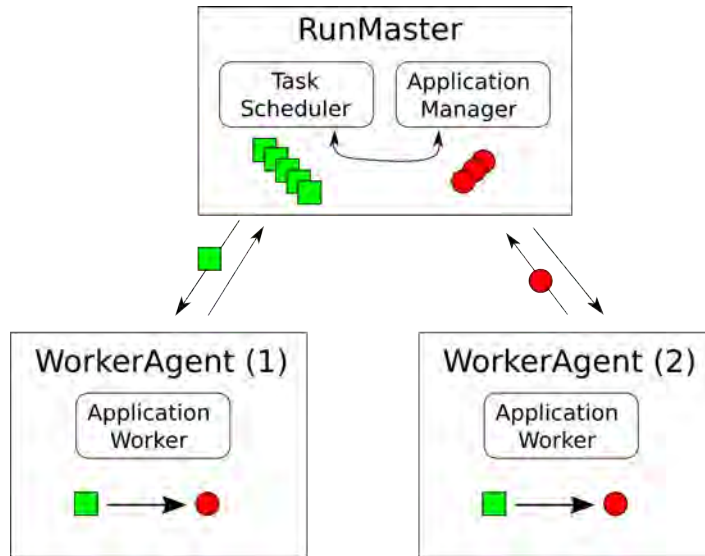


Figure 4.5: Task coordination framework architecture. RunMaster schedules tasks for processing by WorkerAgents using the TaskScheduler component (task inputs are shown as squares). Output of task execution is automatically postprocessed by the RunMaster using the ApplicationManager component (task outputs are shown as circles).

3. transport and networking layer.

Transport and networking layer provides the communication protocol between remote processes and handles other low-level communication aspects such as TCP connection management and server thread management. The core framework implements the Master/Worker processing logic which transparently handles the interaction between the RunMaster and WorkerAgents. It provides a *software bus* which connects components in the application and scheduling layer.

The layered architecture allows to decouple the physical networking topology from the application processing logic. This is convenient because 1) it allows to transparently replace the transport and networking layer by another implementation if needed and 2) it allows to transparently adapt the physical communication model to other topologies such as multi-tier masters. Multi-tier masters would allow the coordination of physically co-located worker nodes to be managed by a local sub-master to increase scalability. Currently in the EGEE Grid this requires setting up dedicated services (so-called VO-boxes) in the grid sites. VO-boxes are used by large VOs to orchestrate critical activities, such as data transfers in High Energy Physics. However, this option is generally not available to individual users. In the future, communication which is now routed via a central point could be short-circuited using multi-tier masters, with advantages for scalability and network traffic control.

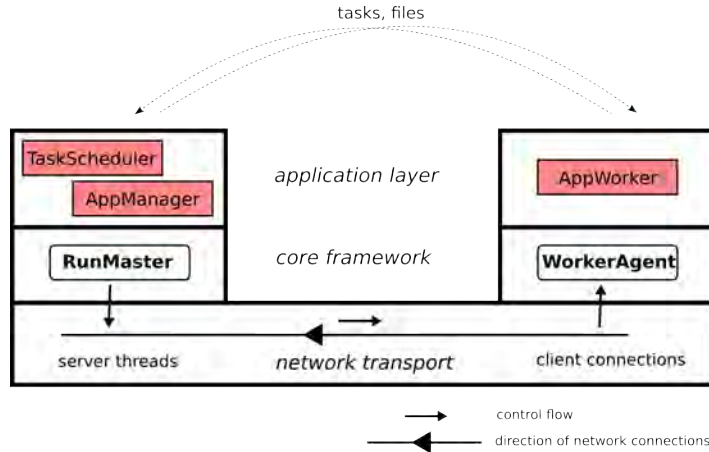


Figure 4.6: Layered architecture of DIANE. Network transport layer handles communication details and is encapsulated and invisible to the application layer which defines application processing logic. Framework layer defines interaction between distributed processing elements.

4.5.2 Application and scheduling layer

Application plugins are implemented as PYTHON classes as shown in Fig. 4.7. TaskScheduler keeps track of processed task entries (which also includes task input and output parameters) and is responsible of mapping tasks to workers and handling the task-failure policies. SimpleTaskScheduler is provided for bag-of-task applications to be used out-of-the box and it uses flat task queues for book-keeping. Alternative implementations of the TaskScheduler allow the task mapping and coordination logic to be easily plugged in. A developer of a specific TaskScheduler implementation is free to choose the most appropriate data structure to keep the task entries for the bookkeeping.

TaskScheduler is an active object (thread) which runs in the master process, alongside the RunMaster thread (core service). The RunMaster notifies the scheduler of the events such as task and worker status changes. The scheduler may schedule/unschedule tasks and remove workers by calling appropriate methods of the RunMaster object. The communication between RunMaster and TaskScheduler is fully asynchronous and implemented through thread-safe in-memory queues.

ApplicationManager handles task creation, pre- and post-processing of task input and output such as merging of output for completed tasks. Task scheduler forwards event notifications to the ApplicationManager. The separation between the application manager and the task scheduler enables to isolate scheduling aspects from application actions (such as merging outputs of successfully completed tasks).

Clear separation of concerns and asynchronous callback architecture provide a flexible and open environment. It is free of a priori constraints of approaches which rely on generalized, abstract representations of application and control-flow models which are often expressed through specialized markup languages such as GSFL [111]. Plugin

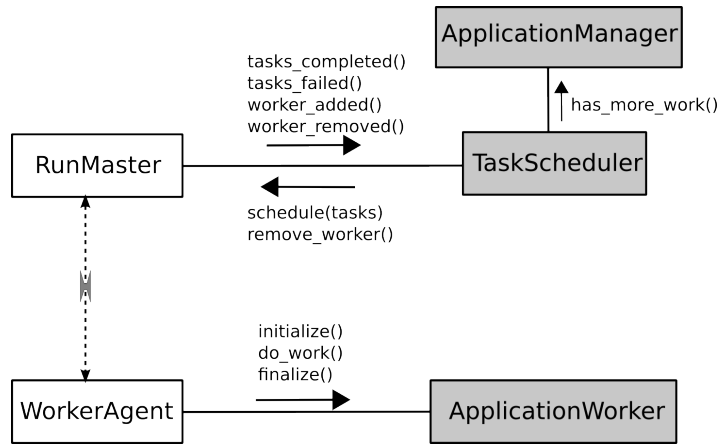


Figure 4.7: Collaboration diagram showing the interaction of RunMaster and Worker-Agent components. Plugin classes (shown in gray) allow to customize the scheduling and application-specific actions of the framework.

components may implement specific application functionality or be generalized for a subset of applications sharing similar logic. For example, a DAG scheduler allows to specify job dependencies using the DAGman [55] representation. DAG scheduler may be easily combined with application-specific ApplicationManager, or even programmatically extended to include application-specific features. This strategy allows to use at the same time the declarative and imperative styles for application modeling, scheduling and control in a flexible, extensible and efficient way.

ApplicationWorker is responsible for actual processing of the task on the worker node. The WorkerAgent process, passes the task parameters received from the RunMaster and waits for output produced by the application worker to send it back to the RunMaster. When a WorkerAgent joins or leaves the worker pool, it is initialized (or finalized) by calling appropriate methods of the ApplicationWorker.

By default ApplicationWorker module is loaded in-process by the WorkerAgent (imported as a PYTHON module), however, a configuration option exists to load the ApplicationWorker as a separate process and use local IPC channel to communicate with the WorkerAgent. This is useful if the ApplicationWorker uses compiled extension modules which may crash the hosting process. With out-of-process option enabled, the WorkerAgent may cleanly report the application crash to the RunMaster.

4.5.3 Core framework

The core framework implements the communication logic of the Master/Worker pattern using CORBA objects. RunMaster keeps track of registered workers in a list. The worker parameter list includes the worker status (idle, busy, dead or lost), history of status update timestamps, processed task identifiers etc. The RunMaster effectively manages a pool of worker nodes – it provides mechanisms to launch new tasks on remote

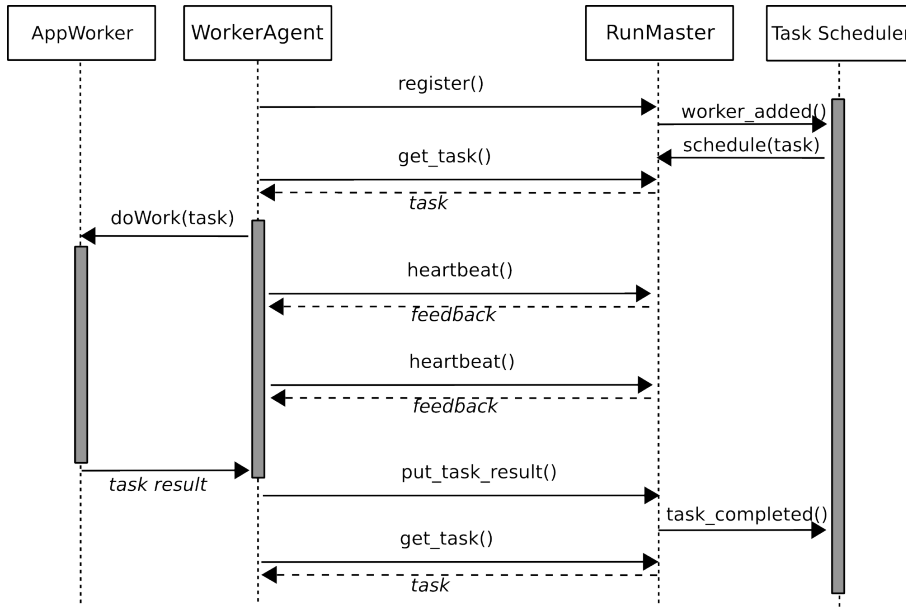


Figure 4.8: Core framework sequence diagram showing a Master/Worker interaction and unidirectional communication model with heartbeats.

resources and to report finished tasks to the application layer, as shown in Fig. 4.8.

The RunMaster detects worker node or network failures through a configurable heartbeat mechanism. WorkerAgents send periodic confirmations of their status to allow the RunMaster to detect crashed or idle workers. If a worker fails to report its status within a specified time it is marked as *lost* and it is up to the components in the application layer to implement a suitable recovery policy (typically rescheduling tasks to another WorkerAgent). Similarly, if a worker remains idle for a specific time then it may be automatically removed from the pool. The framework guarantees consistent handling of the worker pool: once a worker is removed, all further connection attempts of that worker are ignored by the master.

WorkerAgent bookkeeping entries which are used by the RunMaster may be easily extended with arbitrary attributes, for example data tags. Such application-specific information may be used by the TaskScheduler or ApplicationManager to make scheduling decisions based on the tag information derived at runtime by the worker nodes. In addition, the Core framework automatically collects benchmarking and system information about the worker nodes (hardware characteristics, operating system and environment) which may be easily accessed by the scheduler.

4.5.4 Transport and networking layer

The underlying transport and networking layer is based on omniORB with TCP/IP and is invisible to the application layer. This layer is responsible for reliable messaging between the master and the workers, scalable connection management and high-performance server thread management.

From the networking point of view the workers are clients of the master server and the communication is always unidirectional: from the workers to the master. Therefore, the connectivity requirements are minimal: outbound connectivity from the worker nodes and at least one open port for inbound connectivity (through a potential firewall) on the master node.

Bi-directional communication is based on periodic polling through heartbeats. The RunMaster effectively sends feedback to the WorkerAgents in a reply to a heartbeat as shown in Fig. 4.8. This model comes as an improvement of previously used bi-directional communication scheme in which the RunMaster could actively send messages to a WorkerAgent by reusing existing connections to the WorkerAgent. WorkerAgent was effectively acting as a server which proved to be hard to implement, susceptible to deadlocks and induced scalability limitations.

omniORB allows to easily configure the automatic shutdown of connections after a period of inactivity for incoming and outgoing connections by setting `inConScanPeriod` and `outConScanPeriod` options. The heartbeat polling rate combined with the connection shutdown timeout allows to effectively switch from connection-oriented system (where the connections remain open at all times) to connection-less system with short-lived TCP/IP connections (similarly to classic HTTP requests). For example, if average task duration is 15 seconds, it makes sense to keep the connections open all the time to avoid the overhead of frequent connection trashing. On the other hand, if average task duration is 1 hour, it may make sense to set heartbeat rate to 30 minutes and `outConScanPeriod, inConScanPeriod` to 1 second to immediately shutdown the connection (which would otherwise remain idle). This may be easily configured on per-run basis and it allows to control the number of concurrently open TCP/IP connections and increase scalability.

Sophisticated server-side thread management options are provided by omniORB to control efficiency and resource usage trade-offs. Concurrent calls from the same client may be multiplexed using the same connection. A separate server thread may be dedicated to each incoming connection or a thread pool of configurable size may be used. The server may also automatically switch to a thread pool mode if too many connections arrive. For example, if a worker pool is small, the requests may be efficiently handled by dedicated threads. When a worker pool grows, the requests are queued and served in a thread pool. The configuration and runtime handling of threads and connections is completely transparent to the upper layers of the system.

4.5.5 Security

DIANE provides flexible authentication and authorization options. Through omniORB options the RunMaster may be configured to advertise its service using Unix domain or

TCP/IP sockets. Standard transport protocols may be easily combined with SSL and Grid Security Infrastructure (GSI) [65]. With the GSI-based security, the RunMaster creates GSI-secured listening endpoint using user's grid proxy certificate, while the Worker Agents submitted to the grid retain their copy of a proxy certificate as provided by standard middleware. This mechanism is fully compliant with recommended security practices in the grid.

Additionally, multiple network transports may be specified for the same RunMaster service. This allows the WorkerAgents to connect from a local, trusted network with different authentication and authorization rules than are applied to the WorkerAgents running in remote, untrusted locations. Rules for accepting incoming connections may also include the IPv4 address specification with subnet mask bit selection. The endpoint configuration may also be used to prioritize the networks, such that connections over a fast network may be preferred to other networks.

4.5.6 Pre-defined components

Several application-level components which encapsulate most common use-cases are available for a direct use. They include SimpleTaskScheduler, SimpleApplicationManager and Executable application handler. We provide a brief functional overview of these components to illustrate the architecture of the system and existing configuration options.

SimpleTaskScheduler provides scheduling capabilities for sets of independent tasks processed in a self-scheduling mode. The scheduler uses a flat task queue to dispatch the tasks on the first-come first-served basis. When the queue is empty, the run terminates. The scheduler may automatically reschedule failed or lost tasks. No assumptions are made about the order in which these tasks are rescheduled. A worker time limit may be set, to control the worker node time usage. A particular worker which consistently fails to execute the tasks may be automatically black-listed and removed from the pool. The detailed list of policy options may be found in online reference documentation of DIANE.

SimpleApplicationManager allows to specify application-specific splitting and merging capabilities and customize the run termination criteria. In the simplest case the `initialize()` method is used to split the initial workload by creating a static list of tasks which is then passed on to the scheduler. The `tasks_done()` method is called whenever tasks complete and it may implement on-the-fly post-processing such as output merging. The `has_more_work()` method is called periodically and it may be used to determine application-specific conditions to terminate the run. The `finalize()` method may be used for final post-processing and cleanup.

An advanced application may take advantage of dynamic task splitting at runtime. This may be achieved by deriving a specific ApplicationManager from a BaseThread class, which automatically turns the ApplicationManager into an active object with its own control thread. ApplicationManager may take advantage of parallel post-processing in a thread-pool in the cases when sequential post-processing of tasks would lead to bottlenecks.

ExecutableApplication component is provided for easy integration of legacy appli-

```

1 # this file is given to the RunMaster as argument
2 # SimpleTaskScheduler is used by default.
3
4 # tell DIANE that we are just running executables
5 from diane_test_applications import ExecutableApplication as application
6
7 # the run function is called when the master is started
8 # input.data stands for run input parameters
9 def run(input, config):
10     d = input.data.task_defaults # a convenience shortcut
11
12     # all tasks will share these default parameters
13     d.input_files = ['hello']
14     d.output_files = ['message.out']
15     d.executable = 'hello'
16
17     # here are tasks differing by arguments to the executable
18     for i in range(20):
19         t = input.data.newTask()
20         t.args = [str(i)]

```

Figure 4.9: Listing of a run file with ExecutableApplication. A static list of tasks is created at the beginning of a run in a user-defined loop in the `run()` function.

cations. It allows to specify executable tasks with input and output files and variable executable arguments. It extends SimpleApplicationManager and may be directly used with the default scheduler. It provides resource usage monitoring on the worker nodes, including CPU and wall-clock times. It handles transfer of files of arbitrary sizes using a built-in FileTransferService, including the standard output and standard error files. The usage of ExecutableApplication is illustrated in Fig. 4.9.

4.6 The Ganga resource access API and user interface

GANGA is a user-centric tool that allows easy interaction with heterogeneous computational environments, configuration of the applications and coherent organization of jobs [136]. The implementation uses an object-oriented design in PYTHON. GANGA provides uniform access to remote resources and high-level user interface for application configuration and job management. In the context of DIANE, GANGA is used to submit and manage WorkerAgent jobs. It may also be used as a simplified bookkeeping environment for DIANE tasks and assist user in application configuration.

GANGA provides a simple but flexible programming interface that can be used either interactively at the PYTHON or IPYTHON [151] prompt, through a Graphical User Interface (GUI) or programmatically in scripts. This reflects the different working styles in different user communities and addresses various usage scenarios such as using the GUI for training new users, the command line to exploit advanced use-cases, and scripting

for automation of repetitive tasks.

The concept of a *job* component is essential as it contains the full description of a computational task, including: the code to execute; input data for processing; data produced by the application; the specification of the required processing environment; post-processing tasks; and metadata for bookkeeping. GANGA keeps track of all jobs and their status through a repository that archives all information between independent GANGA sessions. It is possible to switch between executing a job on a local computer and executing on the Grid by changing a single parameter of a job object. This simplifies the progression from rapid prototyping on a local computer and small-scale tests on a local batch system, to the analysis of a large dataset using Grid resources.

It is possible to make GANGA available to a user community with a high level of customization. For example, a domain expert can implement a custom application class describing the specific computational task. The class will encapsulate all low-level setup of the application, which is always the same, and only expose a few parameters for configuration of a particular task. The plugin system provided in GANGA means that this expert customization will be integrated seamlessly with the core of GANGA at runtime, and can be used by an end user to process tasks in a way that requires little knowledge about the interfaces of Grid or batch systems. Issues such as differences in data access between jobs executing locally and on the Grid are similarly hidden.

GANGA is a user- and application-oriented layer above existing job submission and management technologies, in Globus⁴, Condor [170], Unicore [168] or gLite [113]. Rather than replacing the existing technologies, GANGA allows them to be used interchangeably, using a common interface as the interoperability layer.

GANGA may be used as a job management system integrated into a larger system. In the case of User-level Overlay GANGA acts as an API for job submission and control.

4.6.1 Architecture and functionality

Fig. 4.10 shows the architecture of GANGA where the three user interfaces are built on top of the GANGA Public Interface (GPI) which in turn provides access to the GANGA core implementation.

A job in GANGA is constructed from a set of components. All jobs are required to have an application component and a backend component, which define respectively the software to be run and the processing system to be used. Many jobs also have input and output dataset components, specifying data to be read and produced. Computationally intensive jobs may have a splitter component which provides a mechanism for dividing into independent subjobs, and a merger component, which allows the aggregation of subjob outputs. The overall component structure of a job is illustrated in Fig. 4.11.

By default, the GPI exposes a simplified, top-level view suitable for most users in their everyday work, but at the same time allows the details of underlying systems to be exposed if needed. GANGA monitors the evolution of submitted jobs and categorizes them into the simplified states *submitted*, *running*, *completed*, *failed* or *killed*.

All job objects are stored in a job repository database, and the input and output

⁴The Globus Alliance, <http://www.globus.org>

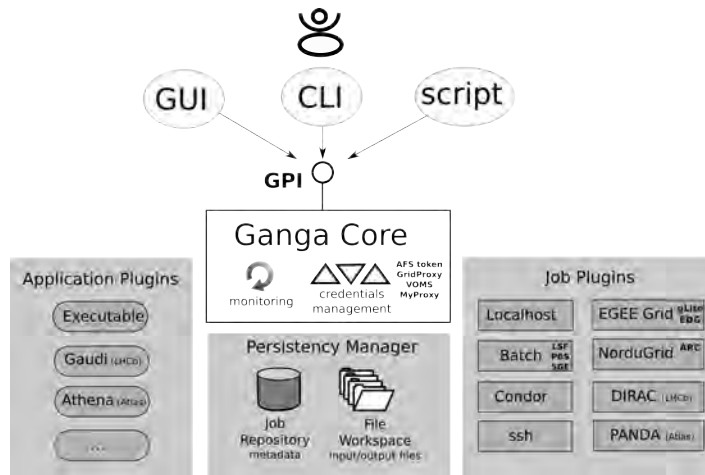


Figure 4.10: Architecture of GANGA. The user interacts with the GANGA Public Interface (GPI) via the Graphical User Interface (GUI), the Command-Line Interface in PYTHON (CLI), or scripts. Plugins are provided for different application types and backends. All jobs are stored in the repository.

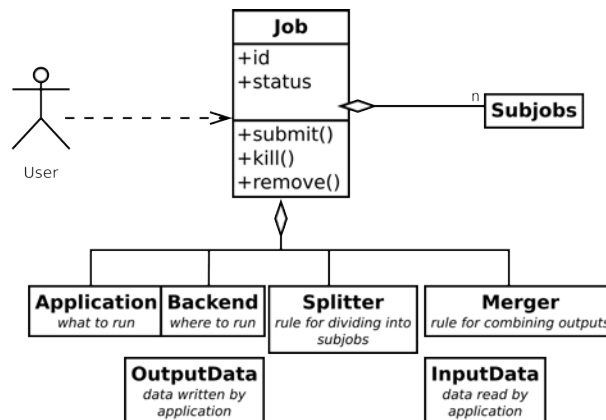


Figure 4.11: A set of classes in GANGA can be combined to form a complete job. The application to run and the backend where it will run are mandatory while all other components are optional.

files associated with the jobs are stored in a file workspace. Both the repository and the workspace may be in a local filesystem or on a remote server.

A large computational task may be split into a number of subjobs automatically according to user-defined criteria and the output merged at a later stage. Each subjob will execute on its own and the merging of the output will take place when all have finalized. The submission of subjobs is automatically optimized if the backend component supports bulk job submission. For example, when submitting to the gLite workload management system [113] the job collection mechanism is used transparently to the user.

Job splitting functionality provides a flat list of subjobs suitable for parallel processing of fully independent workloads. However, certain backends allow users to make use of more-sophisticated pluralization schemes, for example the MPI. In this case, GANGA may be used to manage collections of subjobs corresponding to MPI processes. Similarly the DIANE RunMaster may be specified as a GANGA backend with the server-side splitting and merging capabilities.

GANGA has built-in support for handling user credentials, including classic Grid proxies, proxies with extensions for VOMS (Virtual Organization Management Service) [11], and Kerberos [144] tokens for access to an Andrew filesystem (AFS) [132]. A user may renew and destroy the credentials directly using the GPI. GANGA gives an early warning to a user if the credentials are about to expire. The minimum credential validity and other aspects of the credential management are fully configurable.

GANGA supports multiple security models. For local and batch backends, the authentication and authorization of the users is based on the local security infrastructure including user name and network authentication protocols such as Kerberos. GSI provides for security across organizational boundaries for the Grid backends. Different security models are encapsulated in pluggable components, which may be simultaneously used in the same GANGA session.

Application components

The application component describes the type of computational task to be performed. It allows the characteristics and settings of some piece of software to be defined, and provides methods specifying actions to be taken before and after a job is processed. The pre-processing (configuration) step typically involves examination of the application attributes, and may derive secondary information. For example, intermediate configuration files for the application may be created automatically. The post-processing step can be useful for validation tasks such as determining the validity of the application output.

The simplest application component (**Executable**) has three attributes:

exe : the path to an executable binary or script;

args: a list of arguments to be passed to the executable;

env : a dictionary of environment variables and the values they should be assigned before the executable is run.

The configuration method carries out integrity checks – for example ensuring that a value has been assigned to the `exe` property.

Backend components

A backend component contains parameters describing the behavior of a processing system. The list of parameters can vary significantly from one system to another, but can include, for example, a queue name, a list of requested sites, the minimum memory needed and the processing time required. In addition, some parameters hold information that the system reports back to the user, for example the system-specific job identifier and status, and the machine where a job executed.

A backend component provides methods for submitting jobs, and for cancelling jobs after submission. It also provides methods for updating information on job status, for retrieving output of completed jobs and for examining files produced while a job is running.

Backend components have been implemented for a range of widely used processing systems, including: local host, batch systems (Portable Batch System (PBS) [88], Load Sharing Facility (LSF) [163], Sun Grid Engine (SGE) [72], and Condor [170]), and Grid systems, for example based on gLite [113], ARC [58] and OSG [155]. Remote backend component allows jobs to be launched directly on remote machines using ssh. DIANE backend enables the RunMaster server to be started up directly from GANGA.

As an example, the batch backend component defines a single property that may be set by the user:

`queue` : name of queue to which job should be submitted, queue being used if this left unspecified,

and defines three attributes for storing system information:

`id` : job identifier;

`status` : status as reported by batch system;

`actualqueue`: name of queue to which job has been submitted.

In addition, a remote-backend component allows a job defined in a GANGA session running on one machine to be submitted to a processing system known to a remote machine to which the user has access. For example, a user who has accounts on two clusters may submit jobs to the batch system of each from a single machine.

Dataset components

Dataset components generally define attributes that uniquely identify a particular collection of data, and provide methods for obtaining information about it, for example its location and size. The details of how data collections are described can vary significantly from one problem domain to another, and the only generic dataset component in GANGA represents a null (empty) dataset. Other dataset components are specialized for use with a particular application, and so are discussed later.

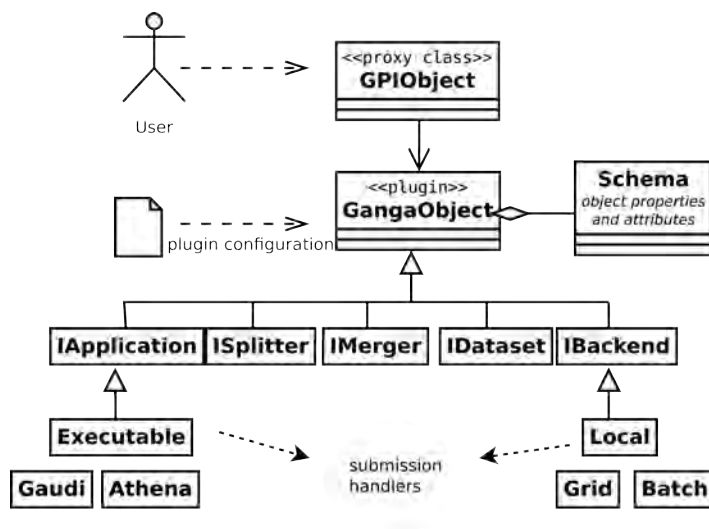


Figure 4.12: A component class implements one of the abstract interfaces corresponding to the different parts of a job. Runtime dependencies between application and backend classes are provided by submission handlers.

A strict distinction is made between the datasets and the sandbox (job) files. The former are the files or databases which are stored externally. The sandbox consists of files which are transferred from the user's filesystem together with the job. The sandbox mechanism is designed to handle small files (typically up to 10 MB) while the datasets may be arbitrarily large.

4.6.2 Implementation

Here we provide details of the actual implementation of some of the most important parts of Ganga.

Components

Job components are implemented as plugin classes, imported by Ganga at start-up if enabled in a user configuration file. This means that users only see the components relevant to their specific area of work. Plugins developed and maintained by the Ganga team are included in the main Ganga distribution and are upgraded automatically when a user installs a newer Ganga version. Currently, the list includes around 15 generic plugins and around 20 plugins specific to HEP data analysis. Plugins specific to other user communities need to be installed separately but could easily be integrated into the main Ganga distribution.

Plugin development is simplified by having a set of internal interfaces and a mechanism for generating proxy classes [71]. Component classes inherit from an interface

class, as seen in Fig. 4.12. Each plugin class defines a schema, which describes the plugin attributes, specifying type (read-only, read-write, internal), visibility, associated user-convenience filters and syntax shortcuts.

The user does not interact with the plugin class directly but rather with an automatically generated proxy class, visible in the GPI. The proxy class only includes attributes defined as visible in the schema and methods selected for export in the plugin class. This separation of the plugin and proxy levels is very flexible. At the GPI level, the plugin implementation details are not visible; all proxy classes follow the same design logic (for example, copy-by-value); persistence is automatic, session-level locking is transparent. In this way the low-level, internal API is separated from the user-level GPI.

The framework does not force developers to support all combinations of applications and backends, but only the ones that are meaningful or interesting. To manage this, the concept of a *submission handler* is introduced. The submission handler is a connector between the application and backend components. At submission time, it translates the internal representation of the application into a representation accepted by a specific backend. This strategy allows integration of inherently different backends and applications without forcing a lowest-common-denominator interface.

Most of the plugins interact with the underlying backends using shell commands. This down-to-earth approach is particularly useful for encapsulating the environments of different subsystems and avoiding environment clashes. In verbose mode, GANGA prints each command executed so that a user may reproduce the commands externally if needed. Higher-level abstractions such as JSDL [126] or OGSA-BES [16] are not currently used, but specific backends that support these standards could readily be added.

Job persistence

The *job repository* provides job persistence in a simple database, so that any subsequent GANGA session has access to all previously defined jobs. Once a job is defined in a GANGA session it is automatically saved in the database. The repository provides a bookkeeping system that can be used to select particular jobs according to job metadata. The metadata includes such parameters as job name, type of application, type of submission backend, and job status. It can readily be extended as required.

GANGA supports both a local and a remote repository. In the case of the former, the database is stored in the local file system, providing a standalone solution. In the case of the latter, the client accesses an AMGA [104] metadata server. The remote server supports secure connections with user authentication and authorization based on Grid certificates. Performance tests of both the local and remote repositories show good scalability for up to 10 thousand jobs per user, with the average time of individual job creation being about 0.2 seconds. There is scope for further optimization in this area by taking advantage of bulk operations and job loading on demand.

The job repository also includes a mechanism to support schema migration, allowing for evolution in the schema of plugin components.

Input and output files

GANGA stores job input and output files in a *job workspace*. The current implementation uses the local file system, and has a simple interface that allows transparent access to job files within the GANGA framework. These files are stored for each job in a separate directory, with sub-directories for input and output and for each subjob.

Users may access the job files directly in the file-system or using GANGA commands such as `job.peek()`. Internally, GANGA handles the input and output files using a simple abstraction layer which allows trivial integration of additional workspace implementations. Tests with a prototype using a WebDav [181] server have shown that all workspace data related to a job can be accessed from different locations. In this case, a workspace cache remains available on the local file system.

The combination of a remote workspace and a remote job repository effectively creates a roaming profile, where the same GANGA session can be accessed at multiple locations, similar to the situation for accessing e-mail messages on an IMAP [1] server.

Monitoring

GANGA automatically keeps track of changes in job status, using a monitoring procedure designed to cope with varying backend response times and load capabilities. Each backend is polled in a different thread taken from a pool, and there is an efficient mechanism to avoid deadlocks from backends that respond slowly. The poll rate may be set separately for each backend.

The monitoring sub-system also keeps track of the remaining validity of authentication credentials, such as Grid proxies and Kerberos tokens. The user is notified that renewal is required, and if no action is taken then GANGA is placed in a state where operations requiring valid credentials are disabled.

4.7 Heuristic resource selection

4.7.1 Submitters

Worker agents are managed as GANGA `Executable` job wrappers. The job wrapper downloads and installs DIANE on-the-fly in the worker node and spawns a Worker-Agent process which connects to the RunMaster. A `Submitter` helper class handles preparation and submission of worker agent wrappers, including low-level details such as inclusion of the master server address, which is encoded as CORBA IOR, into the worker wrapper input sandbox. The helper makes the system not only easy to use but also easy to extend with customized submitter scripts. Fig. 4.13 shows a complete implementation of a GANGA submitter script for the LSF batch system. A simple substitution of a backend object allows to reuse the same script to transparently submit worker agents to the EGEE Grid or any other infrastructure supported by GANGA, including explicitly selected worker nodes in a local computing center via ssh. Shell commands are available to a user for easy submission of WorkerAgents via a selected GANGA backend.

```

1  #!/usr/bin/env ganga
2  ##-python--
3
4  from diane.submitters import Submitter
5
6  submitter = Submitter()
7  submitter.download=False
8  submitter.parser.add_option("--delay" ,...)
9  submitter.parser.add_option("--queue" ,...)
10 submitter.initialize()
11
12 import time
13
14 for i in range(submitter.options.NWORKERS):
15     j = Ganga.GPI.Job()
16     j.backend=Ganga.GPI.LSF(queue=submitter.options.queue)
17     submitter.submit_worker(j)
18     if submitter.options.delay:
19         time.sleep(submitter.options.delay)

```

Figure 4.13: LSF submitter helper (`LSFSubmitter.py`). The actual worker agent submission is handled by GANGA, according to the specified backend.

4.7.2 Simple Agent Factory

In typical cases of Capacity Computing, the lifetime of individual worker agent is much shorter than the makespan of a run. As the number of available resources in the pool decreases with time the submission of new worker agents is needed in the course of the run to maintain the worker pool.

Workers may be easily submitted by hand by a user, but that may require frequent manual operation and may be time consuming and inefficient. Agent Factory is a component which automates the submission process.

The aim of the Agent Factory is to maintain the number of active workers as high as possible but not greater than a specified number. As an example, in case of a sudden drop in the number of workers one would like to react by submitting new workers to replenish the worker pool. On the other hand the submission of new worker agents should be kept under control and on par with the number of available resources in the grid at a given moment. In particular, if not enough free resources are available, then an excessive, continuous submission would have little effect on the speedup of the system but could lead to overloading of grid services.

Simple Agent Factory, shown in Fig. 4.14, periodically checks the size of the worker pool reported by the RunMaster and fills up the pool by submitting more workers to the grid. User defined job resource requirements, such as specified in the GLUE schema [17], may be used to guide the resource selection which is delegated to the grid WMS.

This very simple solution may, however, lead to suboptimal resource selection as it does not take into account runtime failures in the worker nodes. Another common problem is that users are often not able to efficiently define detailed job resource requirements due to abundance of configuration parameters and heterogeneity of the grid environ-

```

1 # worker agent submitter helper with the user-defined
2 # resource requirements. For historic reasons
3 # EGEE Grid is aliased to LCG.
4 submitter = LCGSubmitter(requirements)
5
6 # Ganga job repository (list of worker agents in the pool)
7 jobs = Ganga.GPI.jobs
8
9 while 1:
10     # current size of the worker pool
11     # obtained via a RPC query to the RunMaster
12     current_pool_size=runmaster.get_worker_pool_size()
13
14     # submit the workers to fill up the pool size
15     # choice of CE is left to Grid WMS, based on resource requirements
16     for i in range(N_WORKERS-current_pool_size):
17         submitter.submit_worker(Ganga.GPI.Job())

```

Figure 4.14: Simple Agent Factory algorithm implemented in GANGA. Simple Agent Factory uses GANGA GPI interface to manipulate worker agent jobs and DIANE RPC interface to query the run master. `N_WORKERS` is a size of the worker pool set by the user.

ment. Specific requirements may include hardware and software parameters, resource allocation policy constraints and software dependencies which may also be site-specific and thus impossible to define globally.

4.7.3 Heuristic Agent Factory

Heuristic Agent Factory (HAF) component was designed to optimize the resource selection, based on system feedback using recent performance data. This is accomplished through a resource selection algorithm in which the grid sites are ranked based on their reliability and performance over time. The algorithm is designed to cope with the observed dynamics of the grid where the number of available computing resources is variable in time. Typical cases include sites entering downtime periods and stopping accepting jobs, or sites going back into production after configuration fixes, hardware upgrades etc. The selection algorithm gives more weight to more recent performance data and eventually “forgets” old data. It allows to increase the submission success rate and to maintain the number of worker agents on a predefined level.

Selection algorithm

In EGEE Grid, distributed computational clusters or batch farm are represented by *Computing Elements* (CEs). CE is the smallest management unit for the resource selection algorithm. HAF submits jobs to the CEs via the *Workload Management System* (WMS) which is used as a grid gateway.

The core of HAF is a non-deterministic selection procedure based on a fitness al-

gorithm commonly found in genetic algorithms/evolutionary strategies [131]. When a new worker agent is submitted to the grid, a CE is chosen randomly with probability proportional to its fitness. CEs are selected by the HAF and the WMS simply forwards the job submission requests. The outline of the algorithm is presented in Fig. 4.15 and it shows the actual PYTHON code from the HAF implementation.

For a Computing Element with n total jobs, r jobs currently running and c jobs completed without errors, the $fitness(CE)$ is defined as

$$fitness(CE) = \frac{r + c}{n} \quad (4.1)$$

The fitness value lies in the $[0..1]$ interval. The $fitness = 1$ represents a reliable CE with all workers either running or finished cleanly. If all workers are queuing in a CE or if a CE is unable to correctly execute any jobs for the application then $fitness = 0$.

A *generic CE slot* corresponds to a random CE selected by the WMS and is used for discovery and adaptive ranking of CEs. The $fitness$ of the generic slot is always 1. At bootstrap the list of known CEs is empty and all jobs are submitted via the generic slot. As the list of known CEs grows the HAF keeps on using the generic slot to submit a small fraction of jobs (`MAX_PENDING` parameter) random sites to detect the availability of new resources or an improvement in the performance of CEs with low fitness.

For each CE the probability to be selected by HAF is

$$P(CE) = \frac{fitness(CE)}{1 + \sum fitness(CE_i)} \quad (4.2)$$

The denominator is the total fitness of the population of known CEs, where 1 represents the generic CE slot.

Resources discovery and balancing

Initially the list of known, available CEs is empty. While it is possible to directly query the grid information services to obtain a list of all available Computing Elements, this is of little help since the selection algorithm needs historical data to calculate the fitness. However, this problem does not exist in algorithms which uses the generic slot where the WMS makes a few initial CE choices.

At the beginning, the generic slot is the only resource available and the only candidate for selection. Computing Elements are chosen automatically by the WMS and HAF stores this information. Eventually, with enough data collected, it takes over the decision process. As the number of discovered Computing Elements grows, the likelihood of the generic slot being selected for submission decreases. However, the fitness of the generic slot is permanently fixed at 1, and thus the probability of selecting it never reaches 0. This is important to maintain balance: when known Computing Elements perform well, few or no workers are sent to the grid via the generic slot and new workers are distributed to known Computing Elements. When known Computing Elements start under-performing and the fitness of the population decreases, we can expect the generic slot to be chosen more often, giving WMS a chance to find new resources. This way, we avoid overloading of Computing Elements with excessive resource requests.

```

1  # worker agent submitter helper
2  submitter = LCGSubmitter()
3
4  # Ganga job repository (list of worker agents in the pool)
5  jobs = Ganga.GPI.jobs
6
7  # return jobs which belong to a given CE only (note: by lexical closure)
8  def filter_CE(job_list):
9      return [j for j in job_list if j.backend_actualCE == CE]
10
11 while 1:
12     # current size of the worker pool
13     # obtained via a RPC query to the RunMaster
14     current_pool_size=runmaster.get_worker_pool_size()
15
16     # build the set of all known CEs and prune old jobs
17     knownCEs = set()
18
19     for j in jobs:
20         if too_old(j):
21             j.remove()
22         else:
23             knownCEs.add(j.backend.actualCE)
24
25     # calculate fitness
26     for CE in knownCEs:
27         n = len(filter_CE(jobs))
28         c = len(filter_CE(jobs.select(status='completed')))
29         r = len(filter_CE(jobs.select(status='running')))
30         fitness[CE] = float(c + r) / n
31
32     total_fitness = sum(fitness)
33
34     #allow a small oversubmission of workers (MAX_PENDING)
35     total_pending = jobs.select(status='submitted')
36
37     # submit workers if pool size smaller than N_WORKERS
38     while current_pool_size < N_WORKERS and len(total_pending) < MAX_PENDING:
39
40         #select the CE using fitness-proportional method
41         r = random.uniform(0, total_fitness + 1)
42         for i in range(len(fitness)):
43             r -= fitness[i]
44             if r <= 0:
45                 CE = knownCEs[i]
46                 break
47         submitter.submit_worker(Ganga.GPI.Job(), CE)

```

Figure 4.15: Heuristic Agent Factory (HAF) algorithm implemented in GANGA. HAF uses GANGA GPI interface to manipulate worker agent jobs and DIANE RPC interface to query the run master. `N_WORKERS` is a size of the worker pool set by the user. `MAX_PENDING` controls the job submission redundancy.

Discussion

The added value of the HAF is that it ranks available resources as a function of the current performance for the *specific application*. The HAF is an efficient way to automate resource provisioning without overloading the system with unnecessary submissions, hence to maximize the overall duty cycle of user's application.

It is important to note that functionality of the HAF and that of WMS are complementary. WMS provides global workload balancing between Computing Elements while the role of the HAF is the selection of the most compatible resources in a context of a particular application.

An arbitrary subset of job resource requirements may be easily defined by the user for the generic grid slot of the HAF to constrain the set of Computing Elements for the resource selection algorithm. HAF works on a higher level than the WMS itself and thus all WMS configuration parameters and job requirements are automatically taken into account. However, if job requirements are absent or poorly defined the HAF is still able to dynamically adapt in a non-parametric way to operating within the constraints of the application.

HAF algorithm is also application-agnostic as it does not require any application-specific knowledge and relies solely on the worker agent job status information provided by the underlying grid system.

The evaluation of HAF is provided in the context of a specific application and is presented within Capacity Computing Case Study in Chapter 6.

4.8 Adaptive workload balancing

In Chapter 3 we showed that the late-binding U_0 model defines the maximal theoretical speedup with ideal load balancing, which is also a limit for the U_1 discrete model for small task sizes. We also concluded that the runtime overheads should not exceed 10% of the task execution time. This constraints the task splitting granularity for applications where communication overheads become large. Additionally, in grid environment, worker nodes may be shared among multiple jobs. This may result in varying performance characteristics (for example the CPU load) of the processors and, for longer tasks and simple self-scheduling strategy, may lead to suboptimal schedules.

A number of semi-automatic load balancing methods have been developed (e.g. diffusion self-balancing mechanism, genetic networks load regulation, simulated annealing technique, bidding approaches, multi-parameter optimization, numerous heuristics, etc.), but most of them suffer one or another serious limitation, most noticeably the lack of flexibility, high overheads, or inability to take into consideration specific features of the application. To overcome these limitations we have developed a hybrid approach where the balancing decision is taken in interaction of the application with the execution environment. In this approach the adaptive workload balancing algorithm (AWLB), developed in [105], is applied at the task scheduling level. The resource pool is maintained by a Hybrid Agent Factory, which uses the information provided by the scheduler and application benchmarks, to rank and select the resources.

4.8.1 The AWLB algorithm

In self-scheduling all the workload is divided into tasks of equal size. As soon as a worker becomes available, it is assigned the next task from the task queue. In AWLB, the size of the task assigned to each worker is calculated by the heuristic algorithm, using the resource and application characteristics. Applying dynamic splitting, a scheduler may dynamically choose the task size such that the sum of task execution time and communication overheads is equal for all tasks executed on all processors.

The AWLB provides an optimal distribution of the divisible workload between participating processors according to the computing environment characteristics and the application requirements. The suitability of resources is determined by the application requirements; for traditional parallel computing applications considered here as a test case, it depends on the processing power and network connectivity correlated with the application communication to computation ratio. Thus the main parameters that define a parallel application performance are:

- The application parameter $f_c = N_{comm}/N_{calc}$, where N_{comm} is the total amount of application communications, i.e. data to be exchanged (measured in bit) and N_{calc} is the total amount of computations to be performed (measured in Flop);
- The resource parameters $\mu_i = p_i/n_i$, where p_i is the available performance of the i_{th} processor (measured in Flop/s) and n_i is the network bandwidth to this node (measured in bit/s).

The AWLB algorithm is based on the benchmarking of the available resources capacity, defined as a set of individual resource parameters $\mu = \mu_i$, and experimental estimation of the application parameter f_c . The value of the application parameter f_c is determined by running through the space of possible f_c and finding the value f_c^* which provides minimal runtime of the application on this set of resources. The algorithm is described in more detail in [106, 108].

4.8.2 Hybrid approach to workload balancing

The application may comprise heterogeneous tasks executed at different times with different performance characteristics so the total application performance requirements (f_c) may vary at runtime. Similarly the capacity of grid resources (μ) may vary with time due to inherent grid dynamics. The scheduler may respond to changing application or resource conditions and more suitable resource set may be selected to execute the application at a given time. This may happen at the individual task execution boundary or at other natural boundaries specific to the application model (for example at each iteration for the iterative simulations). This is a distinctive feature, in contrast to the static parallel programs where resources are allocated once and fixed during the execution unless special migration libraries are used such as the Dynamite [95].

We illustrate the hybrid workload balancing approach for an iterative computation pattern where each iteration consists of a set of independent tasks and output of one iteration is used as an input for the next iteration. This pattern is typical for many

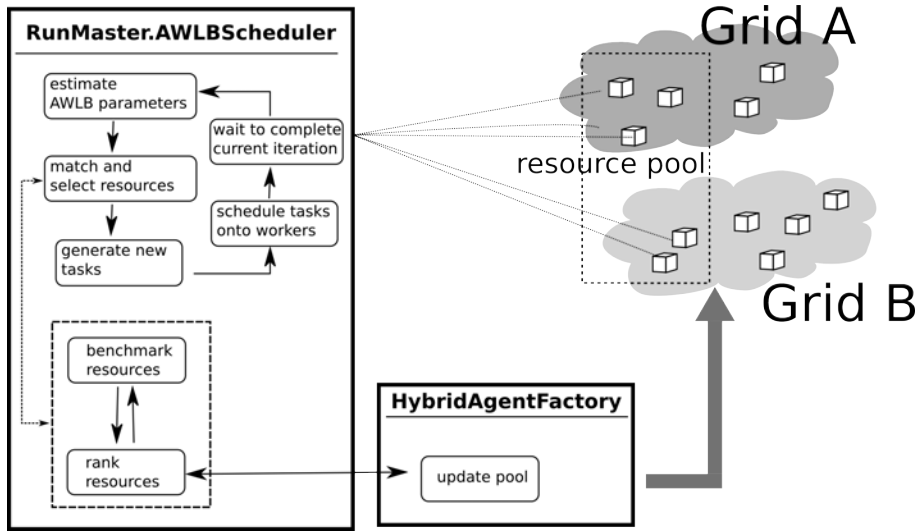


Figure 4.16: Iterative execution of parallel application in the integrated AWLB in User-level Overlay with dynamic resource pool.

time-based optimization algorithms such as SWAT-CUP [5] used for calibration and uncertainty analysis of distributed watershed models in Earth Science applications.

Hybrid workload balancing approach is shown in Fig. 4.16. The application consists of a set of parallel tasks that process the workload scheduled by the RunMaster. The RunMaster collects the information about the available resources and monitors the application responses. At each iteration of the application algorithm the distribution of the workload is re-evaluated on a updated set of resources, and the AWLB parameters are re-estimated. The scheduler uses AWLB benchmarks to rank the resources and generates tasks which duration is proportional to the capacity μ of the resources to minimize the execution time of the iteration. If more resources are available in the pool than required by the optimal schedule only the best subset is selected and used. If runtime conditions, such as CPU load or network bandwidth, change on the worker node, the resource rank in the subsequent iteration is modified.

The selection of suitable resources from available worker pool is performed by the AWLB scheduler. The same information may be used to guide the process of updating the pool which is performed asynchronously by the Agent Factory. For example, the Heuristic Agent Factory described in previous section may be extended such that a *Hybrid* Agent Factory may use the resource ranking information provided by the AWLB scheduler to complement the fitness-proportional ranking of computing elements. The fitness of each computing element may be first determined by the plain heuristic algorithm described in the previous section and then be linearly scaled by the AWLB rank to select the most efficient subset of CEs. This approach is most effective when a benchmark of a single worker node is representative for other nodes in the same farm. This is to a good approximation true for CEs which consist of homogeneous nodes.

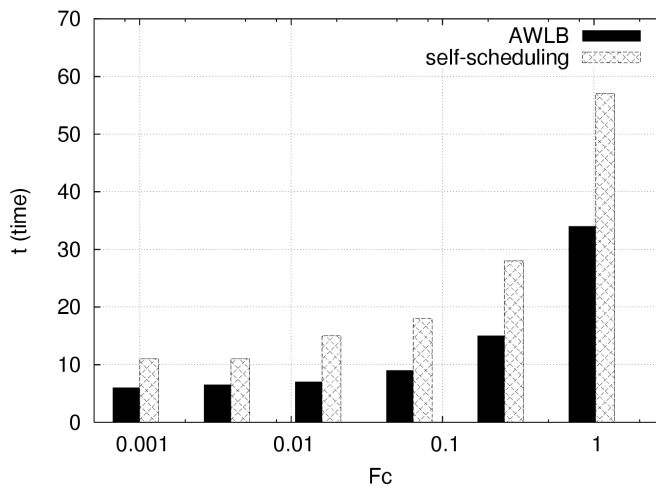


Figure 4.17: Execution time with self-scheduling and AWLB load balancing for six applications characterized by different communication/computation ratio F_c .

4.8.3 Experimental results

The experimental results were achieved for a model application with a synthesized workload and tunable parameter f_c [107], where the task input and output consisted of sequences of random bytes, and the task execution consisted of loops performing a set of standard floating point and integer operations to generate the CPU load. The number of loop cycles was used to control the N_{calc} parameter, whereas the length of input and output sequences was used to control the N_{comm} parameter. The f_c ratio was chosen correspondingly to typical values observed in computing-intensive calculations of Feynman-loops in theoretical physics. A pool of EGEE worker nodes was fixed in each test run.

Fig. 4.17 demonstrates the results for six application workloads, characterized by different communication/computation ratio F_c . In these experiments the computational load (N_{calc}) was kept constant on a fixed set of 16 processors, while the amount of data transferred between the master and the workers (N_{comm}) was varied. The execution time with the AWLB algorithm was up to 2 times lower than self-scheduling for all types of simulations.

Thorough testing of different applications on various sets of resources showed a strong influence of the level of resource heterogeneity on the results achieved. A series of targeted experiments was performed varying the resource heterogeneity in the processor power and the network links bandwidth to calculate the load-balancing speedup θ defined as

$$\theta = \frac{T_{non-balanced}}{T_{balanced}} \quad (4.3)$$

where $T_{non-balanced}$ is the execution time of the parallel application without the load-

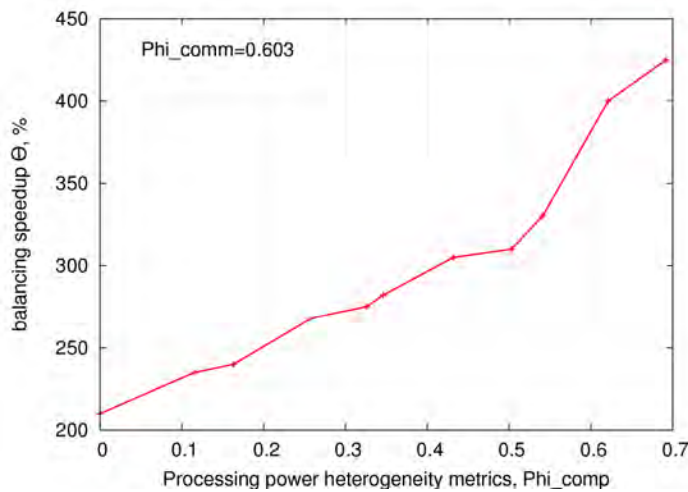


Figure 4.18: Dependency of the load balancing speedup on resource heterogeneity metrics Φ .

balancing, and $T_{balanced}$ is the execution time using load-balancing on the same set of resources. As a sample of these tests, Fig. 4.18 shows the dependency of the load balancing speedup on the processing power heterogeneity metrics Φ , defined as a standard deviation of a set of normalized parameters μ

$$\Phi = \sqrt{\frac{1}{N-1} \sum 1 - \frac{\mu}{\mu_{avg}}}. \quad (4.4)$$

As it may be seen, the speedup grows super-linearly with the heterogeneity level, thus indicating that this approach is especially beneficial for strongly heterogeneous resources, such as grids.

4.9 Summary

We outlined the key architectural principles and functional areas of the User-level Overlay, which are essential to tackle the main challenges of using grids and other distributed environments for processing of scientific tasks. While late binding is the key to mastering of the inherent dynamics in large computing grids, its efficiency may be further improved with strategies such as heuristic resource selection and adaptive workload balancing. This is more easily achieved in an architecture which decouples resource selection from scheduling.

We presented the detailed design of essential software components, and the key features of the implementation in an attempt to create a blueprint with software design and engineering guidelines for building support systems for processing of distributed

tasks in a general context. One of the key aspects is the development of the User-level Overlay as a *hosting environment* for application components, and for scheduling and resource selection algorithms. This enables incremental adaptability to increasingly complex requirements.

The key features of the User-level Overlay strategy proposed in this Chapter include:

- late-binding task processing model,
- ability to interface to a wide range of distributed systems,
- ability to extend and customize the system to cover application-specific scheduling and processing patterns,
- ease of use and lightweight deployment in the user space.

The User-level Overlay concept is universal and may be used above a variety of distributed infrastructures. Software plugins to more than 10 different computational backends have been so far implemented, what provides a good indication of practical importance of this approach. An essential question at this point is, whether the ideas and concepts used to develop the User-level Overlay may stand up to expectations with real-life applications. This is considered in the next Chapter.

CHAPTER 5

User-level Overlay in action

Progress comes from the intelligent
use of experience.

E.Hubbard

In this Chapter¹ 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.

¹The results described in this Chapter formed the basis of the following papers: J. Mościcki, S. Guatelli, A. Mantero, and M. Pia. Distributed Geant4 simulation in medical and space science applications using DIANE framework and the Grid. *Nuclear Physics B - Proceedings Supplements*, 125:327 – 331, 2003; S. Chauvie, P. Lorenzo, A. Lechner, J. Mościcki, and M. Pia. Benchmark of medical dosimetry simulation using the Grid. In *IEEE Nuclear Science Symposium Conference Record NSS '07*, volume 2, pages 1100–1106, 2007; S. C. Pop, T. Glatard, J. Mościcki, H. Benoit-Cattin, and D. Sarrut. Dynamic partitioning of GATE Monte-Carlo simulations on EGEE. *J. Grid Computing*, 8(2):241–259, 2010; J. Mościcki, F. Brochu, J. Ebke, U. Egede, J. Elmsheuser, K. Harrison, R. Jones, H. Lee, D. Liko, A. Maier, A. Muraru, G. Patrick, K. Pajchel, W. Reece, B. Samset, M. Slater, A. Soroko, C. Tan, D. van der Ster, and M. Williams. Ganga: A tool for computational-task management and easy access to Grid resources. *Computer Physics Communications*, 180(11):2303 – 2316, 2009.

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 parallellizing 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 than 6×10^5 SLOC² 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, π^+ , π^- , 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 (B_i) and 85 worker agents in the late-binding mode (A_i). Irregular job completion rate in the early-binding mode (B_1 and B_3) results from short-term variations of the grid dynamics. Moreover, job execution may be arbitrarily detailed what leads to incomplete runs

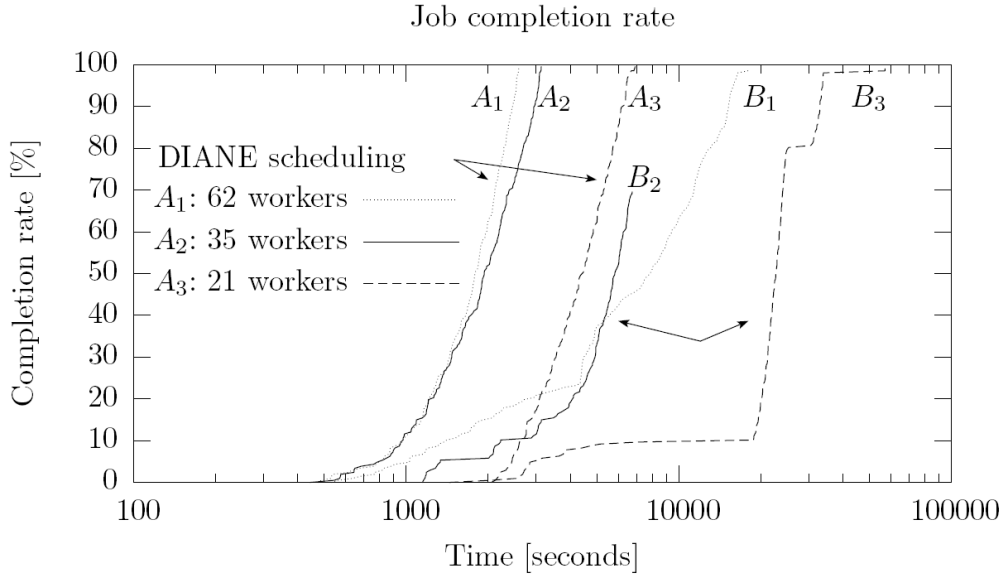


Figure 5.1: Comparison of task completion rate of late-binding scheduling based on DIANE (A) and early-binding scheduling (B). The figure shows three selected runs with typical behavior, for 85 submitted jobs. A number of effective workers in A is lower than the number of submitted jobs and is indicated in the figure.

(B_2). 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 a 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

```

1
2 /// Application class prototype which implements the high-level API used
  by the WorkerAgent process.
3
4 class BrachySimulation : virtual public DIANE::IG4Simulation
5 {
6 public:
7     BrachySimulation(G4int);
8     ~BrachySimulation();
9
10    void setSeed(G4int seed);
11    G4bool initialize(int argc, char** argv);
12    void executeMacro(std::string macroFileName);
13    std::string getOutputFilename();
14    void finish();
15
16 private:
17     G4int seed;
18     G4RunManager* pRunManager;
19 };
20
21
22 //// This is the entry point for loading the application via a shared
  library.
23 extern "C"
24 DIANE::IG4Simulation* createG4Simulation(int seed)
25 {return new BrachySimulation(seed); }

```

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

```

1 int main(int argc, char** argv)
2 {
3     // command-line arguments
4
5     G4String macrofile = argv[1];
6     G4int seed = atoi(argv[2]);
7
8     BrachySimulation *simulation = new BrachySimulation(0);
9
10    simulation -> initialize(argc, argv);
11    simulation -> setSeed(seed);
12    simulation -> executeMacro(macrofile);
13    simulation -> finish();
14
15    delete simulation;
16 }

```

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 ± 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

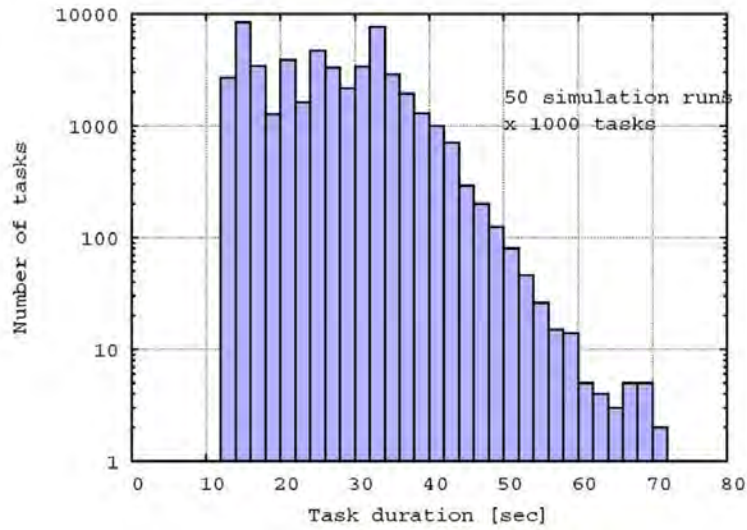


Figure 5.4: Histogram of task duration for brachytherapy simulation on EGEE Grid.

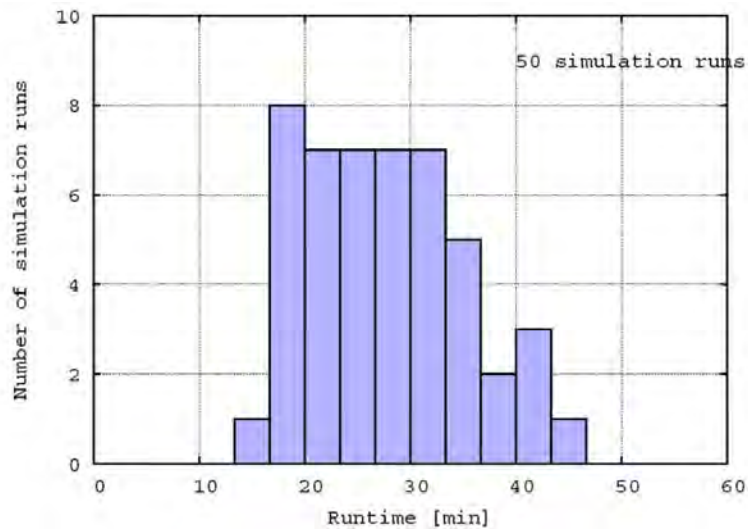


Figure 5.5: Histogram of overall simulation time (makespan) for brachytherapy simulation on EGEE Grid.

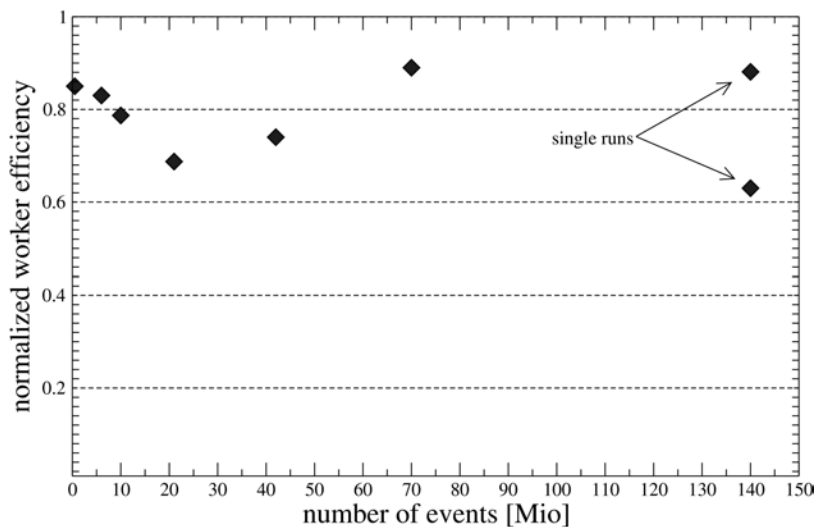


Figure 5.6: Normalized, average worker efficiency for large Geant4 simulation using explicit worker placement with ssh in an interactive cluster. Processors shared by many users at the same time.

$\frac{t_s}{t_c}$ 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

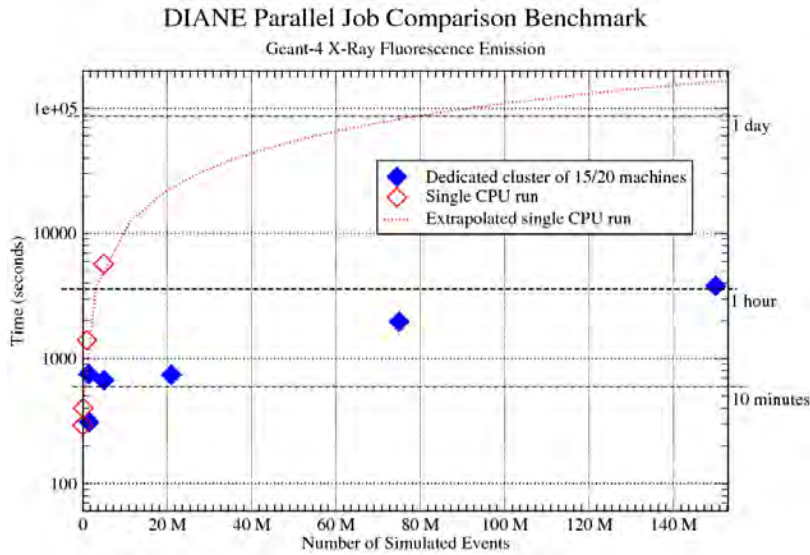


Figure 5.7: Average execution times for large Geant 4 simulation using explicit worker placement with ssh in an interactive cluster. Processors shared by many users at the same time.

engine provides a steering logic and GANGA/DIANE overlay serves as a task processing engine.

The workflow engine combined with GANGA/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.

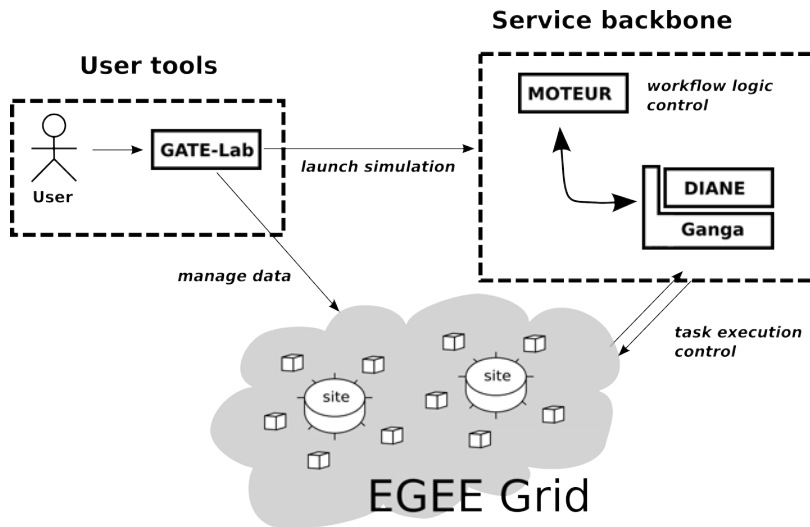


Figure 5.8: Architecture of a virtual laboratory for medical image simulation and analysis. User tools provide transparent grid data management, application configuration, and launching environment. Service backbone integrates the MOTEUR workflow execution engine with GANGA/DIANE overlay and provides a task processing service above grid computing infrastructure.

5.2.2 Results

The virtual laboratory service is operated at the Creatis research facility in Lyon³. From July 2009 to August 2010 more than 360 DIANE RunMaster instances were activated in the service backbone which handled 58×10^3 worker agent jobs completing more than 113×10^3 simulation tasks.

In addition, several targeted experiments were performed to assess the dynamic, workflow-based partitioning and simulation steering using GANGA/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 20×10^6 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

³Creatis is a part of *Centre National de la Recherche Scientifique, France*.

retrieved simulation results. On the other hand the GANGA/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/GANGA 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 LHC***b*** experiments

The ATLAS and LHC***b*** 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 LHC***b*** 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 LHC***b*** Grid jobs are routed through the DIRAC [176] workload management system (DIRAC WMS). In both experiments GANGA is used as the primary user interface and workload management is implemented by experiment-specific systems. Usage of GANGA 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

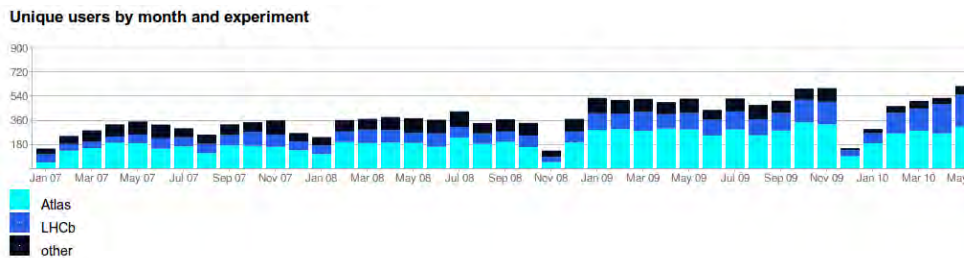


Figure 5.9: Evolution of the number of GANGA users in LHC communities since 2007.

distributed infrastructures. Therefore, applications components defined in GANGA 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 GANGA 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 GANGA 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 4×10^5 Grid jobs in ATLAS, and more than 3×10^5 Grid jobs in LHCb were submitted by end users with GANGA. Since 2007 1930 users in ATLAS and 630 users in LHCb were recorded⁴. 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 GANGA 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 GANGA. In contrast, individual researchers or developers in the context of rapid prototyping activities may prefer to use generic application components. In such cases, GANGA 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 GANGA to run the GARFIELD [178] simulation program on the Grid. A GANGA script has been written that generates a chain of simulation jobs using the GARFIELD generator of macro files and GANGA's **Executable** application component. The GARFIELD executables, and a few small input files, are placed in the input sandbox of each job. Histograms and

⁴data 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 GANGA 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 GANGA/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 GANGA/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 were 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 GANGA/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

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

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]

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 Platform⁵ (GAP) which allows biologists and other researchers to use a web-based, domain-specific portal to perform screening in distributed environments. The platform embeds GANGA 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.

⁵available at <http://gap.grid.sinica.edu.tw>

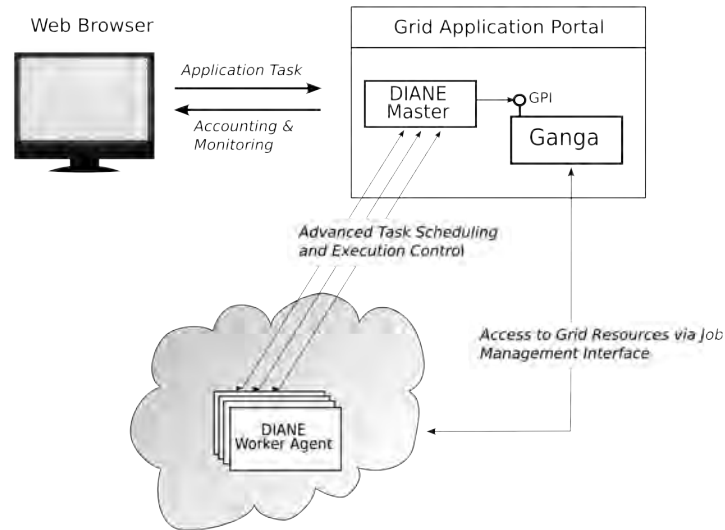


Figure 5.10: Architecture of Grid Application Portal with embedded DIANE servers and GANGA sessions. Source: <http://gap.grid.sinica.edu.tw>

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

⁶http://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 GANGA, 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.

Capability computing case study: ITU broadcasting planning

Grids, as federations of batch systems, are not designed to handle short-deadline, capability-computing applications. In this Chapter¹ we present a case study of an application for planning of the digital terrestrial broadcasting service for a large number of member states of the International Telecommunication Union (ITU²). We explain how the User-level Overlay was used to implement a dependable, just-in-time service based on the EGEE Grid where fast and reliable response is a key metric of success for a mission-critical application.

6.1 Introduction

The radio-frequency spectrum is a limited natural resource which must be used rationally, efficiently and economically, so that countries or groups of countries may have equitable access to it [96]. The analogue terrestrial broadcasting has been regulated since 1961 by the Stockholm Agreement in Europe (ST61) and since 1989 by the Geneva Agreement for Africa (GE89). The necessity of new regulations has become increasingly important due to a fast development of digital broadcasting technologies in recent years.

From 15 May 2006 to 15 June 2006 a session of the Regional Radiocommunication Conference (RRC06), organized by the International Telecommunication Union (ITU), was held in Geneva. Delegations from 104 countries in Europe, Africa, Middle-East and ex-USSR gathered to negotiate a new frequency plan for the digital terrestrial

¹The results described in this Chapter formed the basis of the following paper: J. Mościcki, A. Manara, M. Lamanna, P. Mendez, and A. Muraru. Dependable distributed computing for the International Telecommunication Union Regional Radio Conference RRC06. *CERN Technical Report, arxiv:0906.2143*, 2009.

²International Telecommunication Union, <http://www.itu.int>

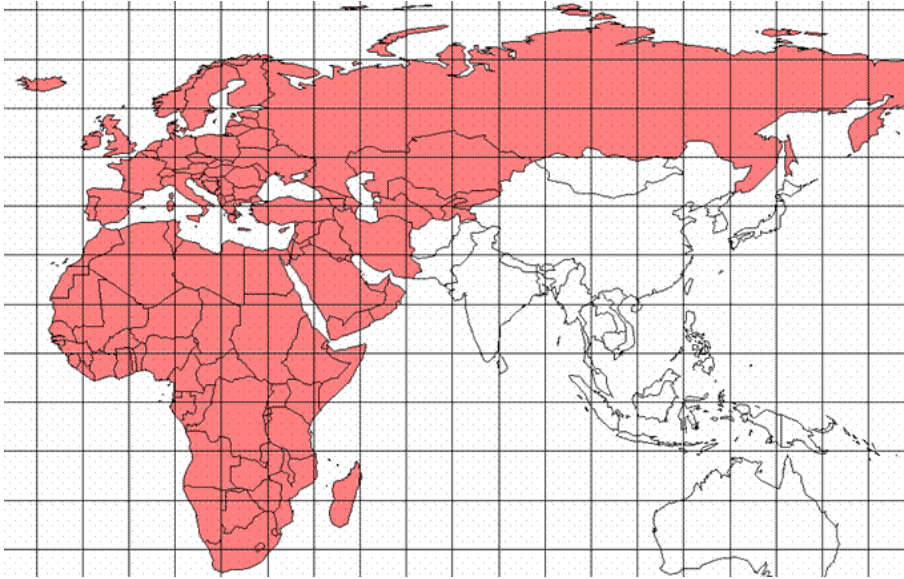


Figure 6.1: Geographical extent of the RRC06 i.e. area covered by compatibility analysis.

broadcasting services in UHF (470-862 MHz) and VHF (174-230 MHz) bands. The resulting frequency plan became a part of a new international agreement, the RRC06 Final Acts [2], which enables an efficient move to the new era of digital broadcasting at international level. Fig. 6.1 shows the area covered by the RRC06.

Preliminary analysis indicated that one component of the planning process, the compatibility analysis, was highly CPU intensive. The goal of the compatibility analysis is to evaluate the interference between broadcasting stations to identify those that can share the same channel. The analysis includes several parameters of the broadcasting stations such as the geographic location, the signal strength and other technical characteristics.

Total computing capacity required for the compatibility analysis was estimated at few hundred CPU-days on a high-end 2006 PC. The RRC06 required the output of the compatibility analysis to be delivered at the specified deadline and within few hours of computing time (this corresponds to 100-500 speedup factor). The problem may be therefore described as on-demand, capability computing in the grid.

6.2 Broadcasting planning process

The RRC06 planning process consisted of several iterations which interleaved the compatibility analysis step and assessment of the analysis results, shown as a loop in Fig. 6.2. The output of the assessment step was used as an input for the subsequent analysis iteration. One iteration of the loop extended to one full week: the assessment of the analysis

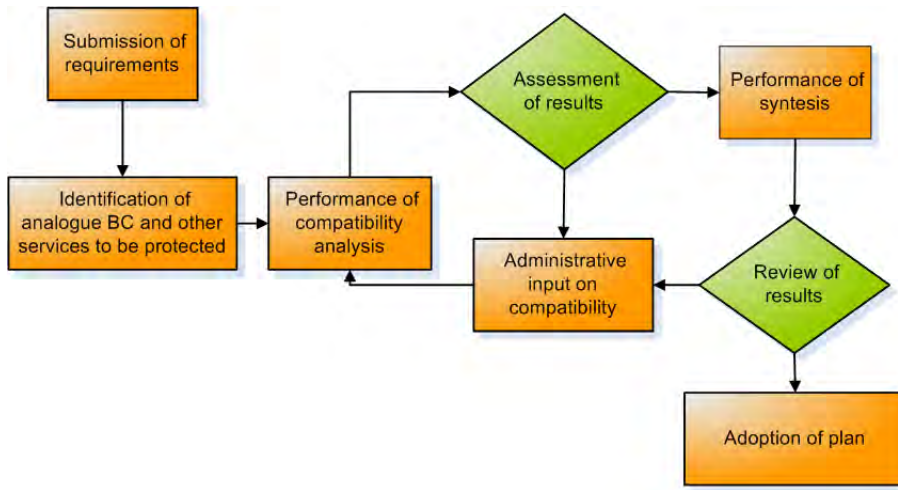


Figure 6.2: ITU negotiation workflow. The compatibility analysis was performed in several iterations as a part of the analysis loop. The actual iteration could implement one of the two loop variants (with or without synthesis step).

results from a previous iteration was performed during weekdays and was followed by the compatibility analysis step (with optional synthesis) performed at weekends.

The assessment step was based on bilateral and multilateral negotiations and co-ordination discussions between 1200 representatives of the ITU member states at the RRC06 conference. A new, refined version of the frequency plan was produced at the end of each week and was used as an input to the next compatibility analysis iteration.

The workload of one compatibility analysis run at the RRC06 corresponded to several hundred CPU hours to be completed within few hours. The time constraint was critical: a problem with timely delivery of analysis results would have resulted in a failure of the international negotiations.

6.3 Compatibility analysis

The compatibility analysis is a calculation of the interference between digital broadcasting stations and services, using established statistical models of signal propagation such as the ITU-R Recommendation P1546-1 [97]. Radio communication services are described by administrative and technical parameters, so called “broadcasting requirements”³. For example, administrative parameters include the name of the ITU member state (“notifying administration”), site name, geographic location, site altitude. Technical parameters include the power levels, assigned frequency, network topology, etc. The input data for the compatibility analysis is a set of broadcasting requirements and for the RRC06 consisted of about 95×10^4 digital requirements, about 95×10^4 analog TV

³Digital requirements are specified using T-DAB (radio) or DVB-T (television) standards.

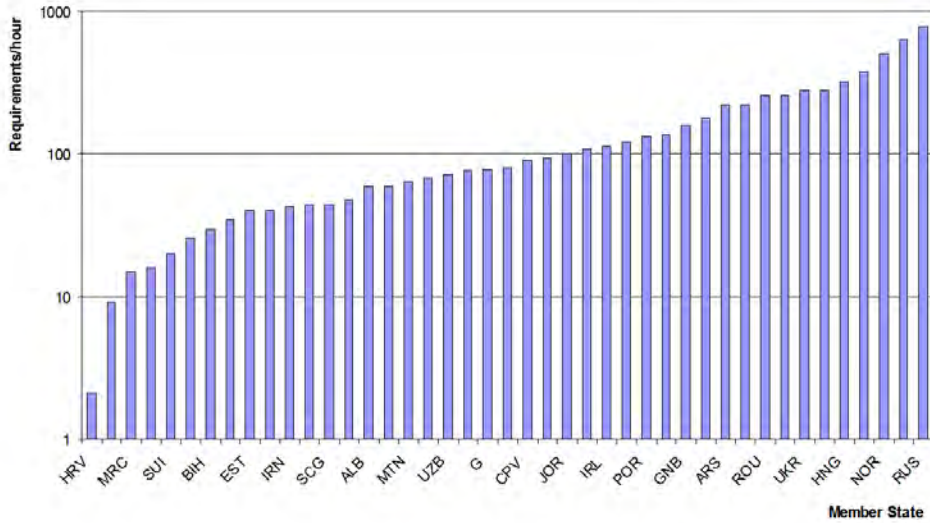


Figure 6.3: Distribution of the number of processed requirements per hour for d2dUHF analysis type for different ITU member states.

requirements and 10^4 requirements for other services. In addition, a few millions of so called “administrative declarations” were included to indicate which requirement conflicts might be safely ignored. The analysis was performed for two frequency bands, VHF and UHF, which resulted in six types of analysis tasks: digital versus digital (d2dUHF and d2dVHF), digital versus other services (d2oUHF and d2oVHF) and other services versus digital (o2dUHF and o2dVHF).

In the compatibility analysis each requirement must be run against all the others for six different types of analysis tasks. The term *atomic calculations* is used to refer to individual, indivisible calculations defined in compatibility analysis datasets. The term *task* refers to a unit of work which corresponds to a set of atomic calculations. The term *job* refers to a grid worker agent.

The distribution of computing time for atomic calculations strongly depends on parameters of corresponding broadcasting requirements and exhibits large variations. The processing time of broadcasting requirements of different ITU member states may span up to three orders of magnitude, as shown in Fig. 6.3. Such large differences result from a different number of acceptable broadcasting channels specified by the requirements, topology of broadcasting networks and signal propagation properties specific to geographical areas of involved countries.

Further investigation showed that a complete static optimization of the workload, i.e. clustering of atomic calculations such that the execution time of each cluster (task) is equal, was not possible. This is because a change of requirements in between the analysis iterations resulted in large and unpredictable changes in the computational

iteration	d2dUHF	d2dVHF	d2oUHF	d2oVHF	o2dUHF	o2dVHF
1	3	5	100	100	100	100
2	4	4	50	50	100	100
3	2	2	50	50	50	50
4	2	2	50	50	50	50

Table 6.1: Number of atomic calculations per task (task granularity) for each analysis round and the six analysis types.

complexity of the atomic calculations.

However, there was clearly a need to create small clusters of atomic calculation for the most CPU demanding type of analysis d2dUHF and d2dVHF minimizing the spread between the shortest and longest tasks. Table 6.1 shows the granularity for the different types of analysis in the RRC06 iterations for the EGEE Grid. The granularity was adjusted manually in between the iterations.

6.4 Implementation of grid-based analysis system for the RRC06

The computing support system to perform the compatibility analysis for the RRC06, shown in Fig. 6.4, was composed of two, redundant systems for improved dependability: a dedicated farm of 84 high-end desktop PCs deployed at ITU headquarters and the EGEE Grid infrastructure. Analysis was run by few users simultaneously with the two subsystems. Common monitoring and accounting system was provided by a MonALISA service⁴. The ITU system is described in more detail in [141].

Grid-based system was implemented with DIANE/GANGA User-level Overlay. Compatibility analysis application modules were developed and plugged in the DIANE framework. The job submission was done by GANGA using standard gLite backend plugin. Distribution of input data and software installation was performed by a VO manager in a separate step.

Task were defined by a set of application-specific parameters: a pair of requirement identifiers and an identifier of an analysis type. Task execution consisted of running a standalone executable on a pre-installed input dataset and with appropriate parameters. The task dispatching performed by the RunMaster consisted of selecting a task from the head of the task queue and allocating it to a next available worker by transferring appropriate parameters (c.f. Fig. 4.3). The communication overhead in this case is typically much smaller than in the systems based on checkpointing and task migration and it allows scheduling with a high rate of incoming and outgoing tasks. The DIANE RunMaster routinely achieved peaks of 110-120 Hz without observable degradation of the performance. This means that scheduling overhead is negligible for up to $N \times 120$ worker agents if average task duration is N seconds.

⁴MonALISA: Monitoring Agents in A Large Integrated Services Architecture, <http://monalisa.cern.ch>

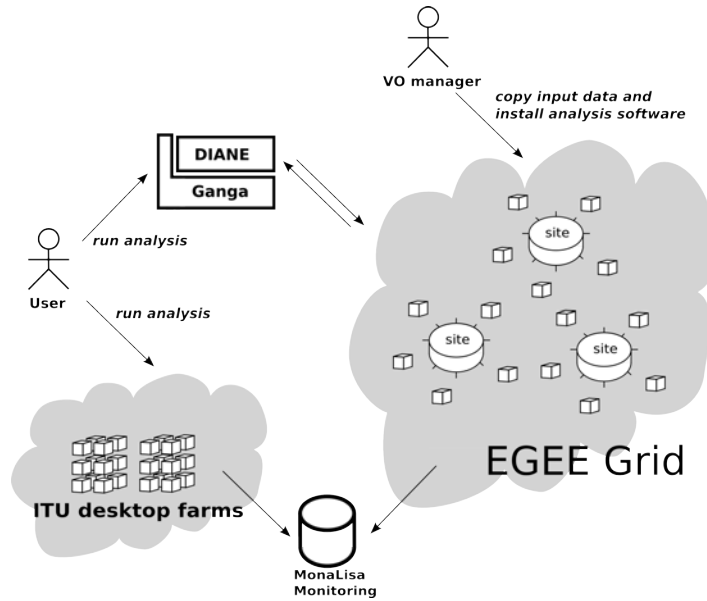


Figure 6.4: Data analysis system for ITU RRC06 consisting of two redundant subsystems: dedicated ITU farms and the EGEE Grid.

To improve dependability, some key services, such as the Master servers and critical gLite WMS services were deployed in several instances to increase the redundancy and to allow failover in case of problems. For storing the application output a globally distributed filesystem (AFS) and local filesystem were used simultaneously.

Reliable and fast delivery of the compatibility analysis results was a key to success of the RRC06. Each compatibility analysis run was preceded by an update of analysis software and input data with 2 hour notice from the RRC06 operational team. In this time-window the grid-based system had to be up and ready to start the analysis at full speed. The distribution of updated input data was the first step of compatibility analysis run and it was performed by a VO-manager using a set of grid jobs which downloaded a 100 MB installation package from a central repository, ensuring consistency with MD5 checksums.

The CPU demand for compatibility analysis was estimated at 500-1000 CPUh per iteration what is smaller than for other grid applications. On the other hand the availability of resources within well-defined and strict time constraints was critical. Therefore, high-availability centers⁵ in the EGEE Grid were involved. The resources at these centers were not dedicated to the RRC06 activity, however, the job priority parameters were adjusted during short periods of intensive processing of the RRC06 compatibility analysis. On average 300 CPUs were observed to be available at all times with occasional peaks of 600 CPUs.

⁵CERN, CNAF+few other sites(I), PIC(E), DESY(D), MSU(RU), CYFRONET(PL)

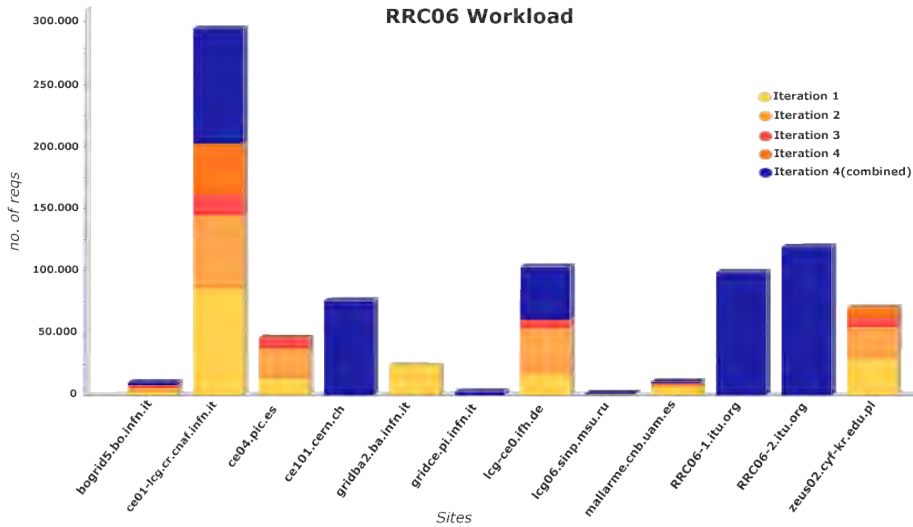


Figure 6.5: Workload executed shown as the number of processed broadcasting requirements in the selected EGEE Grid sites and the two ITU desktop farms: RRC06-1.itu.org and RRC06-2.itu.org. Data provided by MonALISA monitoring service.

Fig. 6.5 shows the total workload executed by ITU clusters and the EGEE sites. Fine-grained monitoring of the grid worker nodes and ITU farm nodes to produce in real-time, high-level reports and charts delivered by the MonALISA framework which provides a set of pluggable distributed services for monitoring, control, management and global optimization for large-scale distributed systems.

6.5 Analysis of task processing

The summary of the RRC06 iterations is presented in Table 6.2. For each analysis iteration the total workload consisted of N_{calc} atomic calculations. The calculations were executed in bunches according to previously defined static clustering (section 6.3). The reliability of DIANE/GANGA system exceeds by several orders of magnitude the reliability of standard job submission: in run 1 less than 10 tasks were lost, in run 2 only one task was lost while in runs 3 and 4 all tasks were successfully completed.

The total CPU demand decreased with each RRC06 iteration. The member states decreased the number of requirements and the number of acceptable channels for each requirement, therefore reducing the total workload at each analysis iteration. As the frequency plan was refined during successful negotiations between the member states, the number of conflicting requirements also decreased.

During pre-conference preparatory planning activities only 34% of requirements were satisfied. At the first iteration of the RRC06 the percentage increased to 64% (UHF) and 74% (VHF), to reach a satisfactory 93% (UHF) and 98% (VHF) for the final plan.

iteration	N_{calc} ($\times 10^3$)	N_{task} ($\times 10^3$)	t_{total} (hours)	t_{worker} (hours)	N_{worker}	r_{fail}
1	243	26	6.66	425	190	$< 3 \times 10^{-4}$
2	237	23	6.50	332	125	4×10^{-5}
3	224	40	1.58	192	210	0
4	218	39	1.01	151	320	0

Table 6.2: Summary of the RRC06 compatibility analysis iterations. The N_{task} tasks were distributed dynamically to the N_{worker} Worker agents. The Worker agents were submitted as *jobs* and executed on the grid worker nodes. t_{total} is the makespan or the total time to complete the compatibility analysis. t_{worker} is the integrated elapsed time on the worker nodes. r_{fail} is the reliability of the system and corresponds to the number of failed tasks which could not automatically recover.

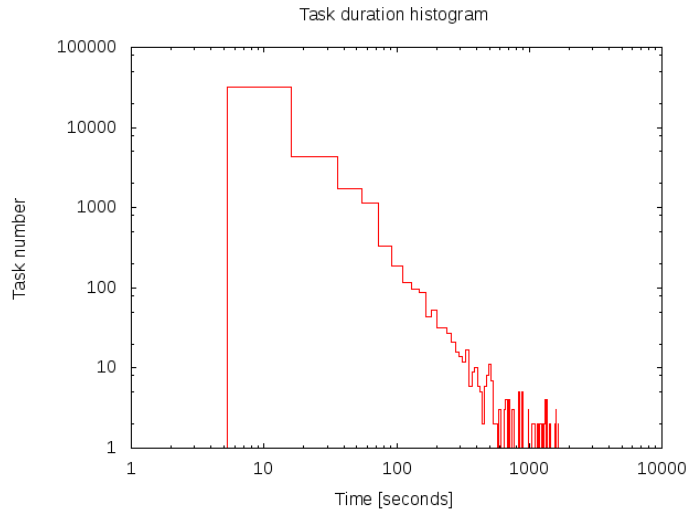


Figure 6.6: Distribution of the wallclock task execution time. Most of 40000 tasks execute in less then 10 seconds; individual tasks execute in 1000 seconds.

Due to grid dynamics, a different set of worker nodes was used at each iteration, resulting in different CPU and memory characteristics of the worker nodes actually used. Therefore, it is not possible to easily normalize and compare the execution times t_{total} and t_{worker} . The distribution of wallclock execution time of the tasks is almost exponential and is shown in Fig. 6.6.

6.5.1 The 3-phase anatomy of analysis runs

The task processing efficiency depends on the grid job submission latency, efficiency of task scheduling and workload balancing at the end of the run. Fig. 6.7 shows the workload and task processing history for selected runs. Three processing phases may be distinguished. N_w worker agents are submitted at $t_0 = 0$. In the submission phase, $t < t_1$, the throughput of the system is limited by the submission latency. As the pool of worker nodes increases the target of N_w workers is reached at time t_1 . In the main processing phase, $t_1 < t < t_2$, the pool of worker nodes remains stable and the system throughput mainly depends on the efficiency of scheduling. At time t_2 the number of remaining tasks becomes smaller than the number of processors in the pool and the system enters the final phase, in which the execution time is dominated by the workload-balancing effects from the few slowest tasks.

The number of available worker nodes may vary significantly in the grid from one run to another. The contribution of the job submission latency to the total execution time may be approximated by the area between the target line (requested size of the worker pool achieved when the system enters the main phase) and the curve representing the actual size of the worker pool (see Fig. 6.7). In run 3 the latency of job submission corresponded to 12% of the total execution time, whereas in run 4 it corresponded to 48%: 33% in the submission phase and 15% in the main processing phase.

The integrated difference between the worker pool size and the number of busy workers corresponds to the scheduling overhead. This overhead includes the network latency and throughput as well as the task handling efficiency of the master server. In run 3 the scheduling overhead in the submission and processing phases corresponded to 2-3%. In run 4 the 30% scheduling overhead in the submission phase was observed and 10% in the processing phase.

The unbalanced execution of the slowest tasks in the last phase contributed to 26% of the total execution time in run 3 and to 5% in run 4. In this phase the utilization of available resources was very low, 5% in run 3 and 20% in run 4. The majority of the workers in the pool remained idle while the few remaining tasks were being finished.

In other words in run 3 task scheduling overhead was very small but workload balancing was poor. Conversely, in run 4 task scheduling overhead was large but workload balancing was good.

6.5.2 Impact of task ordering on load balancing and efficiency

Why in runs 3 and 4 such radical differences in scheduling overheads and quality of workload balancing were observed? This may be explained by radically different ordering of tasks in both runs, which may be analysed using a workload distribution graph

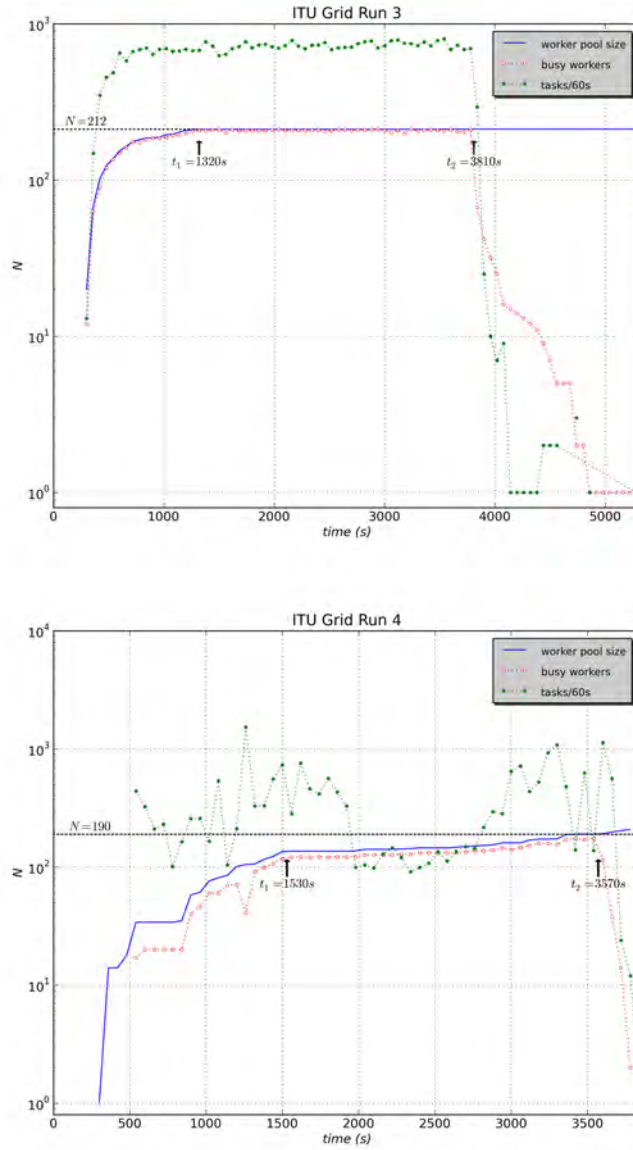


Figure 6.7: Workload and task processing history plot for run 3 and 4 showing the size of the worker pool, number of busy workers and number of processed tasks per minute. The solid target line ($N = 212$ for run 3 and $N = 190$ for run 4) indicates the requested worker pool size. Data is sampled in 60 s bins. In run 4 two parallel master servers were used and this figure corresponds to one of the masters hence to a half of the total processed workload in run 4.

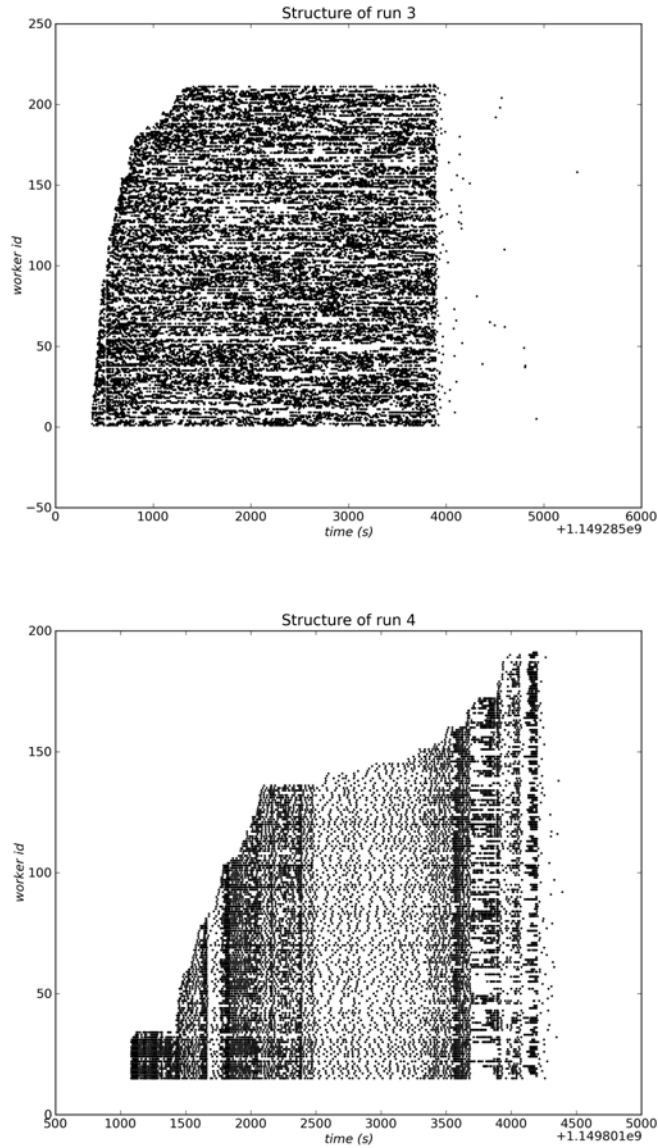


Figure 6.8: Workload distribution graphs showing the completion time of tasks for run 3 (random task order) and run 4 (natural task order as defined in the ITU dataset). One point corresponds to one completed task. Higher density of points indicates higher load on the task scheduler.

for a late-binding system introduced in Fig. 3.2 in Chapter 3. The actual workload distribution graph for runs 3 and 4 are shown in Fig. 6.8, where a task completed by a worker w at time t is represented as a point (t, w) . In run 4 the tasks were dispatched in the natural order of atomic calculations as defined in the input data set. It happened that the atomic calculations (requirements) were defined country-by-country in the ITU frequency plan. Therefore, clusters of very short tasks could occasionally generate very high load on the server. By a pure chance, the longest tasks were processed in the middle of the run and did not affect the overall load-balancing. On the other hand, in run 3 the tasks were dispatched in a random order by the scheduler. The momentary load on the server was reduced as long and short tasks were more uniformly distributed in time across the entire run. However, there were a few long tasks at the end of the run that resulted in poor load-balancing. This effect is systematic and occurs with a probability proportional to the number of long tasks in the dataset (the phenomenon occurs if long tasks are drawn at the end of a run). Both task ordering methods inevitably result in inefficiencies.

6.6 Summary

The system based on the User-level Overlay in the EGEE Grid contributed to the success of the RRC06 Conference. The GE06 frequency plan is now a part of a new international treaty.

The intrinsic job submission latency in grids makes it hard to run a large number of short jobs in a short time. For the RRC06 using the User-level Overlay allowed to reduce grid overheads and provided efficient management of a large number of tasks. Additionally a runtime workload balancing allowed to evenly distribute the workload without precise, a priori knowledge of the task execution times in the dataset. The overhead reduction and dynamic workload balancing were the crucial factors of the successful usage of the EGEE Grid for the RRC06, however, due to statistical effects of workload distribution the efficiency of task processing may not be always optimal.

A seamless access and integration at the application level of grid resources and corporate infrastructures may be beneficial for other user communities. A typical use-case could include dedicated in-situ resources for fast response and grid resources when facing peak demand. In such a scenario, grids could provide a competitive alternative to traditional procurement of resources. The EGEE Grid delivered dependable peak capacity to an organization which normally does not require a large permanent computing infrastructure. Additionally grid technology was successfully used in a new area to provide a dependable just-in-time service with limited support and training required by the ITU personnel.

CHAPTER 7

Capacity computing case study: LatticeQCD simulation

I am now convinced that theoretical physics is actually philosophy.

Max Born

We saw that the User-level Overlay with late binding may provide important efficiency improvements for small and medium-sized workloads. The main target for grids, however, are the capacity-computing applications and very large workloads. Hence, the following question arise: what are the advantages of using the User-level Overlay with late binding for very large workloads processed over long time? In this Chapter¹ we attempt answering this question by presenting a case study of a high-throughput processing system for solving a problem in theoretical physics. We also demonstrate additional benefits of the DIANE/GANGA User-level Overlay which include a transparent access to different types of computing resources.

7.1 Introduction

Quantum Chromodynamics (QCD) describes strong interactions between quarks and gluons which are normally confined inside protons, neutrons and other baryons. Because the interactions are strong, the analytic perturbative expansion, where one determines exactly the first few orders of a Taylor expansion in the coupling constant, converges poorly. Thus, one commonly resorts to large-scale Monte Carlo computer simulations,

¹The results described in this Chapter formed the basis of the following paper: J. T. Mościcki, M. Wos, M. Lamanna, P. de Forcrand, and O. Philipsen. Lattice QCD thermodynamics on the Grid. *Computer Physics Communications*, 181(10):1715 – 1726, 2010.

where the complete properties of QCD can be obtained numerically, up to controllable statistical and systematic errors.

In order to simulate QCD on a computer, one discretizes space and time into a 4-dimensional grid. The quark and gluon fields live respectively on the sites and bonds of this *lattice*². The simulation generates a sample of the most important configurations of quark and gluon fields, evolving them one Monte Carlo step at a time. Statistical errors come from the Monte Carlo sampling, and systematic errors come from the finite lattice spacing a and finite size of the simulated 4-dimensional "box" $N_x a \times N_y a \times N_z a \times N_\tau a$. While the three space dimensions should be in principle infinitely large, the fourth dimension defines the temperature T of the lattice: $T = 1/(N_\tau a)$.

The majority of Lattice QCD simulations are used to study properties of the $T = 0$ theory. Thus, all four dimensions are large, and state-of-the-art projects with, say, $N_x = N_y = N_z \sim \mathcal{O}(32)$, $N_\tau \sim \mathcal{O}(64)$, require distributing the quark and gluon fields over many CPUs which must be efficiently interconnected to maintain a reasonable efficiency. The accumulated statistics typically reach $\mathcal{O}(10^{3 \div 4})$ Monte Carlo "trajectories".

In contrast, we are interested in high-precision measurements of some properties of QCD at finite temperature. This means that the lattice we study, of size $16^3 \times 4$, fits into the memory of a single CPU, and that our large CPU requirements stem from the high statistics required, $\mathcal{O}(10^6)$ trajectories. In this case, a large pool of independent CPUs represents a cheap, efficient alternative to a high-performance cluster. This is why, in our case, using the EGEE Grid was the logical choice.

7.2 Problem to be solved

The physics problem we address is the following. At high temperature or density, the confinement of quarks and gluons inside baryons disappears: baryons "melt" and quarks and gluons, now deconfined, form a plasma. When the net baryon density is zero, this change is a rapid but analytic ("smooth") crossover as the temperature is raised. On the contrary, at high baryon density it is believed to proceed through a true non-analytic, first order phase transition. This change of the nature of the transition as a function of baryon density or chemical potential is analogous to the one occurring in liquid-gas transitions as a function of pressure: at low pressure water boils, and in this first-order transition, it absorbs latent heat. With increasing pressure, the transition (boiling) temperature rises and the first order transition weakens, i.e. the latent heat decreases until it vanishes altogether at a critical point, where the transition is second order. Beyond this critical pressure, the transition to the gaseous phase proceeds continuously as a crossover (with no latent heat). Correspondingly in QCD, there may exist a particular intermediate baryon density where the latent heat of the QCD phase transition vanishes and the phase transition is second-order. The corresponding temperature and baryon density or chemical potential define the so-called QCD critical point, which is the object of both experimental and theoretical searches, the former by heavy-ion collision

²The term *lattice* is used in this Chapter to refer to the simulated physical system to distinguish it from the computing *system* used to perform the simulation.

experiments at RHIC (Brookhaven) and soon at LHC (CERN), the latter by numerical lattice simulations.

In theoretical studies, one may also consider the u, d, s quark masses as variable and investigate their influence on the order of the transition as a function of $m_u = m_d$ and m_s for zero baryon density. For small enough quark masses the phase transition is of first order and corresponds to a high-temperature restoration of the chiral symmetry, which is spontaneously broken at low temperature. This chiral phase transition weakens with increasing quark masses until it vanishes along a chiral critical line, which is known to belong to the $Z(2)$ universality class of the 3d Ising model [103, 50]. For still larger quark masses, the transition is an analytic crossover. At finite density, it is generally expected that the $Z(2)$ chiral critical line shifts continuously with μ until it passes through the physical point at μ_E , corresponding to the critical point of the QCD phase diagram.

What makes the study particularly interesting is that earlier results obtained for three degenerate quark flavours indicate that QCD critical point disappears [51, 52], contrary to standard expectations. This surprising result needs to be confirmed for non-degenerate quark masses, which we do here, and on finer lattices. If it turns out to be a property of the continuum QCD theory, it will have a profound impact on the QCD phase diagram. In particular, it will make it unlikely to find a QCD critical point at small baryon density.

7.3 Simulation model

The quark and gluon fields in the Lattice QCD simulation are mapped onto a discrete space-time $16^3 \times 4$ lattice. The simulation evolves the configurations of quark and gluon fields in a succession of Monte Carlo trajectories. The lattice is studied at 18 different temperatures around that of the phase transition. The temperatures correspond to the values of the parameter β , the lattice gauge coupling constant. What is measured is the response to a small increase in the baryon density. The signal is tiny and easily drowned by the statistical fluctuations.

A complete lattice configuration is kept in a *snapshot* file and the initial configurations for each β -value are called *mother snapshots*. Each snapshot may be evolved in Monte Carlo time by a series of *iterations*. The signal to noise ratio is very small and requires a large number of iterations to become significant. However, if random number sequences are different then multiple parallel Monte Carlo *tracks* may be used for the same β -value. The mother snapshots may be replicated and tracks use different *random seeds*. The tracks execute independently of one another and represent sequences of independently evolving simulation steps. The snapshot's *maturity* is the number of iterations performed on that snapshot (see Fig. 7.1).

At first the replicas of the mother snapshots are identical. The subsequent iterations lead to randomization of the replicas. After a large number of iterations snapshots mature and diverge enough to contribute statistically independent simulation results. Before the randomization point is reached the snapshots are *immature* and only provide statistically correlated contributions. The number of iterations needed to randomize

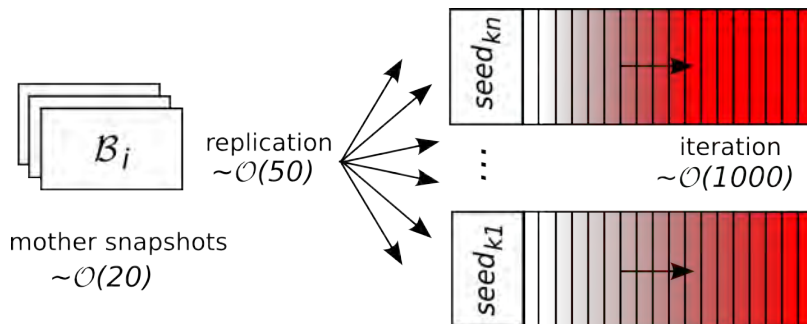


Figure 7.1: The high-level structure of LQCD simulation. Mother snapshots are replicated into independently evolving simulation tracks which consist of a large number of tasks. Tasks perform several simulation steps at a time, increasing the maturity of the initial snapshot.

the lattice was not a priori known and it was estimated to be between 300 and 500 iterations (corresponding to 20-30 CPU days on a standard 2008 PC) per snapshot. This corresponds to the amount of processor time “wasted” on randomization of the lattice. The sequential overhead in this case is very large, both in absolute terms and as a fraction of entire computation. As the number of available processors varies on much shorter time scales compared to the duration of the computation at some point the number of available processors may become smaller than the number of snapshots. Then a scheduling problem arises: how to choose a subset of snapshots in order to achieve a required number of “useful” iterations before the specified deadline? Processing more snapshots than the number of available processors would result in serialization of computations. Given the long randomization time and the large number of snapshots, a naive scheduling would lead to spending all CPU time in randomizing the replicas rather than doing “useful” work. Due to the dynamic nature of grids, large fluctuations in the number of simultaneously running jobs were expected. Therefore, the system was facing the following challenges:

- adapt the scheduling algorithm so that the number of useful iterations may be maximized,
- manage the utilization of resources available in the Grid on par with the number of parallel simulation tracks,
- run autonomously over long periods of time.

7.4 Implementation and operation of the simulation system

7.4.1 Processing with HPC resources

The pre-thermalization of the Lattice QCD system was performed on a NEC-SX8 vector machine (Table 7.1) at HLRS in Stuttgart³. About 10 CPU minutes were required per Monte Carlo trajectory, and about 500 trajectories per β -value were produced. The fundamental reason for using vector machines in the pre-thermalization phase is the considerably higher throughput than the average node on the Grid. As finer lattice spacings are involved and the lattices get larger, exploiting fine-grained parallelism may also be beneficial. In this case a parallel architecture with low-latency interconnect is required.

Peak Performance	1.2 TFlops
Processors	80 CPUs
Number of Nodes	10
Memory/node	128 GB
Disk	160 TB shared disk
Node-node interconnect	IXS 8 GB/s per node

Table 7.1: NEC-SX8 supercomputer characteristics.

The mother snapshots obtained on a NEC-SX8 vector machine were then used for the subsequent processing on the EGEE Grid which took place from April to October 2008 (and then was followed by additional runs).

7.4.2 Processing with the EGEE Grid

The QCD simulation system for the EGEE Grid was implemented with the User-level Overlay as shown in Fig. 7.2.

The master is responsible for the task scheduling and controls the order in which the snapshots are scheduled for processing to individual workers. The snapshot files are stored on the local file-system of the master and are exchanged with the worker nodes using the DIANE file transfer service. Small application plugins written in the PYTHON programming language are used to customize the DIANE framework for the needs of the Lattice QCD processing.

Each worker performs a given number of iterations and uploads the resulting snapshot file back to the master. The snapshot is then ready to be evolved further by a free worker agent. In order to avoid unnecessary network traffic, once a particular worker agent downloads a snapshot, it keeps processing it as long as possible. Therefore, the snapshot does not have to be downloaded multiple times and the worker continues the simulation using the snapshot already cached at the worker node. The worker agent

³Nec-SX8 technical description, <http://www.hlrs.de/systems/platforms/nec-sx8>

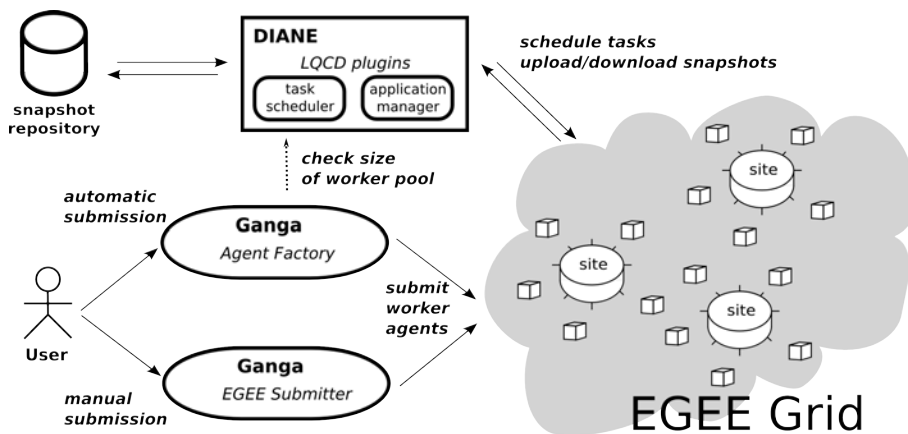


Figure 7.2: Simulation system for LQCD studies. Task processing is handled by DIANE RunMaster with LQCD-specific plugins. Worker agent submission is performed manually with a help of GANGA submitter scripts, or automatically by the GANGA-based Agent Factory. Snapshot files are transferred using a built-in DIANE File Transfer Service to and from the Grid and stored in a local repository.

runs as a grid job and it has limited lifetime. The typical limiting factor is the time limit on the batch systems at the Grid sites.

Worker agents are submitted using GANGA. In the initial phases of the study the submission was done manually by the users with agent submitter scripts developed in Sec. 4.7.1. In later phases of the study the submission was performed by the Heuristic Agent Factory developed in Sec. 4.7.3

The processing on the EGEE Grid was split in several runs. The processing workflow (Fig. 7.3) involved an active participation of the end users: the intermediate simulation results were analyzed on-the-fly by the theoretical physicists. This lead to several modifications and fine tuning of the processing including the simulation code, the number of β -values, the number of snapshots and the scheduling algorithms. The processing was also interrupted due to technical reasons such as service upgrades or hardware downtime. The processing phases are summarized in Table 7.2.

run	N_β	$N_{snapshot}$	duration [weeks]	iterations [$\times 10^3$]	N_{CPU}	T_{CPU} [years]	data transfer [TB]
1	16	400	11	300	4142	52	1.4
2	24	1450	9	700	21432	121	3.4
3	18	1063	3	267	12197	47	1.3
4	18	1063	8	266	12105	59	1.3
total			31	1533	49876	279	7.5

Table 7.2: Summary of LQCD processing runs

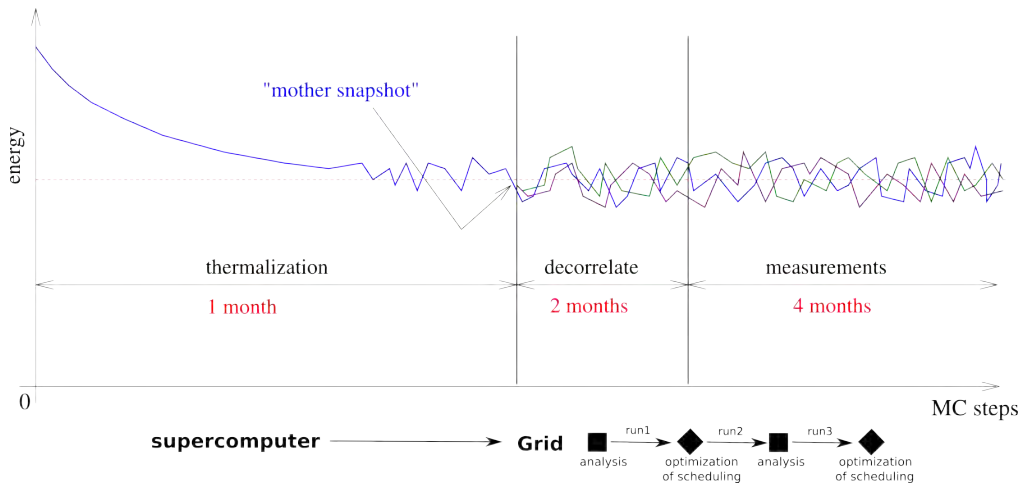


Figure 7.3: Monte Carlo time history of a typical measured QCD observable, for example the energy, illustrating the LQCD processing steps. The observable is relaxing towards its equilibrium value and then fluctuating around it. Simulation with a supercomputer is performed to produce a mother snapshot, which then serves as a starting point for a number of grid runs with different random number initializations, yielding measurements shown by the multiple fluctuating lines.

The goal of runs 1 and 2 in the first, most critical phase of the processing, was to achieve 700,000 iterations, including the snapshot randomization, within approximately 10 weeks, in order to obtain publication-quality results. The average execution time (wall-clock elapsed on the worker node) per iteration was estimated at 1.5-2.5 CPU h on a typical 2008 PC. The size of the snapshot file (input and output of each iteration) was 10 MB. In run 1 the lattice was analyzed for a quark mass of $am = 0.0065$ with 16 β -parameters (i.e. temperatures) uniformly distributed in a value range (5.1805, 5.1880). In run 2 the estimation of the quark mass was refined at $am = 0.0050$, i.e. $m_s/m_{u,d} = 50$. Additional β -parameters were defined in the middle of the range and placed in between the existing values to provide more simulation points in the vicinity of the phase-transition point to the quark-gluon plasma, further referred to as *sensitive region*.

The reduction of the quark mass lead to longer execution time for the Monte Carlo step. This was compensated by reducing the frequency of measurements, to obtain a small overall reduction in CPU time per iteration. Run 1 and 2 were performed in parallel. The simulation parameters of run 2 were better tuned, therefore run 2 has eventually become the reference for the publication of physics results [53]. Processing runs 3 and 4 were performed in a second phase and provided more precise data for subsequent studies. The β -value range was reduced as well as the total number of snapshots.

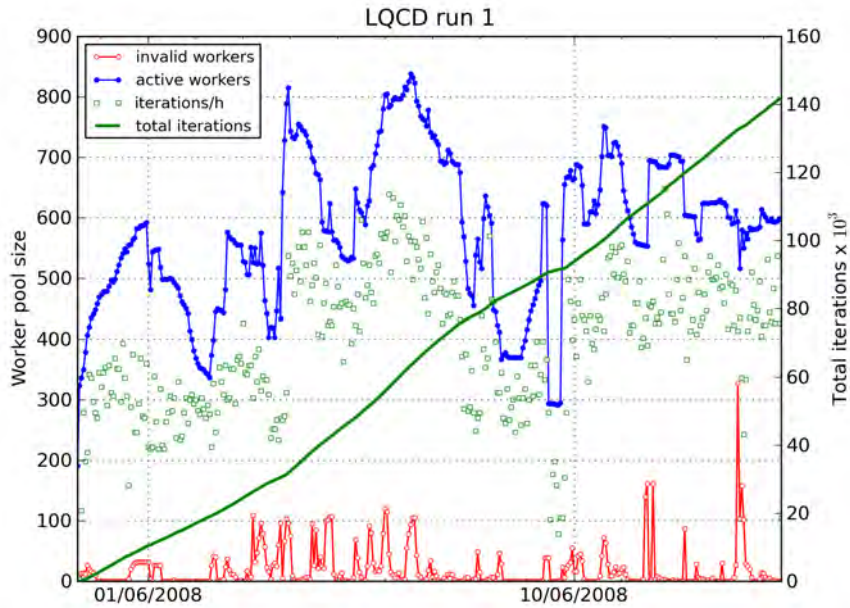


Figure 7.4: History plot showing the evolution of processing and the worker pool size in run 1 (for a selected period). Manual submission of worker agents.

7.4.3 Analysis of system performance

Analysis of the system performance is based on the monitoring data collected by the DIANE master. For each run a journal file is generated which contains a complete record of events that occur between the master and the workers, and which is used to extract system parameters such as the number of active workers, the number of added workers, task duration etc. All the quantities have been sampled in one hour intervals.

The evolution of run 1 is presented in Fig. 7.4, run 2 in Fig. 7.5, run 3 in Fig. 7.6 and run 4 in Fig. 7.7. Due to missing data only selected periods of each run are shown. The left vertical axis shows the size of the worker pool i.e. the number of worker agents and the number of produced iterations in each time interval. The right vertical axis provides the scale for the total number of iterations. Some exceptional events occurring during the runs, such as expiry of grid user credentials or server problems, add to natural fluctuations of the worker pool. The most important events are marked with arrows and described in the captions.

The lifetime of worker agents is limited by the batch queue limits and therefore the pool of productive workers is constantly changing. The workers which run the simulation during the time interval and successfully upload the result later are considered *active*. Workers which run the simulation but which do not upload the result are not considered

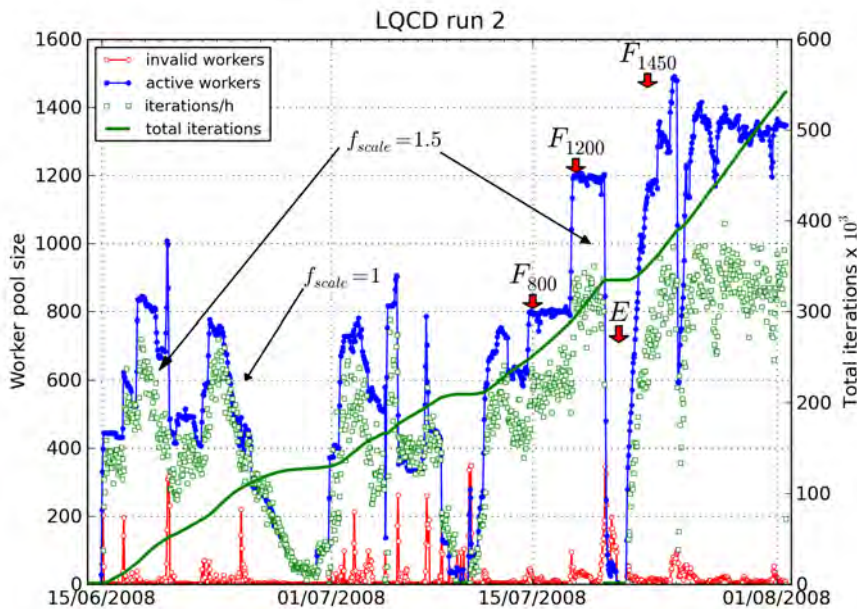


Figure 7.5: History plot of run 2 (selected period). Manual submission until 15/07. Meaning of symbols: F_N – indicates the moment when the factory was enabled to keep N worker agents in the pool; E – workers dropped due to expired user grid credentials; f_{scale} – rescaling factor for the number of iterations per hour.

active in a given time interval. This is the case of workers which were interrupted by the batch system due to the batch system time limits. In practice every worker becomes eventually inactive for a certain time before termination: a worker gets the workload from the master and runs the simulation which is interrupted by the batch system when the time limit is exceeded. This effect is called premature worker cancellation. The workers which never became active, i.e. did not manage to upload any results at all, are considered *invalid*.

The worker failure reasons are multiple. In a sample of 1625 invalid workers failure logs we have found $\mathcal{O}(20)$ different failure reasons which may be broadly attributed to 3 classes: (A) incompatible processor architecture and floating-point instruction set, (B) incompatible or misconfigured system environment in the worker nodes and (C) transient connectivity problems in the DIANE processing layer. While it is possible for the class A and B to specify the resource requirements to exclude a subset of incompatible resources, it is not always efficient from the user point of view as it effectively requires long and tedious error log analysis. Another strategy for dealing with problems in class A would be to compile on-the-fly the simulation program. In our particular case this option was not possible because we used the Intel FORTRAN optimizing compiler

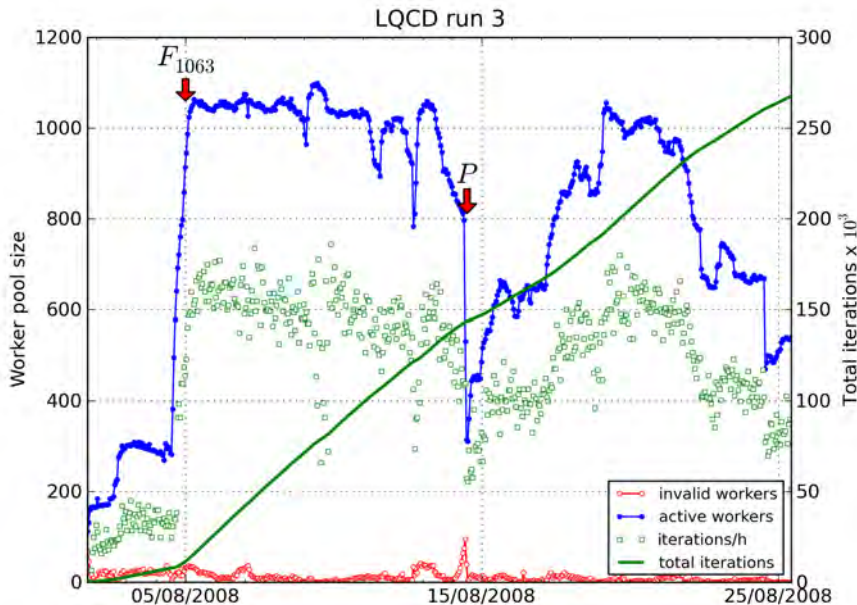


Figure 7.6: History plot of run 3 (selected period). Meaning of symbols: F_{1063} – factory enabled for 1063 workers in the entire period; P – power failure of the master server.

which is not readily available in the Grid sites. The resource selection problem may be efficiently handled in a general way using the Heuristic Agent Factory developed in Sec. 4.7.3. Efficiency of this approach for LQCD simulations is analyzed in Sec. 7.6.

The number of performed iterations in a given time interval is proportional to the number of completed snapshots by the active workers in the pool. Each snapshot is uploaded after 3 completed iterations. The ratio between the number of active workers and the number of produced snapshots per hour is indicated by f_{scale} on Fig. 7.5: $f_{scale} \simeq 1.5$ is a typical value for most of the runs while $f_{scale} \simeq 1.0$ corresponds to a larger number of faster workers being available in the grid in certain periods.

7.5 Task scheduling and prioritization

In this section we describe an application-specific scheduling which was developed for the LQCD simulation system. Its main feature is usage of knowledge of the β -parameter space to improve the total simulation throughput by ranking and prioritization of tasks to maximize the scientific content of the simulation output.

The computational complexity of the simulation (an average execution time) decreases when the temperature of the quark-gluon plasma increases (higher β values).

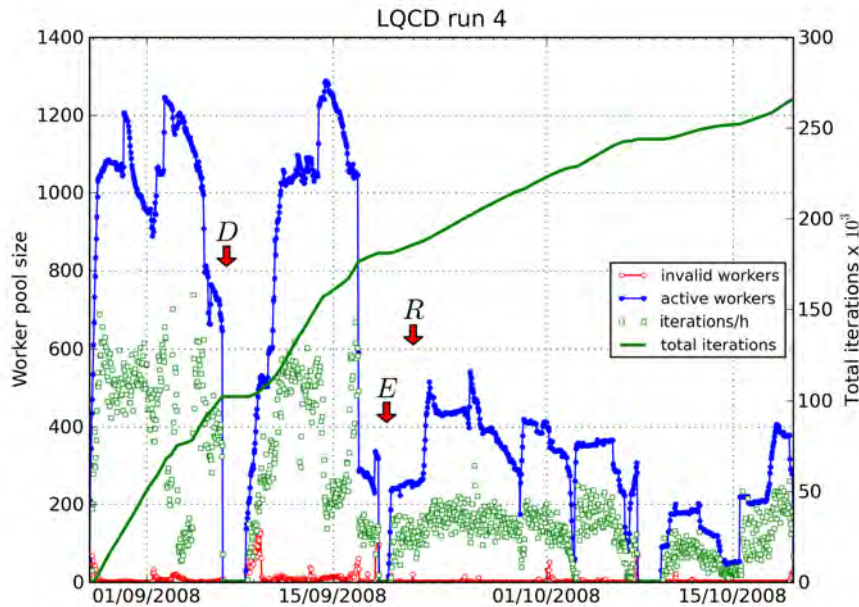


Figure 7.7: History plot of run 4 (selected period). Meaning of symbols: F_{1063} – factory enabled for 1063 workers in the entire period; D – file-server running out of file descriptors, system halted; E – workers dropped due to expired user grid credentials; R – beginning of a period of low resource availability in the Grid, system working in low regime.

This effect is related to the physical behavior of the lattice across the transition temperature. The theoretical curve of the distribution of the execution time as a function of β should be S-shaped and monotonically decreasing, with the inflection point at the phase-transition temperature (however, an exact model function of this dependency remains unknown).

Fig. 7.8 shows the execution time of iterations for each run. The vertical bars show the execution time range between the 25% and the 75% percentile. The points show the value of the median (50% percentile). The absolute values of execution times for each run are different because the internal parameters of the simulated lattice were modified between the runs. The rather large vertical bars account for relatively broad distributions which do not allow to further constrain the transition temperature using the execution time information. The observed distributions of execution times result from the convolution of the intrinsic distribution of the amount of required computations (given by the properties of the simulated QCD lattice) with another distribution reflecting the variability of grid computing resources.

Variability of grid computing resources comes from the non-uniform distribution of

processor speeds across the grid computing nodes. Fig. 7.9 shows the distribution of nominal time to perform 10^9 cycles for all processors used in run 4. This distribution spans almost one order of magnitude⁴ and is further smeared by the fluctuations which result from sharing of the same computing nodes by different jobs of many users. The amount of sharing is defined locally by the grid sites, however, it may be expected to occur frequently as almost all computing nodes in the EGEE Grid are multi-core machines: in run 4 we recorded the majority of nodes to be 8-cores (66%) or 4-cores (25%) and only less than 1% to be single core.

The amount of computation required by the simulation depends on β value and two categories may be distinguished: $\beta \leq 5.1818$, where the QCD lattice is at low temperature, characterized by many small eigenvalues of the Dirac matrix and hence a larger computing time in the associated linear solver, and $\beta \geq 5.1820$ at high temperature, where the plasma is formed and the small Dirac eigenvalues disappear. We also expect a secondary peak of the other kind in each distribution because the separation into two categories is valid only for an infinitely large lattice. This is confirmed by the task execution histogram presented in Fig. 7.10 (bottom) which shows a secondary peak at small CPU-time for low value of β .

We tried to disentangle the intrinsic and grid distributions by first considering the highest temperature, where the amount of computation required fluctuates the least. The distribution of execution times t is shown in Fig. 7.10 (top). The grid variability produces a tail in the distribution, which appears consistent with an exponential form. Given that there exists an intrinsic, minimum execution time t_0 imposed by the hardware, a simple empirical ansatz for the distribution of execution times is then

$$t_{exec} \sim (x - 1)^\nu \exp(-cx) \quad (7.1)$$

where $x = t/t_0$. This ansatz gives a reasonable description of the data at high β (Fig. 7.10, top), with $\nu \approx 3/2$ and $c \approx 3$. It turns out that the optimal values of ν and c hardly varied with β , so that we kept them fixed and considered a single fit parameter $t_0(\beta)$, reflecting the variation of the computing requirements with β due to the physics of the problem. The fit remained rather good at all temperatures β , with $t_0(\beta)$ increasing monotonically as β is decreased, and most steeply at the critical temperature, as expected on physical grounds.

7.5.1 Maturity-based scheduling

In run 1 the snapshots were dynamically prioritized based on their maturity: the snapshots with the least number of Monte Carlo iterations were scheduled before the more mature ones. Let $S_\beta^s(k)$ denote a snapshot after k iterations at a given temperature β and for initial random seed s . Let $S_{\beta_1}^{s_1}(k_1) < S_{\beta_2}^{s_2}(k_2)$ denote that $S_{\beta_1}^{s_1}(k_1)$ should be scheduled before $S_{\beta_2}^{s_2}(k_2)$. The maturity-based scheduling policy may be defined as:

$$S_{\beta_1}^{s_1}(k_1) < S_{\beta_2}^{s_2}(k_2) \iff k_1 < k_2 . \quad (7.2)$$

⁴While largely approximate, the clock speed provides an acceptable estimate of processing power for CPU-bound simulations.

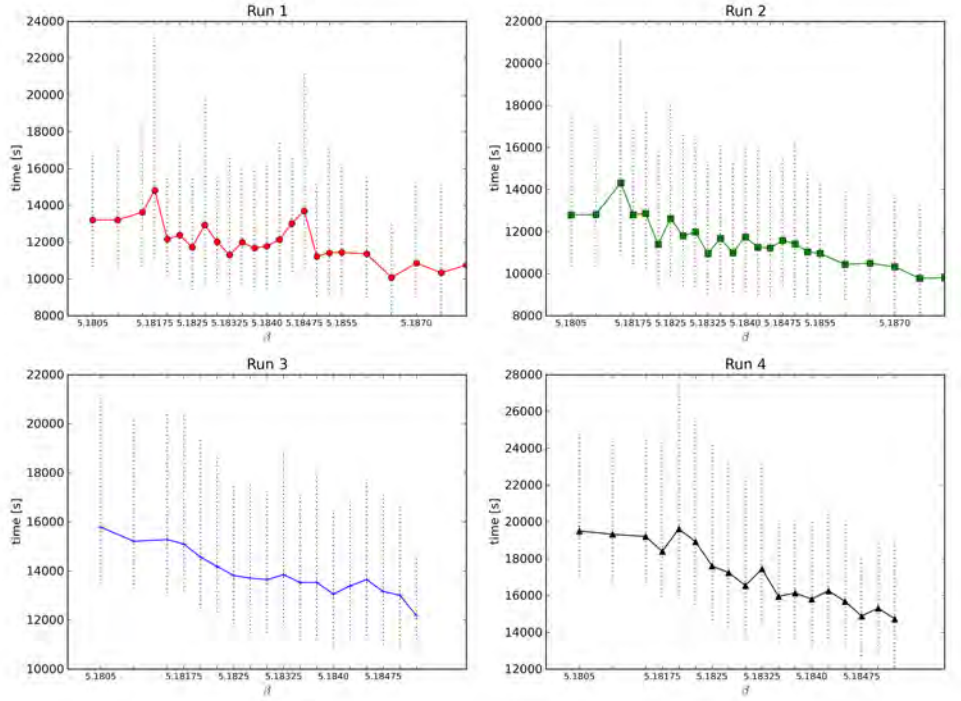


Figure 7.8: Median of elapsed wall-clock time of 3 iterations as a function of β -value. The vertical bars show the execution time range between the 25% and the 75% percentile. The points show the value of the median (50% percentile).

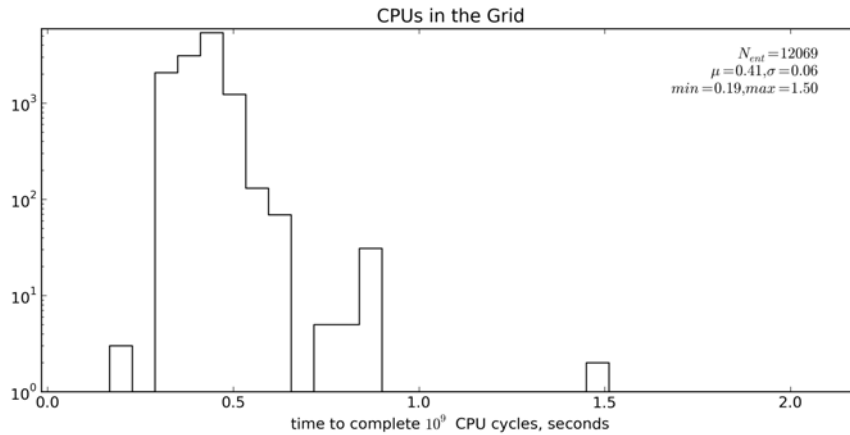


Figure 7.9: Characteristics of CPUs in the EGEE Grid. Distribution of time to complete 10^9 CPU cycles per processor. The highest peak is in 2-3 GHz CPU clock rate range.

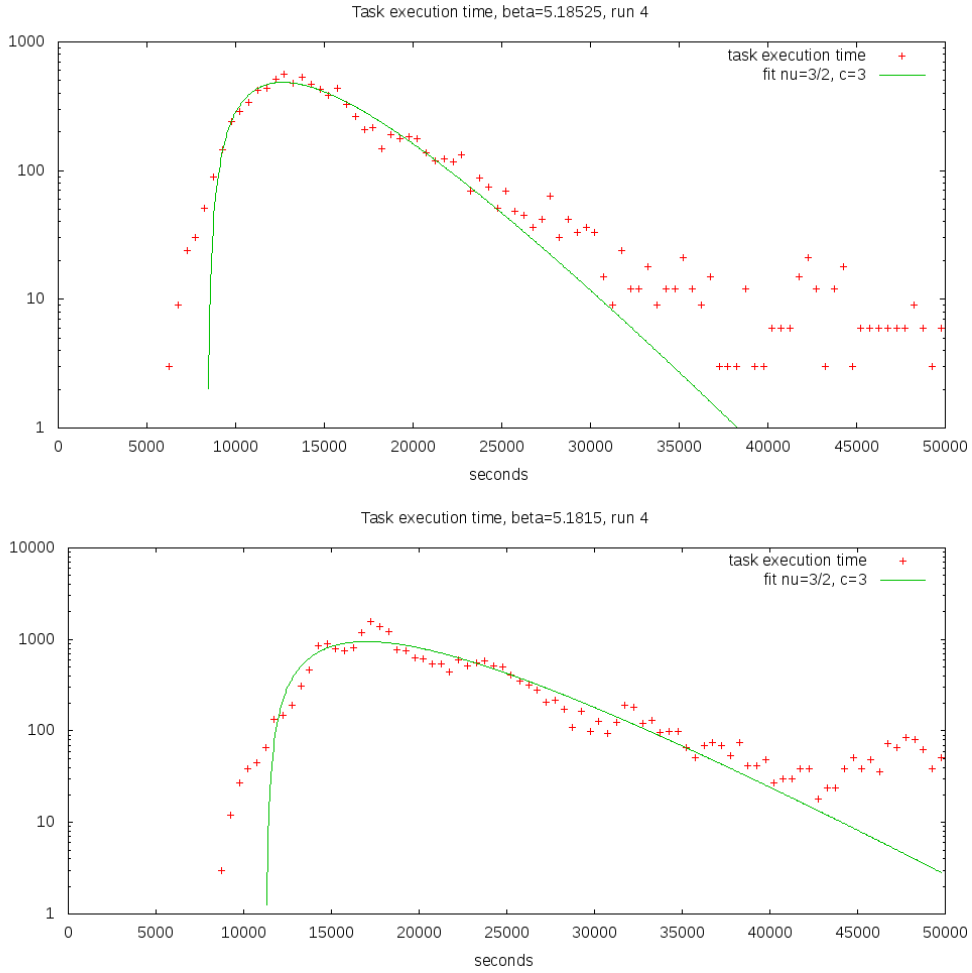


Figure 7.10: Distribution of task execution times (elapsed wall-clock time on the worker node) in run 4 for high β (top) and low β (bottom). For high β the empirical function describes the data very well. For low β , the secondary peak at small CPU time may be explained on physical grounds, as a remnant of the high-temperature phase.

In first approximation, the objective is to evolve all snapshots keeping the spread in the iteration number as small as possible.

7.5.2 Scheduling in the sensitive region

After the initial analysis of simulation results, it was decided to change the range of β -parameters and the scheduling policy for run 2 to improve the convergence speed. A finer-grained sensitive β -region $R = [5.1815, 5.18525]$ around the expected plasma transition temperature was defined. Within the sensitive region R the scheduling policy was to select snapshots with a smaller β -value first

$$S_{\beta_1}^{s_1}(k_1) < S_{\beta_2}^{s_2}(k_2) \iff \beta_1 < \beta_2 . \quad (7.3)$$

Outside of the sensitive region the maturity-based prioritization was kept. Snapshots from the sensitive region were always selected before any snapshots from the outside of the region. Thus the scheduling policy in the entire range was defined as

$$S_{\beta_1}^{s_1}(k_1) < S_{\beta_2}^{s_2}(k_2) \iff \begin{cases} \beta_1 \in R, \beta_2 \notin R & \text{or} \\ \beta_1 < \beta_2 \text{ and } \beta_1, \beta_2 \in R & \text{or} \\ k_1 < k_2 \text{ and } \beta_1, \beta_2 \notin R \end{cases} . \quad (7.4)$$

This gives absolute priority to the sensitive region, and within that region to smaller β values.

In runs 3 and 4 the sensitive region was expanded to include β -values below the expected phase transition point, $R = [5.1805, 5.18525]$. At the same time the β -values above the sensitive region were removed from the simulation.

7.5.3 Analysis of the scheduling results

Fig. 7.11 shows the maturity of the snapshots at the end of each of the runs, grouped by the values of β . The final distribution of maturity depends on the computational requirements for each β -value, the scheduling policy and runtime factors.

Results obtained for run 1 and run 2 show that the maturity-based scheduling is implemented efficiently. However, the final maturity distribution in the sensitive region is influenced by number of available processors in the system.

Scheduling in the sensitive region is based on the ordering of the snapshots (with respect to the β parameter). Therefore, in first approximation the number of completed iterations per snapshot should be larger for smaller β -values. This applies to a system working at low regime, i.e. when the number of available processors is smaller than the number of snapshots. The effect is observed for run 4 in Fig. 7.7: more than half of the time the system is working at 500 workers or less, what corresponds to 50% of the required processing capacity.

At full capacity, the number of available processors is equal to or higher than the number of snapshots. In this case the actual ordering priority does not matter because each snapshot is processed at any time. Considering that the worker pool is constantly changing (workers join and leave quite often) the ordering of snapshots does not influence

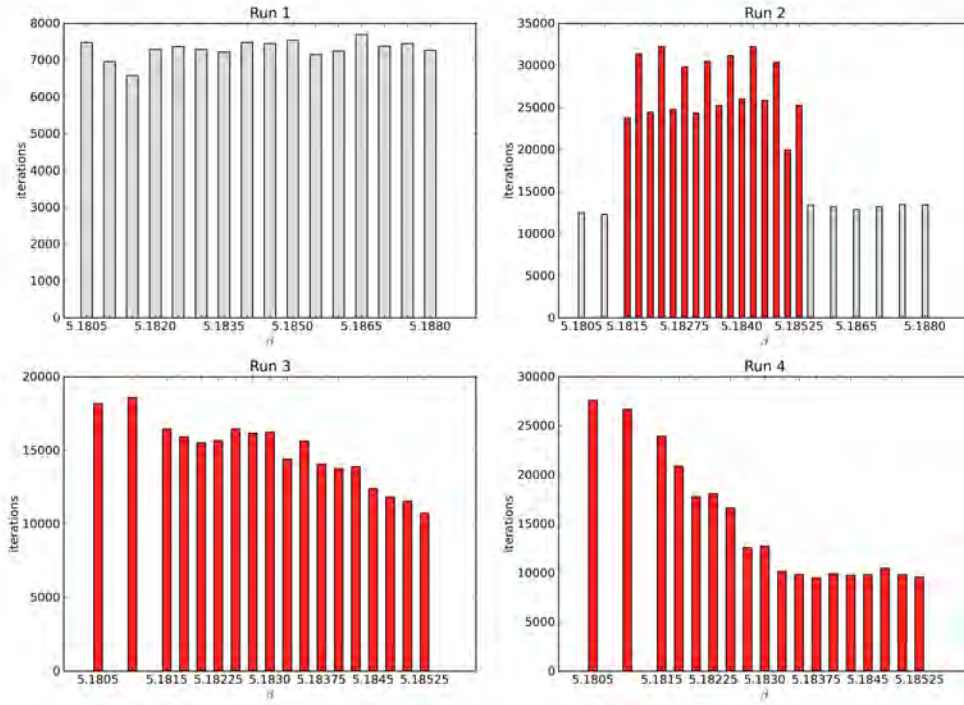


Figure 7.11: Total number of completed iterations in the different β ranges for all runs. The sensitive region is indicated in darker colour (red). The sawtooth pattern visible in run 2 is due to interleaved β -values added after the run started, thus completing less iterations.

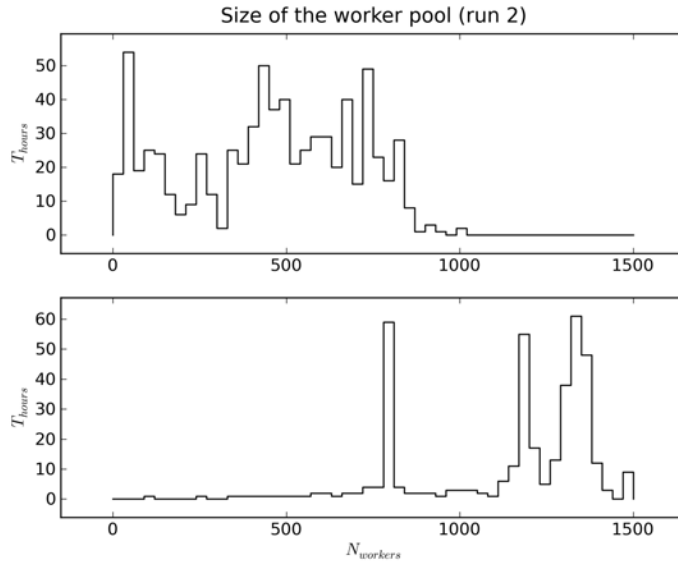


Figure 7.12: Size of the worker pool (active workers) in run 2 for submission without (top) and with (bottom) Heuristic Agent Factory. The histogram shows how many hours a specific number of worker agents was simultaneously used in the simulation system.

the effective allocation of snapshots to workers. The final maturity of snapshots at the end of a run depends on the amount of processing in the function of β and the distribution of processing power of the workers.

7.6 Analysis of adaptive resource selection

In run 1 (Fig. 7.4) and in the first part of run 2 until 15/07 (Fig. 7.5), the workers were added to the pool with manual job submission by users without adhering to any particular submission schedule. In the remainder of run 2 and in runs 3,4 (Figs. 7.6, 7.7) the submission was controlled by the Heuristic Agent Factory (HAF) developed in Chapter 4. HAF was enabled to maintain N active workers in a pool as indicated by F_N events. When HAF is enabled, the number of invalid workers is less scattered and under better control. The resource selection algorithm implemented by HAF reduces the number of invalid worker agents and thus reduces the number of failing jobs flowing in the Grid which have negative impact on scheduling due to premature worker cancellation as described in Section 7.4.3. However, a small background of invalid workers exists and it is a feature of the selection algorithm were a fraction of jobs are submitted to random CEs via the generic slot.

In the case of exceptional events, such as expiry of grid credentials (E), the number

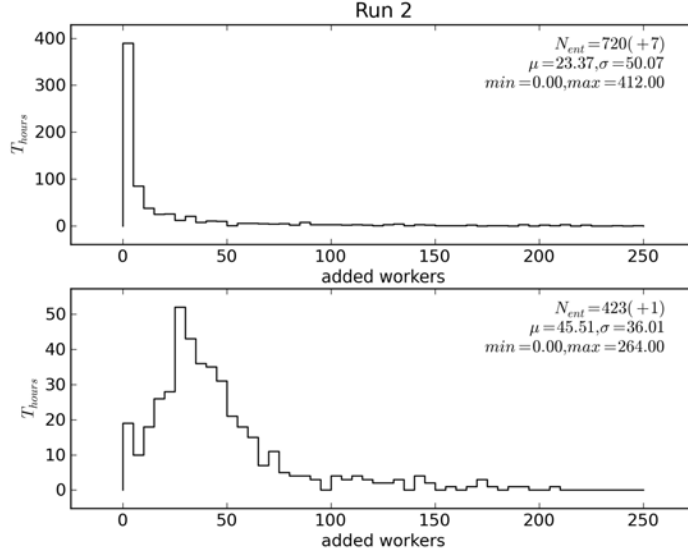


Figure 7.13: Distribution of the number of added workers per hour in run 2 for submission without (top) and with (bottom) Heuristic Agent Factory.

of invalid workers rapidly increases as the number of active workers falls sharply. The number of compatible resources suddenly drops to zero as all new submissions fail and all running workers are interrupted. Such events have a similar impact on the system, independently of whether the workers are submitted manually or via the HAF.

The distribution of the worker pool size in run 2 is shown in Fig. 7.12. In the manual submission mode, the distribution shows a large scatter below the optimal threshold for $N_{workers} = N_{snapshots} = 1450$, which indicates under-provision of the worker agents to the system. In case of HAF, the three clear peaks of the distribution correspond to three stages of the run, as indicated by the events F_{800} , F_{1200} and F_{1450} . When HAF is enabled and resources are available, then the number of active workers quickly converges to the requested level and is maintained for an extended period of time.

The HAF may not maintain the required level of the workers in the pool if there are not enough resources in the Grid. In run 4, the processing enters a low regime phase R , where the amount of available resources is clearly below the optimal target of $N_{workers} = N_{snapshots} = 1063$. Under such conditions the resource selection based on best fitness allows to reduce the number of invalid workers as compared to the manual submission.

Occasionally the HAF leads to oversubmission of worker agents, e.g. F_{1063} in run 4. The default policy of the Agent Factory is to fill up available computing slots within a Computing Element until the worker agents start queuing. If many computing slots become available at the same time in a larger number of Computing Elements, then a

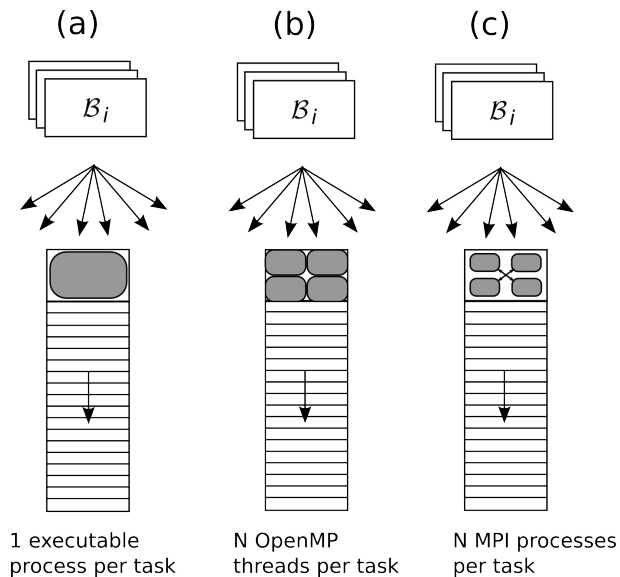


Figure 7.14: Application model deployed on the EGEE Grid (a) compared with low-level parallelism possible with TeraGrid resources using shared-memory OpenMP threads (b) and distributed-memory MPI processes (c). Each simulation track consists of a sequence of tasks which are managed by the User-level Overlay. The realization of a single task differs in all three cases.

large number of queuing jobs suddenly start running and the worker pool could grow beyond the requested size.

The HAF submits workers more efficiently such that a larger number of active workers is added to the system in a time unit. In run 2, the HAF yields an average stream of 45 active worker agents per hour as compared to 23 active worker agents per hour in manual submission mode as visible in Fig. 7.13. The distribution of the number of added workers in a unit of time displays a clear difference in the submission patterns.

Finally, the HAF allows the system to work autonomously and a drastic cut in the time needed for human operation was observed during the study. Only seldom incidents, such as power cuts, needed manual interventions.

7.7 Exploiting low-level parallelism for finer lattices

Our study indicated that for physical quark masses the QCD transition is a crossover at $\mu = 0$ which becomes even softer as a small chemical potential is switched on. It is now most interesting and important to repeat these calculations on finer lattices, in order to see whether this behavior of the chiral critical surface is also realized in continuum QCD.

Increasing the lattice size for our application would require an efficient strategy of

handling $\mathcal{O}(100)$ distributed simulation tracks each being a locally-parallel job with $\mathcal{O}(30)$ processes. First, experiments with GANGA/DIANE overlay extended to use multicore processors showed feasibility of using OpenMP as a parallelization technique if lattice size is increased moderately. Multicore resources available in TeraGrid are accessed via a SAGA thanks to a GANGA plugin. A task corresponds to execution of several OpenMP threads in parallel (with shared memory) on the same computing node (see Fig. 7.14). The task management is handled by the User-level Overlay as in the case of the EGEE Grid. This approach allows to easily reuse existing application code without much refactoring. However, the efficiency of parallelization is limited to a relatively small number of processors – initial tests indicate that no major gain is achieved beyond 4 parallel OpenMP threads per task.

A similar approach, which may be investigated in the future, consists of enabling the application for parallel processing with MPI. The management of tasks and jobs may be done in a similar way as in the case of OpenMP using the User-level Overlay. SAGA-based GANGA plugin could provide simultaneous allocation of groups of worker agent jobs and DIANE framework for subsequent task scheduling such that one group of worker agents handles one task at a time. One task would correspond to a set of MPI processes (with distributed memory) running on a group of closely-connected computing nodes (see Fig. 7.14)

7.8 Summary

We demonstrated that a User-level Overlay may be used efficiently, yielding $\mathcal{O}(10^3)$ speedup, to produce complex scientific results and to process very large workloads over long periods of time (capacity computing). The Lattice QCD application described in this Chapter has the following features: a large granularity (one iteration took over an hour), a small I/O requirement (10 MB per hour or less), and a robust single-CPU code. These features are not typical for other Lattice QCD applications which often simulate too many degrees of freedom to be handled by a single CPU.

For our application, the management and scheduling of $\mathcal{O}(10^3)$ independent simulation tracks was advantageously handled by GANGA/DIANE User-level Overlay. With limited high-level scripting, we plugged into the Master service scheduling algorithms which exploited the knowledge of the status of the simulated lattice. Dynamic resource selection based on the application feedback was automatically provided by Heuristic Agent Factory and allowed to reduce wasted resources to $\mathcal{O}(10\%)$. With the exception of external events such as service power outage or minor manual interventions such as upgrade of the application code or renewal of user credentials to the grid, the system operated autonomously for several months and showed exceptional stability.

For this application the EGEE Grid enabled to obtain scientific results faster and at a lower cost than using massively-parallel computing facilities such as High-Performance Computing centers. While comparable CPU resources are commonly available there, they are usually packaged within expensive massively parallel machines. High-availability computing nodes and high-bandwidth network interconnects are a more expensive option than less reliable, commodity elements used in grids. Grids also enable resource

sharing to lower the total cost of ownership for resource providers, as the computing resources may be leased or borrowed more flexibly, according to current needs. While for this application the initial pre-thermalization phase was conducted using a super-computer in a HPC center, for the bulk of the simulations a very large pool of loosely connected, heterogeneous PCs provided an adequate, cheaper platform.

Finally, it is worth mentioning that the study described in this Chapter has shown that there is no QCD chiral critical point at temperature T and small quark chemical potential μ_q satisfying $\mu_q/T < \mathcal{O}(1)$, on a coarse lattice with 4 time-slices, with a strange quark having its physical mass.

Conclusions and future work

He looked up and said, "there's an infinite number of monkeys outside who want to talk to us about this script for Hamlet they've worked out."

The Hitchhiker's Guide to the Galaxy by Douglas Adams

8.1 Grid dynamics and its consequences for task processing

The development of the EGEE Grid is a major achievement: the EGEE Grid successfully provides a large-scale computing service to numerous research communities. It is the largest grid in operation and it is representative for other grids as a complex and highly dynamic system.

The main structural reasons for the complexity of large grids are 1) large scale of the infrastructure and 2) decentralized organization and operation. As the result grids are highly heterogeneous and integrate computing sites of varying internal structures, computing capacities and sizes which span several orders of magnitude. In addition, grids are mainly composed of commodity elements which expose high failure rates. Individual elements of large grids are maintained, operated and evolved with limited central coordination, and often, without efficient communication and coordination with other stakeholders. Coordination is a non-trivial issue due to conflicting requirements and

priorities of different user groups: end users, local system administrators, middleware development groups and support teams.

Architecture and implementation of middleware services in the EGEE Grid contributes to end users' perception of unreliable and unpredictable system behavior. This is especially true for temporal dynamics related to the submission of jobs due to the early-binding processing model, and inherently inaccurate nature of data provided by information services for the use in workload management. As a result, heavily-tailed job queuing times contribute to low efficiency of task processing and a perception of lower Quality of Service.

A large number of active user communities contributes to observed spatial and temporal dynamics, which may be both short- and long-range. Heavy user communities, such as the LHC experiments, generate considerable and sustained load on large numbers of grid elements. Despite this substantial background, specific workload distribution structures may be distinguished. Temporary pockets of activity are visible in smaller clusters of grid elements and they correspond to smaller-scale analysis activities. Centralized workload distribution patterns in some VOs are visible as consisting of one central element processing a large number of jobs, and a number of smaller "satellite" elements processing less jobs.

In this study we focused on the temporal dynamics resulting in long tails of distributions of job queuing times. It has profound impact on the efficiency of task processing and may lead to complex distributions of makespan with non-trivial dependencies between their statistical properties such as mean and variance. These properties are effectively the measures of Quality of Service from end user's perspective.

8.2 Contributions of this work

This thesis is a contribution to the debate if Quality of Service in grids may be efficiently implemented at the application level. Here we summarize the results of our research for which detailed objectives were stated in section 1.5.

First, we described and analyzed the spatial and temporal dynamics in the world's largest grid in production, the EGEE Grid (Chapter 2). Then, we developed the task processing model for the late-binding scheduling method which explains *why* late binding significantly improves the distribution of makespan, albeit in statistical sense (Chapter 3). Using simple mathematical modeling we concluded that the late-binding method reduces the mean and standard deviation of makespan on a basis of the Central Limit Theorem (Sec. 3.4). We have shown that it is possible to achieve shorter and more predictable processing times above the raw grid processing services, which at their core, however, remain "best-effort". The processing efficiency may be further improved by combining the late-binding scheduling with adaptive workload-balancing and heuristic resource selection. We have developed a User-level Overlay to make such a combination easy and efficient (Chapter 4).

We demonstrated that adaptive workload balancing may give an additional advantage in specific cases, where the scheduling overheads effectively impose limits on task granularity. Unlike other approaches, fine tuning of the workload-balancing algorithm

capability	application
scaling out on demand	ITU RRC06
short-deadline computing	ITU RRC06
application-aware task prioritization	LatticeQCD
autonomous processing	LatticeQCD
heuristic resource selection	LatticeQCD
Grid/HPC interoperation	LatticeQCD
operational efficiency	AvianFlu
web portal integration	AvianFlu
workflow service integration	GATE
shell environment integration	HEP/GANGA
increased predictability of job completion times	Geant4
flexible application interfacing modes	Geant4

Table 8.1: Extended processing capabilities achieved with DIANE/GANGA User-level Overlay.

is done in the interaction with the execution environment. Automatically collected benchmarks allow to respond to possibly varying application and resource characteristics (Sec. 4.8). The resource selection uses simple but efficient heuristics to improve the duty cycle of an application, yet without a priori knowledge of the application itself (Sec. 4.7.3 and 7.6).

The User-level Overlay was developed as an application hosting environment to facilitate gridification of applications within existing boundary conditions for the largest production grids (Sec. 4.1). This strategy has proven efficient and successful in a number of computing activities in diverse application domains (Chapter 5). It allows for various additional capabilities to be achieved, as summarized in Tab. 8.1. It also allows to achieve application-level interoperability between different computing infrastructures, as summarized in Tab. 8.2.

The impact of our work was validated by several examples of successful use of the ideas and tools, which was sometimes undertaken by independent teams in diverse application areas. While presenting such examples we also point out specific characteristics and properties of the User-level Overlay, which may be used as inspiration or even guidelines for future applications.

A unique feature of our User-level Overlay is an efficient support of radically different use-cases:

- capability computing – where tens of thousands of very short tasks must be completed on a strict deadline (Chapter 6), and
- capacity computing – where cumulative resource usage reaches hundreds of CPU-years over several weeks of uninterrupted service (Chapter 7).

We have laid out the key elements of the software design and architecture which allow to achieve such a flexibility with a strict control of the efficiency trade-offs. They

resources	application
interactive clusters, batch farms, EGEE Grid	ATLAS, LHCb, Geant4
desktop-PC farm, EGEE Grid	ITU RRC06
HPC, EGEE, TeraGrid	LatticeQCD
PRAGMA, EGEE Grid	BLAST
Amazon EC2 clouds	Google SoC09

Table 8.2: Summary of cases of application-level interoperability between various distributed infrastructures achieved with DIANE/GANGA User-level Overlay.

include layering with asynchronous component interaction, loose component coupling via a clear separation of concerns and a correct choice of foundation technologies.

Programming which exploits common application patterns [101] in the context of loosely-coupled distributed systems is becoming increasingly important in scientific applications. New trends such as Many Task Computing attempt at bridging the gap between High Performance and High Throughput worlds. The User-level Overlay based on GANGA/DIANE which implements the late-binding task processing model allows to handle such applications with improved efficiency and reliability.

Summing up, in this way the central research hypothesis of this work defined in Sec. 1.5 has been demonstrated. We believe that our strategy represents a serious step forward bridging the gap between the system domain – infrastructure and middleware – and the application domain in large production grids.

8.3 Open issues

In the Introduction we defined the MTAs as a class of applications targeted by the User-level Overlay. Clearly, the Master/Worker coordination backbone which is well adapted for structured forms of parallelism (such as iterative applications, DAGs or workflows), is not ideal for irregular or speculative parallelism where worker processes must communicate in complex ways with one another. Additionally, routing of communication via a central point in a wide-area network may create bottlenecks for applications with very high communication to computation ratio. The User-level Overlay may not be an efficient method for traditional, MPI-style applications although we indicated applications which use the MPI for implementation but may be easily ported to the User-level Overlay.

The User-level Overlay approach is a design trade-off between usability for application providers, typically computing experts within application communities, and runtime efficiency. Our approach is placed mid-way between a generic scheduling service and an embedded scheduler tightly-coupled with the application code. Predefined, standard modules provided with the User-level Overlay allow a straightforward interfacing of legacy and black-box applications. However, applications with complex communication structures must be refactored and instrumented with the scheduling functionality. In spite of many differences, this may be compared to implicit parallelism provided by OpenMP versus explicit refactoring of application code for MPI. Some form of com-

piler support for User-level Overlay, for example similar to OpenMP directives, would be desirable to automate the porting process.

User-level Overlay is a powerful tool which may be abused by a careless user by creating large pools of idle worker agent jobs. This may happen for example by choosing inappropriate configuration parameters or using erroneous application plugins. The risk may be mitigated with off-the-shelf plugins developed and tested by power users and domain experts. Incorrect usage of grid resource is a general concern, however, because grids are open for easy abuse even by fairly unsophisticated means. This is because the fair share at a user-level is not currently implemented in the large production grids, and there are limited ways of controlling the resource usage for individual users.

Existing fair-share models developed for batch systems may be a basis for an additional study of how User-level Overlay affects fair share in grids. This could be beneficial in preparation for the future introduction of fair share in the EGEE Grid, which – will most likely be based on techniques known from batch systems. This could be achieved with a simulation of an environment where a group of “standard” users competes for resources with a group of User-level Overlay users. Such a study could also include a possible job “starvation” and its consequences, especially in situations where the number of available resources is much smaller than the number of user jobs. This could lead to the question if and how the worker agent lifetime should be limited and what is the impact of such a limit. The current implementation allows to define the worker agent time limit as a configuration parameter for individual runs. A strategy for coordination of worker time limits in a user community could require some form of distributed knowledge and decentralized decision process to be put in place.

We observed a practical problem for some users with running of the RunMaster service in their institutes. RunMaster must be instantiated in a host with inbound access from the outside networks. Enabling such access typically requires opening of the firewall by a local system administrator and is not an automatic procedure. This problem could be overcome by the development of a generic virtualization service for running RunMaster components remotely. This requires, however, to understand and manage the right balance between the security and flexibility.

8.4 Future work

In this thesis we focused on analyzing the impact of the job waiting time on task processing, deliberately simplifying other parameters of the model. In the future, for a higher precision of the model, more parameters need to be considered to reflect the distribution of processing speed of grid resources (heterogeneity), variable processing efficiency (time sharing of multi-code nodes) and the presence of resubmission and runtime errors.

Quantitative evaluation and simulation of G and U models for High Throughput application use-cases is also a possible extension of current work. In the High Throughput case, the total workload is so large that for K simultaneously submitted worker agent jobs the maximum allowed usage time of individual computing resources is exceeded, $\frac{W}{K} \gg \nu_{max}$.

Further work is required to quantitatively study the interaction of late-binding

method with fair-share scheduling and mutual exclusion of users competing for the same, scarcely available resources.

Characterizing the late-binding task processing method from the queuing theory perspective is another interesting area of future investigations. User-level Overlay effectively creates a single queue dedicated to one user above a set of distributed queues in the Grid which are shared between many users.

Finally, we want to explore the exponential property of τ distribution and provide more detailed and deeper analysis of the Grid using methods elaborated in the complex systems research area.

8.5 Postscriptum

Stepping away and grasping a macroscopic picture of a large grid, an analogy to a large river immanently appears. The main current – global infrastructures, middleware and virtual organizations – is powerful but slow and steady. At the same time, on the surface fast local currents flow in various directions – local resource providers and user communities – creating whirlpools and backflows. Some travelers – capability computing users – try to cross from one bank to another in a shortest time possible. Some others – capacity computing users – need to cross the river many times and must be sure that this process is trustworthy and easy to foretell. However, the dynamics on the surface are unpredictable and chaotic, so some traveler communities may build dams or bridges – specialized VO services – which then change local dynamics for other travelers. We may see that the system is dynamic but the large-scale landscape is steady. The river lazily swells. But then a new paradigm-shift comes to turn the river around and make it flow in another direction!¹.

We hope that this thesis provides new insights on understanding and improving fundamental properties of large grids. We also hope that it will be possible to carry them through and to apply, some part of them, equally efficiently in the future generation computing systems, after a future, inevitable paradigm shift.

¹ *F. Golden, E. Amfitheatrof – Making Rivers Run Backward; TIME June, 14 1982*

Bibliography

- [1] RFC3501 Internet Message Access Protocol (IMAP).
- [2] *Final Acts of the Regional Radiocommunication Conference for planning of the digital terrestrial broadcasting service in parts of Regions 1 and 3, in the frequency bands 174-230 MHz and 470-862 MHz (RRC-06)*. ITU Conference Publications, 2006.
- [3] LUSTRE: High-performance storage architecture and scalable cluster file system. *White Paper, Sun Microsystems*, 2007.
- [4] Perspectives workshop: The future of grid computing. In *Dagstuhl Seminars*. 2009.
- [5] K. Abbaspour, M. Vejdani, and S. Haghighat. SWAT-CUP: Calibration and uncertainty programs for SWAT. In *Oxley, L. and Kulasiri, D. (eds) MODSIM 2007 International Congress on Modelling and Simulation*.
- [6] D. Abramson, J. Giddy, and L. Kotler. High performance parametric modeling with Nimrod/G: Killer application for the global grid? *Parallel and Distributed Processing Symposium, International*, 0:520, 2000.
- [7] E. Adar. GUESS: a language and interface for graph exploration. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 791–800, New York, NY, USA, 2006. ACM.
- [8] M. Aderholz, K. Amako, E. Auge, G. Bagliesi, L. Barone, G. Battistoni, M. Bernardi, M. Boschini, A. Brunengo, J. J. Bunn, J. Butler, M. Campanella, P. Capiluppi, F. Carminati, M. D’Amato, M. Dameri, A. Di Mattia, A. E. Dorokhov, G. Erbacci, U. Gasparini, F. Gagliardi, I. Gaines, P. Galvez, A. Ghiselli, J. Gordon, C. Grandi, F. Harris, K. Holtman, V. Karimaaki, Y. Karita, J. T. Klem, I. Legrand, M. Leltchouk, D. Linglin, P. Lubrano, L. Luminari,

- A. L. Maslennikov, A. Mattasoglio, M. Michelotto, I. C. McArthur, Y. Morita, A. Nazarenko, H. Newman, V. O'Dell, S. W. O'Neale, B. Osculati, M. Pepe, L. Perini, J. L. Pinfold, R. Pordes, F. Prelz, A. Putzer, S. Resconi, L. Robertson, S. Rolli, T. Sasaki, H. Sato, L. Servoli, R. D. Schaffer, T. L. Schalk, M. Sgaravatto, J. Shiers, L. Silvestris, G. P. Siroli, K. Sliwa, T. Smith, R. Somigliana, C. Stanescu, H. E. Stockinger, D. Ugolotti, E. Valente, C. Vistoli, I. M. Willers, R. P. Wilkinson, and D. O. Williams. Models of networked analysis at regional centres for LHC experiments (MONARC), phase 2 report, 24th march 2000. Technical Report CERN-LCB-2000-001. KEK-2000-8, CERN, Geneva, Apr 2000.
- [9] C. Aftimiei, P. Andreetto, S. Bertocco, S. D. Fina, A. Dorigo, E. Frizziero, A. Giannelle, M. Marzolla, M. Mazzucato, M. Sgaravatto, S. Traldi, and L. Zangrando. Design and implementation of the gLite CREAM job management service. *Future Generation Computer Systems*, 26(4):654 – 667, 2010.
- [10] R. Al-Ali, G. von Laszewski, K. Amin, M. Hategan, O. Rana, D. Walker, and N. Zaluzec. QoS support for high-performance scientific grid applications. In *CCGRID '04: Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, pages 134–143, Washington, DC, USA, 2004. IEEE Computer Society.
- [11] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, K. Lrentey, and F. Spataro. From gridmap-file to VOMS: managing authorization in a grid environment. *Future Gener. Comput. Syst.*, 21(4):549–558, 2005.
- [12] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in high-performance computational grid environments. *Parallel Computing*, 28(5):749 – 771, 2002.
- [13] Allison et al. Geant4 developments and applications. *IEEE Transactions on Nuclear Science*, 53:270–278, 2006. LAL 06-69.
- [14] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS '67 (Spring): Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, New York, NY, USA, 1967. ACM.
- [15] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.
- [16] S. Andreozzi and M. Marzolla. A RESTful approach to the OGSA basic execution service specification. In *ICIW '09: Proceedings of the 2009 Fourth International Conference on Internet and Web Applications and Services*, pages 131–136, Washington, DC, USA, 2009. IEEE Computer Society.
- [17] S. Andreozzi, M. Sgaravatto, and M. C. Vistoli. Sharing a conceptual model of grid resources and services. *CoRR*, cs.DC/0306111, 2003.

- [18] A. Andronico, R. Barbera, A. Falzone, P. Kunszt, G. L. Re, A. Pulvirenti, and A. Rodolico. GENIUS: a simple and easy way to access computational and data grids. *Future Generation Computer Systems*, 19(6):805 – 813, 2003. 3rd biennial International Grid applications-driven testbed event, Amsterdam, The Netherlands, 23-26 September 2002.
- [19] I. Antcheva, M. Ballintijn, B. Bellenot, M. Biskup, R. Brun, N. Buncic, P. Canal, D. Casadei, O. Couet, V. Fine, L. Franco, G. Ganis, A. Gheata, D. G. Maline, M. Goto, J. Iwaszkiewicz, A. Kreshuk, D. M. Segura, R. Maunder, L. Moneta, A. Naumann, E. Offermann, V. Onuchin, S. Panacek, F. Rademakers, P. Russo, and M. Tadel. ROOT – A C++ framework for petabyte data storage, statistical analysis and visualization. *Computer Physics Communications*, 180(12):2499 – 2512, 2009. 40 YEARS OF CPC: A celebratory issue focused on quality software for high performance, grid and novel computing architectures.
- [20] R. Antunes-Nobrega et al. LHCb computing - Technical Design Report CERN/LHCC 2005-019 LHCb TDR-11.
- [21] P. Bar, C. Coti, D. Groen, T. Herault, V. Kravtsov, A. Schuster, and M. Swain. Running parallel applications with topology-aware grid middleware. *IEEE e-Science, 2009.*, pages 292–299, dec. 2009.
- [22] G. Barrand, I. Belyaev, P. Binko, M. Cattaneo, R. Chytrcek, G. Corti, M. Frank, G. Gracia, J. Harvey, E. van Herwijnen, P. Maley, P. Mato, S. Probst, and F. Ranjard. GAUDI – a software architecture and framework for building HEP data processing applications. *Computer Physics Communications*, 140(1-2):45 – 55, 2001.
- [23] D. Beazley. Automated scientific software scripting with SWIG. *Future Generation Computer Systems*, 19(5):599 – 609, 2003. Tools for Program Development and Analysis. Best papers from two Technical Sessions, at ICCS2001, San Francisco, CA, USA, and ICCS2002, Amsterdam, The Netherlands.
- [24] M. Berger and T. Fahringer. Practical experience from porting and executing the Wien2k application on the EGEE production grid infrastructure. *Journal of Grid Computing*, 8:261–279, 2010. 10.1007/s10723-010-9156-x.
- [25] M. Berger, T. Zangerl, and T. Fahringer. Analysis of Overhead and Waiting Time in the EGEE Production Grid. In *Proceedings of the Cracow Grid Workshop 2008*, pages 287–294, 2009.
- [26] R. Berlich, M. Kunze, and K. Schwarz. Grid computing in Europe: from research to deployment. In *ACSW Frontiers '05: Proceedings of the 2005 Australasian workshop on Grid computing and e-research*, pages 21–27, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.
- [27] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and

- D. Zagorodnov. Adaptive computing on the grid using AppLeS. *IEEE Trans. Parallel Distrib. Syst.*, 14(4):369–382, 2003.
- [28] V. Bharadwaj, D. Ghose, V. Mani, and T. G. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, 1996.
- [29] E.-J. Bos, E. Martelli, and P. Moroni. LHC Tier-0 to Tier-1 high-level network architecture. *CERN, Tech. Rep.*, 2005.
- [30] M. Branco, D. Cameron, B. Gaidioz, V. Garonne, B. Koblitz, M. Lassnig, R. Rocha, P. Salgado, and T. Wenaus. Managing ATLAS data on a petabyte-scale with DQ2. *Journal of Physics: Conference Series*, 119(6):062017, 2008.
- [31] M. Bubak, M. Malawski, T. Gubala, M. Kasztelnik, P. Nowakowski, D. Harezlak, T. Bartynski, J. Kocot, E. Ciepiela, W. Funika, D. Krol, B. Balis, M. Assel, and A. T. Ramos. Virtual laboratory for collaborative applications. In M. Cannataro, editor, *Handbook of Research on Computational Grid Technologies for Life Sciences, Biomedicine and Healthcare*, pages 531–551, 2009.
- [32] M. Bubak, J. Mościcki, and J. Shiers. Design of high-performance C++ package for handling of multidimensional histograms. In P. Sloot, M. Bubak, A. Hoekstra, and B. Hertzberger, editors, *High-Performance Computing and Networking*, volume 1593 of *Lecture Notes in Computer Science*, pages 543–552. Springer Berlin / Heidelberg, 1999. 10.1007/BFb0100615.
- [33] R. Buyya, M. Murshed, D. Abramson, and S. Venugopal. Scheduling parameter sweep applications on global grids: a deadline and budget constrained cost-time optimization algorithm. *Softw. Pract. Exper.*, 35(5):491–512, 2005.
- [34] B. Caccia, M. Mattia, G. Amati, C. Andenna, M. Benassi, A. d’Angelo, G. Frustagli, G. Iaccarino, A. Occhigrossi, and S. Valentini. Monte Carlo in radiotherapy: experience in a distributed computational environment. *Journal of Physics: Conference Series*, 74(1):021001, 2007.
- [35] G. Carrera, E. de Andres, J. Mościcki, A. Muraru, S. Scheres, and J. Carazo. Heavy computational tasks on the EGEE grid: 2D/3D maximum-likelihood refinement. Jan. 2007. Network of Excellence 3DEM Annual Meeting, Palma.
- [36] H. Casanova. Benefits and drawbacks of redundant batch requests. *J. Grid Comput.*, 5(2):235–250, 2007.
- [37] S. Chauvie, P. Lorenzo, A. Lechner, J. Mościcki, and M. Pia. Benchmark of medical dosimetry simulation using the Grid. In *IEEE Nuclear Science Symposium Conference Record NSS ’07*, volume 2, pages 1100–1106, 2007.
- [38] K. Christodoulopoulos, V. Gkamas, and E. Varvarigos. Statistical analysis and modeling of jobs in a grid environment. *Springer Journal of Grid Computing*, 6:77–101, November 2007.

- [39] R. Chytracsek, D. Dullmann, M. Frank, M. Girone, G. Govi, J. Mościcki, I. Papadopoulos, H. Schmuecker, K. Karr, D. Malon, A. Vaniachine, W. Tanenbaum, Z. Xie, T. Barrass, and C. Cioffi. LCG POOL development status and production experience. In *IEEE Nuclear Science Symposium Conference Record*, volume 4, pages 2077–2081 Vol. 4, 2004.
- [40] W. Cirne and F. Berman. A model for moldable supercomputer jobs. In *IPDPS '01: Proceedings of the 15th International Parallel & Distributed Processing Symposium*, page 59, Washington, DC, USA, 2001. IEEE Computer Society.
- [41] W. Cirne and F. Berman. Using moldability to improve the performance of supercomputer jobs. *J. Parallel Distrib. Comput.*, 62(10):1571–1601, 2002.
- [42] W. Cirne, F. Brasileiro, D. Paranhos, L. Goes, and W. Voorsluys. On the efficacy, efficiency and emergent behavior of task replication in large distributed systems. *Parallel Computing*, 33:213–234, 2007.
- [43] E. Clevede, D. Weissenbach, and B. Gotab. Distributed jobs on EGEE Grid infrastructure for an Earth science application: moment tensor computation at the centroid of an earthquake. *Earth Science Informatics*, 2:97–106, 2009. 10.1007/s12145-009-0029-4.
- [44] M. Cole. Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming. *Parallel Computing*, 30(3):389 – 406, 2004.
- [45] M. Congreve, C. W. Murray, and T. L. Blundell. Keynote review: Structural biology and drug discovery. *Drug Discovery Today*, 10(13):895 – 907, 2005.
- [46] G. Cooperman, V. H. Nguyen, and I. Malioutov. Parallelization of Geant4 Using TOP-C and Marshagen. In *NCA '06: Proceedings of the Fifth IEEE International Symposium on Network Computing and Applications*, pages 48–55, Washington, DC, USA, 2006. IEEE Computer Society.
- [47] O. Couet, D. Ferrero-Merlino, Z. Molnar, J. Mościcki, A. Pfeiffer, and M. Sang. Anaphe - OO libraries and tools for data analysis. Technical Report CERN-IT-2001-012, CERN, Geneva, Sep 2001.
- [48] K. Czajkowski, I. T. Foster, N. T. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, London, UK, 1998. Springer-Verlag.
- [49] A. E. Darling, L. Carey, and W. chun Feng. The design, implementation, and evaluation of mpiBLAST. In *Proceedings of ClusterWorld 2003*, 2003. Available online.
- [50] P. de Forcrand and O. Philipsen. The QCD phase diagram for three degenerate flavors and small baryon density. *Nucl. Phys. B*, 673:170, 2003.

- [51] P. de Forcrand and O. Philipsen. The chiral critical line of $N_f = 2 + 1$ QCD at zero and non-zero baryon density. *JHEP*, 0701:077, 2007.
- [52] P. de Forcrand and O. Philipsen. The chiral critical point of $N_f = 3$ QCD at finite density to the order $(\mu/T)^4$. *JHEP*, 0811:012, 2008.
- [53] P. de Forcrand and O. Philipsen. The curvature of the critical surface $(m_{ud}, m_s)^{\text{crit}}(\mu)$: a progress report. *PoS LATTICE2008*, page 208, 2008.
- [54] M. de Oliveira Branco. Distributed data management for large scale applications. *PhD thesis*, November 2009.
- [55] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Gener. Comput. Syst.*, 25(5):528–540, 2009.
- [56] M. den Burger, C. Jacobs, T. Kielmann, A. Merzky, O. Weidner, and H. Kaiser. What is the price of simplicity? a cross-platform evaluation of the SAGA API. 2010.
- [57] G. Duckeck et al. ATLAS computing - Technical Design Report CERN/LHCC 2005-022 ATLAS TDR-017 (2005).
- [58] M. Ellert, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. L. Nielsen, M. Niinimäki, O. Smirnova, and A. Wäänänen. Advanced resource connector middleware for lightweight computational grids. *Future Gener. Comput. Syst.*, 23(2):219–240, 2007.
- [59] J. Elmsheuser, F. Brochu, U. Egede, B. Gaidioz, K. Harrison, H. Lee, D. Liko, A. Maier, J. Mościcki, A. Muraru, V. Romanovsky, A. Soroko, and C. Tan. Distributed analysis using Ganga on the EGEE/LCG infrastructure. *Journal of Physics: Conference Series*, 119(7):072014 (8pp), 2008.
- [60] D. G. Feitelson and L. Rudolph. Toward convergence in job schedulers for parallel supercomputers. In *In Job Scheduling Strategies for Parallel Processing*, pages 1–26. Springer-Verlag, 1996.
- [61] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and practice in parallel job scheduling. In *IPPS '97: Proceedings of the Job Scheduling Strategies for Parallel Processing*, pages 1–34, London, UK, 1997. Springer-Verlag.
- [62] F. Foppiano, S. Guatelli, J. Mościcki, and M. Pia. From DICOM to Grid: a dosimetric system for brachytherapy born from HEP. In *IEEE Nuclear Science Symposium Conference Record*, volume 3, pages 1746–1750 Vol.3, 2003.
- [63] D. Forrest and F. J. P. Soler. A new application for the grid: muon ionization cooling for a neutrino factory. *Philos Transact A Math Phys Eng Sci*, 368(1926):4103–13, 2010.

- [64] I. Foster. What is the Grid? - a three point checklist. *GRIDtoday*, 1(6), July 2002.
- [65] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *CCS '98: Proceedings of the 5th ACM conference on Computer and communications security*, pages 83–92, New York, NY, USA, 1998. ACM.
- [66] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, 2001.
- [67] J. Frey, T. Tannenbaum, M. Livny, I. T. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.
- [68] S. Gadowski. Swiss ATLAS computing: the interactive system. *ATLAS Software and Computing Workshop*, May 2005.
- [69] M. Gallas, J. Mościcki, M. Lamanna, and L. Mancera. Quality assurance and testing in LCG. In CERN, editor, *Computing for High Energy Physics, 2004*. Interlaken (Switzerland), September 2004.
- [70] Y. P. Galyuk, V. Memnonov, S. E. Zhuravleva, and V. I. Zolotarev. Grid technology with dynamic load balancing for Monte Carlo simulations. In *PARA '02: Proceedings of the 6th International Conference on Applied Parallel Computing Advanced Scientific Computing*, pages 515–520, London, UK, 2002. Springer-Verlag.
- [71] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.
- [72] W. Gentzsch. Sun Grid Engine: Towards creating a compute power grid. In *CC-GRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 35, Washington, DC, USA, 2001. IEEE Computer Society.
- [73] C. Germain-Renaud, C. Loomis, J. Mościcki, and R. Texier. Scheduling for responsive Grids. *J. Grid Computing*, 6:15–27, 2008.
- [74] C. Germain-Renaud, R. Texier, and A. Osorio. Interactive reconstruction and measurement on the grid. *Methods of Information in Medicine*, 44(2):227–232, 2005.
- [75] T. Glatard and S. Camarasu-Pop. Modelling pilot-job applications on production grids. In *Euro-Par Workshops*, pages 140–149, 2009.
- [76] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec. Flexible and Efficient Workflow Deployment of Data-Intensive Applications On Grids With MOTEUR. *International Journal of High Performance Computing Applications*, 22(3):347–360, 2008.

- [77] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, G. V. Laszewski, C. Lee, A. Merzky, H. Rajic, and J. Shalf. SAGA: A simple API for grid applications. high-level application programming on the grid. In *Computational Methods in Science and Technology*, volume 12, pages 7–20, 2006.
- [78] X. Grehant and I. Demeure. Symmetric mapping: An architectural pattern for resource supply in grids and clouds. volume 0, pages 1–8, Los Alamitos, CA, USA, 2009. IEEE Computer Society.
- [79] I. J. Grimstead, N. J. Avis, and D. W. Walker. RAVE: the resource-aware visualization environment. *Concurr. Comput. : Pract. Exper.*, 21(4):415–448, 2009.
- [80] D. Groen. Reliability analysis of grid resources: A user perspective – UvA MSc thesis, 2006.
- [81] D. Groen, S. Harfst, and S. Portegies Zwart. On the origin of grid species: The living application. In *ICCS '09: Proceedings of the 9th International Conference on Computational Science*, pages 205–212, Berlin, Heidelberg, 2009. Springer-Verlag.
- [82] G. Grzeslo, T. Szepieniec, and M. Bubak. DAG4DIANE - enabling DAG-based applications on DIANE framework. *CGW Book of Abstracts*, 2009.
- [83] S. Guatelli, A. Mantero, P. Mendez-Lorenzo, J. Mościcki, and M. Pia. Geant4 simulation in a distributed computing environment. In *IEEE Nuclear Science Symposium Conference Record, 2006*, volume 1, pages 110–113, 2006.
- [84] S. Guatelli, M. Reinhard, B. Mascialino, D. Prokopovich, A. Dzurak, M. Zaider, and A. Rosenfeld. Tissue equivalence correction in silicon microdosimetry for protons characteristic of the LEO space environment. *Nuclear Science, IEEE Transactions on*, 55(6):3407–3413, dec. 2008.
- [85] T. Gubala, M. Bubak, and P. Sloot. Semantic integration for research environments. In *M. Cannataro, editor, Handbook of Research on Computational Grid Technologies for Life Sciences, Biomedicine and Healthcare*, pages 514–530, 2009.
- [86] J. L. Gustafson. Reevaluating Amdahl’s law. *Commun. ACM*, 31(5):532–533, 1988.
- [87] E. M. Heien, Y. Takata, K. Hagihara, and A. Kornafeld. PyMW - a Python module for desktop grid and volunteer computing. *Parallel and Distributed Processing Symposium, International*, 0:1–7, 2009.
- [88] R. L. Henderson. Job scheduling under the Portable Batch System. In *IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 279–294, London, UK, 1995. Springer-Verlag.
- [89] A. J. G. Hey and A. E. Trefethen. The data deluge: An e-science perspective. *Grid Computing - Making the Global Infrastructure a Reality*, pages 809–824, 2003.

- [90] A. Howard and H. Araujo. Simulation and analysis for astroparticle experiments. *Nuclear Physics B - Proceedings Supplements*, 125:320 – 326, 2003. Innovative Particle and Radiation Detectors.
- [91] E. Huedo, R. S. Montero, and I. Llorente. The GridWay framework for adaptive scheduling and execution on grids. *Scalable Computing - Practice and Experience*, 6(3):1–8, 2005.
- [92] E. Huedo, R. S. Montero, and I. M. Llorente. Evaluating the reliability of computational grids from the end user’s point of view. *Journal of Systems Architecture*, 52(12):727 – 736, 2006.
- [93] L. Ilijašić and L. Saitta. Characterization of a computational grid as a complex system. In *GMAC ’09: Proceedings of the 6th international conference industry session on Grids meets autonomic computing*, pages 9–18, New York, NY, USA, 2009. ACM.
- [94] A. Iosup, C. Dumitrescu, D. Epema, H. Li, and L. Wolters. How are real grids used? the analysis of four grid traces and its implications. In *Grid Computing, 7th IEEE/ACM International Conference on*, pages 262–269, Sept. 2006.
- [95] K. A. Iskra, F. van der Linden, Z. W. Hendrikse, B. J. Overeinder, G. D. van Albada, and P. M. A. Sloot. The implementation of dynamite: an environment for migrating PVM tasks. *SIGOPS Oper. Syst. Rev.*, 34(3):40–55, 2000.
- [96] ITU. Constitution of the ITU, Chapter VII, Art. 44, “Use of the Radio-Frequency Spectrum and of the Geostationary-Satellite and Other Satellite Orbits”, 1992.
- [97] ITU. Method for point-to-area predictions for terrestrial services in the frequency range 30 MHz to 3 000 MHz. *ITU-R P.1546-4*, 2009.
- [98] N. Jacq, V. Breton, H.-Y. Chen, L.-Y. Ho, M. H. 0009, H.-C. Lee, Y. Legré, S. C. Lin, A. Maaß, E. Medernach, I. Merelli, L. Milanesi, G. Rastelli, M. Reichstadt, J. Salzemann, H. Schwichtenberg, M. Sridhar, V. Kasam, Y.-T. Wu, and M. Zimmermann. Grid-enabled high throughput virtual screening. In *GCCB*, pages 45–59, 2006.
- [99] N. Jacq, J. Salzemann, F. Jacq, Y. Legré, E. Medernach, J. Montagnat, A. Maaß, M. Reichstadt, H. Schwichtenberg, M. Sridhar, V. Kasam, M. Zimmermann, M. Hofmann, and V. Breton. Grid-enabled virtual screening against malaria. *J. Grid Comput.*, 6(1):29–43, 2008.
- [100] S. Jan, G. Santin, D. Strul, S. Staelens, K. Assie, D. Autret, S. Avner, R. Barbier, M. Bardies, P. M. Bloomfield, D. Brasse, V. Breton, P. Bruyndonckx, I. Buvat, A. F. Chatziioannou, Y. Choi, Y. H. Chung, C. Comtat, D. Donnarieix, L. Ferrer, S. J. Glick, C. J. Groiselle, D. Guez, P. F. Honore, S. Kerhoas-Cavata, A. S. Kirov, V. Kohli, M. Koole, M. Krieguer, D. J. van der Laan, F. Lamare, G. Largeron, C. Lartizien, D. Lazaro, M. C. Maas, L. Maigne, F. Mayet, F. Melot, C. Merheb,

- E. Pennacchio, J. Perez, U. Pietrzyk, F. R. Rannou, M. Rey, D. R. Schaart, C. R. Schmidtlein, L. Simon, T. Y. Song, J. M. Vieira, D. Visvikis, R. V. de Walle, E. Wieers, and C. Morel. GATE: a simulation toolkit for PET and SPECT. *Phys Med Biol*, 49(19):4543–4561, Oct 2004.
- [101] S. Jha, M. Cole, D. S. Katz, M. Parashar, O. R. Rana, and J. Weissman. Abstractions for large-scale distributed applications and systems. *ACM Surveys*, 2009. Available online.
- [102] N. T. Karonis, B. Toonen, and I. Foster. MPICH-G2: A grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing*, 63(5):551 – 563, 2003. Special Issue on Computational Grids.
- [103] F. Karsch, E. Laermann, and C. Schmidt. The chiral critical point in 3-flavor QCD. *Phys. Lett. B*, 520:41, 2001.
- [104] B. Koblitz, N. Santos, and V. Pose. The AMGA metadata service. *Journal of Grid Computing*, 6:61–76, 2008. 10.1007/s10723-007-9084-6.
- [105] V. Korkhov and V. Krzhizhanovskaya. Benchmarking and adaptive load balancing of the virtual reactor application on the Russian-Dutch Grid. In *Proceedings of the 6th International Conference on Computational Science*, volume 3991 of *Lecture Notes in Computer Science*, pages 530–538, Reading, UK, 2006. Springer Berlin, Heidelberg.
- [106] V. Korkhov, V. Krzhizhanovskaya, and P. Sloot. A grid-based virtual reactor: Parallel performance and adaptive load balancing. *Journal of Parallel and Distributed Computing*, 68(5):596–608, 2008.
- [107] V. Korkhov, J. Mościcki, and V. Krzhizhanovskaya. The user-level scheduling of divisible load parallel applications with resource selection and adaptive workload balancing on the Grid. *IEEE Systems Journal*, 3:121–130, March 2009.
- [108] V. Korkhov, J. T. Mościcki, and V. Krzhizhanovskaya. Dynamic workload balancing of parallel applications with user-level scheduling on the Grid. *Future Generation Computer Systems*, 25(1):28 – 34, 2009.
- [109] T. Kosar and M. Balman. A new paradigm: Data-aware scheduling in grid computing. *Future Generation Computer Systems*, 25(4):406 – 413, 2009.
- [110] J. Kosinski, P. Nawrocki, D. Radziszowski, K. Zielinski, S. Zielinski, G. Przybylski, and P. Wnek. SLA monitoring and management framework for telecommunication services. In *ICNS*, pages 170–175, 2008.
- [111] S. Krishnan, P. Wagstrom, and G. V. Laszewski. Gsfl: A workflow framework for grid services. Technical report, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439, 2002.

- [112] K. Lagouvardos, E. Floros, and V. Kotroni. A grid-enabled regional-scale ensemble forecasting system in the Mediterranean area. *Journal of Grid Computing*, 8:181–197, 2010. 10.1007/s10723-010-9150-3.
- [113] E. Laure, C. Gr, S. Fisher, A. Frohner, P. Kunszt, A. Krennek, O. Mulmo, F. Pacini, F. Prelz, J. White, M. Barroso, P. Buncic, R. Byrom, L. Cornwall, M. Craig, A. D. Meglio, A. Djaoui, F. Giacomini, J. Hahkala, F. Hemmer, S. Hicks, A. Edlund, A. Maraschini, R. Middleton, M. Sgaravatto, M. Steenbakkers, J. Walk, and A. Wilson. Programming the Grid with gLite. In *Computational Methods in Science and Technology*, volume 12, pages 33–45, 2006.
- [114] K. Leal, E. Huedo, and I. M. Llorente. A decentralized model for scheduling independent tasks in federated grids. *Future Generation Computer Systems*, 25(8):840 – 852, 2009.
- [115] H.-C. Lee et al. Grid-enabled high-throughput in silico screening against Influenza A Neuraminidase. *IEEE Transactions on NanoBioscience*, 5:288–295, 2006.
- [116] A. Lehmann et al. The Black Sea catchment observation system built on a grid-enabled spatial data infrastructure. In *INSPIRE, GMES and GEOSS Activities, Methods and Tools towards a Single Information Space in Europe for the Environment*, 2009.
- [117] H. Li and R. Buyya. Model-based simulation and performance evaluation of grid scheduling strategies. *Future Gener. Comput. Syst.*, 25(4):460–465, 2009.
- [118] D. Lingrand, J. Montagnat, J. Martyniak, and D. Colling. Analyzing the EGEE production grid workload: Application to jobs submission optimization. *Job Scheduling Strategies for Parallel Processing: 14th International Workshop, JSSPP 2009, Rome, Italy, May 29, 2009. Revised Papers*, pages 37–58, 2009.
- [119] C. Loomis. The grid observatory. In *GMAC '09: Proceedings of the 6th international conference industry session on Grids meets autonomic computing*, pages 41–42, New York, NY, USA, 2009. ACM.
- [120] T. Maeno. PanDA: distributed production and distributed analysis system for ATLAS. *Journal of Physics: Conference Series*, 119(6):062036 (4pp), 2008.
- [121] A. Maier, F. Brochu, G. Cowan, U. Egede, J. Elmsheuser, B. Gaidioz, K. Harrison, H.-C. Lee, D. Liko, J. Mościcki, A. Muraru, K. Pajchel, W. Reece, B. Samset, M. Slater, A. Soroko, D. van der Ster, M. Williams, and C. L. Tan. User analysis of LHCb data with Ganga. *Journal of Physics: Conference Series*, 219(7):072008, 2010.
- [122] L. Maigne, D. Hill, P. Calvat, V. Breton, D. Lazaro, R. Reuillon, Y. Légré, and D. Donnarieix. Parallelization of Monte-Carlo simulations and submission to a grid environment. In *Parallel Processing Letters HealthGRID 2004*, volume 14, pages 177–196, Clermont-Ferrand France, 2004.

- [123] M. Malawski, T. Bartynski, and M. Bubak. Invocation of operations from script-based grid applications. *Future Generation Computer Systems*, 26(1):138 – 146, 2010.
- [124] A. Mantero, B. Bavdaz, A. Owens, T. Peacock, and M. Pia. Simulation of x-ray fluorescence and application to planetary astrophysics. In *Nuclear Science Symposium Conference Record, 2003 IEEE*, volume 3, pages 1527 – 1529 Vol.3, oct. 2003.
- [125] A. N. Marty, M. A. Humphrey, and A. S. Grimshaw. Capacity and capability computing using Legion. In *Proceedings of the 2001 International Conference on Computational Science (ICCS, 2001)*.
- [126] M. Marzolla, P. Andreetto, V. Venturi, A. Ferraro, S. Memon, S. Memon, B. Twedell, M. Riedel, D. Mallmann, A. Streit, S. v. d. Berghe, V. Li, D. Snelling, K. Stamou, Z. A. Shah, and F. Hedman. Open standards-based interoperability of job submission and management interfaces across the grid middleware platforms glite and UNICORE. In *E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, pages 592–601, Washington, DC, USA, 2007. IEEE Computer Society.
- [127] M. Mascagni and Y. Li. Computational infrastructure for parallel, distributed, and grid-based Monte Carlo computations. In *LSSC*, pages 39–52, 2003.
- [128] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.
- [129] E. Medernach. Workload analysis of a cluster in a grid environment. In *JSSPP*, pages 36–61, 2005.
- [130] R. Mendez-Lorenzo, J. Mościcki, and A. Ribon. Experiences in the gridification of the Geant4 toolkit in the WLCG/EGEE environment. In *IEEE Nuclear Science Symposium Conference Record*, volume 2, pages 879–884, 2006.
- [131] T. Mitchel. *Machine Learning*. McGraw Hill Higher Education, 1997.
- [132] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. Rosenthal, and F. D. Smith. Andrew: a distributed personal computing environment. *Commun. ACM*, 29(3):184–201, 1986.
- [133] J. Mościcki. DIANE - distributed analysis environment for grid-enabled simulation and analysis of physics data. In *IEEE Nuclear Science Symposium Conference Record*, volume 3, pages 1617–1620 Vol.3, 2003.
- [134] J. Mościcki. Distributed analysis environment for HEP and interdisciplinary applications. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 502(2-3):426 – 429, 2003. Proceedings of the VIII International Workshop on Advanced Computing and Analysis Techniques in Physics Research.

- [135] J. Mościcki. The DIANE user-scheduler provides quality of service. *CERN Computer Newsletter*, 9 2006.
- [136] J. Mościcki, F. Brochu, J. Ebke, U. Egede, J. Elmsheuser, K. Harrison, R. Jones, H. Lee, D. Liko, A. Maier, A. Muraru, G. Patrick, K. Pajchel, W. Reece, B. Samset, M. Slater, A. Soroko, C. Tan, D. van der Ster, and M. Williams. Ganga: A tool for computational-task management and easy access to Grid resources. *Computer Physics Communications*, 180(11):2303 – 2316, 2009.
- [137] J. Mościcki, M. Bubak, H. Lee, A. Muraru, and P. Sloot. Quality of service on the grid with user level scheduling. In M. Bubak, M. Turala, and K. Wiatr, editors, *CGW'06 Proceedings*, pages 119–129. 2007.
- [138] J. Mościcki, S. Guatelli, A. Mantero, and M. Pia. Distributed Geant4 simulation in medical and space science applications using DIANE framework and the Grid. *Nuclear Physics B - Proceedings Supplements*, 125:327 – 331, 2003.
- [139] J. Mościcki, M. Lamanna, M. Bubak, and P. Sloot. Processing moldable tasks on the Grid: late job binding with lightweight User-level Overlay. (*accepted for publication*) in *Future Generation Computer Systems*, 2011.
- [140] J. Mościcki, H. Lee, S. Guatelli, S. Lin, and M. Pia. Biomedical applications on the Grid: efficient management of parallel jobs. In *IEEE Nuclear Science Symposium Conference Record, 2004*, volume 4, pages 2143 – 2147, 2004.
- [141] J. Mościcki, A. Manara, M. Lamanna, P. Mendez, and A. Muraru. Dependable distributed computing for the International Telecommunication Union Regional Radio Conference RRC06. *CERN Technical Report*, *arxiv:0906.2143*, 2009.
- [142] J. T. Mościcki, M. Wos, M. Lamanna, P. de Forcrand, and O. Philipsen. Lattice QCD thermodynamics on the Grid. *Computer Physics Communications*, 181(10):1715 – 1726, 2010.
- [143] K. Neocleous, M. D. Dikaiakos, P. Fragopoulou, and E. Markatos. Grid reliability: A study of failures on the egee infrastrcture. In *CoreGRID Workshop on Grid Systems, Tools and Environments in Conjunction with GRIDS@work: CoreGRID Conference, Grid Plugtests and Contest*, Sophia-Antipolis, France, December 2006.
- [144] B. C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, 32:33–38, 1994.
- [145] S. Newhouse. The EGEE distributed computing infrastructure. *Connexions*, September 21, 2009.
- [146] G. J. v. t. Noordende, S. D. Olabarriaga, M. R. Koot, and C. T. A. M. d. Laat. A trusted data storage infrastructure for grid-based medical applications. In *CC-GRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, pages 627–632, Washington, DC, USA, 2008. IEEE Computer Society.

- [147] S. D. Olabarriaga, T. Glatard, and P. T. de Boer. A virtual laboratory for medical image analysis. *Information Technology in Biomedicine, IEEE Transactions on*, 14(4):979–985, July 2010.
- [148] T. E. Oliphant. Python for scientific computing. *Computing in Science and Engineering*, 9:10–20, 2007.
- [149] G. Pallis, A. Katsifodimos, and M. Dikaiakos. Searching for software on the EGEE infrastructure. *Journal of Grid Computing*, 8:281–304, 2010. 10.1007/s10723-010-9155-y.
- [150] S. K. Paterson and A. Maier. Distributed data analysis in LHCb. *Journal of Physics: Conference Series*, 119(7):072026, 2008.
- [151] F. Perez and B. E. Granger. IPython: A system for interactive scientific computing. *Computing in Science and Engineering*, 9:21–29, 2007.
- [152] A. Pfeiffer, L. Moneta, V. Innocente, H. C. Lee, and W. L. Ueng. The LCG PI Project: Using Interfaces for Physics Data Analysis. *IEEE Transactions on Nuclear Science*, 52:2823–2826, Dec. 2005.
- [153] C. Pinchak, P. Lu, and M. Goldenberg. Practical heterogeneous placeholder scheduling. In *In Proc. 8th Workshop on Job Scheduling Strategies for Parallel Processing JSSPP*, pages 85–105. Springer Verlag, 2002.
- [154] S. C. Pop, T. Glatard, J. Mościcki, H. Benoit-Cattin, and D. Sarrut. Dynamic partitioning of GATE Monte-Carlo simulations on EGEE. *J. Grid Computing*, 8(2):241–259, 2010.
- [155] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Wurthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. McGee, and R. Quick. The open science grid. *Journal of Physics: Conference Series*, 78(1):012057, 2007.
- [156] G. L. Presti, O. Barring, A. Earl, R. M. G. Rioja, S. Ponce, G. Taurelli, D. Waldron, and M. C. D. Santos. CASTOR: A distributed storage resource facility for high performance data processing at CERN. *Mass Storage Systems and Technologies, IEEE / NASA Goddard Conference on*, 0:275–280, 2007.
- [157] R. Procassini, M. O’Brien, and J. Taylor. Load Balancing of Parallel Monte Carlo Transport Calculations. In *Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications*, Palais des Papes, Avignon, Fra, Sept. 2005.
- [158] I. Raicu, Z. Zhang, M. Wilde, I. Foster, P. Beckman, K. Iskra, and B. Clifford. Toward loosely coupled programming on petascale systems. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing, SC ’08*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

- [159] Y. Robert and F. Vivien. *Introduction to Scheduling*. CRC Press, Inc., Boca Raton, FL, USA, 2009.
- [160] A. Roy and V. Sander. GARA: a uniform quality of service architecture. pages 377–394, 2004.
- [161] P. Saiz et al. AliEn–ALICE environment on the Grid. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 502:437–440, 2003.
- [162] J. M. Schopf and B. Nitzberg. Grids: The top ten questions. *Sci. Program.*, 10(2):103–111, 2002.
- [163] U. Schwickerath and V. Lefebure. Usage of LSF for batch farms at CERN. *Journal of Physics: Conference Series*, 119(4):042025, 2008.
- [164] I. Sfiligoi. GlideinWMS – a generic pilot-based workload management system. *Journal of Physics: Conference Series*, 119(6):062044, 2008.
- [165] G. Shao, F. Berman, and R. Wolski. Master/slave computing on the grid. In *Heterogeneous Computing Workshop*, pages 3–16, 2000.
- [166] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI-The Complete Reference, Volume 1: The MPI Core*. MIT Press, Cambridge, MA, USA, 1998.
- [167] I. Stokes-Rees, A. Tsaregorodtsev, V. Garonne, R. Graciani, M. Sanchez, M. Frank, and J. Closier. Developing LHCb Grid software: experiences and advances. *Concurrency and Computation: Practice and Experience*, 19(2):133–152, 2007.
- [168] A. Streit, D. Erwin, T. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and P. Wieder. Unicore – from project results to production grids. In L. Grandinetti, editor, *Grid Computing The New Frontier of High Performance Computing*, volume 14 of *Advances in Parallel Computing*, pages 357 – 376. North-Holland, 2005.
- [169] W.-J. Tan, C. T. M. Ching, S. Camarasu-Pop, P. Calvat, and T. Glatard. Two experiments with application-level quality of service on the EGEE Grid. In *GMAC '10: Proceeding of the 2nd workshop on Grids meets autonomic computing*, pages 11–20, New York, NY, USA, 2010. ACM.
- [170] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.
- [171] F. Tischler and A. Uhl. Limitations of cluster computing in a communication intensive multimedia application. In M. Vajtersic, R. Trobec, P. Zinterhof, and A. Uhl, editors, *PARALLEL NUMERICS 05 – Theory and Applications*. 2005.

- [172] V. Tola, F. Lillo, M. Gallegati, and R. N. Mantegna. Cluster analysis for portfolio optimization. *Journal of Economic Dynamics and Control*, 32(1):235 – 258, 2008. Applications of statistical physics in economics and finance.
- [173] P. Tollman, P. Guy, J. Altshuler, A. Flanagan, and M. Steiner. A revolution in R&D: How genomics and genetics are transforming the biopharmaceutical industry. *BCG Report*, 2002.
- [174] C. Town and K. Harrison. Large-scale grid computing for content-based image retrieval. *ISKO (International Society for Knowledge Organization) conference on "Content Architecture: Exploiting and Managing Diverse Resources"*, 2009.
- [175] C. Town and D. Sinclair. Language-based querying of image collections on the basis of an extensible ontology. *Image and Vision Computing*, 22(3):251 – 267, 2004.
- [176] A. Tsaregorodtsev et al. DIRAC: A community grid solution. *J. Phys. Conf. Ser.*, 119:062048, 2008.
- [177] D. C. Vanderster, J. Elmsheuser, M. Biglietti, F. Galeazzi, C. Serfon, and M. Slater. Functional and large-scale testing of the ATLAS distributed analysis facilities with Ganga. *Journal of Physics: Conference Series*, 219(7):072021, 2010.
- [178] R. Veenhof. Garfield, recent developments. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 419(2-3):726 – 730, 1998.
- [179] Z. Wandan, C. Guiran, Z. Dengke, and Z. Xiuying. G-RSVPM: A grid resource reservation model. In *SKG '05: Proceedings of the First International Conference on Semantics, Knowledge and Grid*, page 79, Washington, DC, USA, 2005. IEEE Computer Society.
- [180] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2009.
- [181] E. J. Whitehead, Jr. World Wide Web distributed authoring and versioning (webdav): an introduction. *StandardView*, 5(1):3–8, 1997.
- [182] M. Wiczeorek, A. Hoheisel, and R. Prodan. Towards a general model of the multi-criteria workflow scheduling on the grid. *Future Generation Computer Systems*, 25(3):237 – 256, 2009.
- [183] R. Wolski, N. T. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5-6):757 – 768, 1999.

Summary

Scientific communities are using a growing number of distributed systems, from local batch systems, community-specific services and supercomputers to general-purpose, global grid infrastructures. Increasing the research capabilities for science is the *raison d'être* of such infrastructures which provide access to diversified computational, storage and data resources at large scales. Grids are rather chaotic, highly heterogeneous, decentralized systems where unpredictable workloads, component failures and variability of execution environments are commonplace. Understanding and mastering the heterogeneity and dynamics of such distributed systems is prohibitive for end users if they are not supported by appropriate methods and tools. The time cost to learn and use the interfaces and idiosyncrasies of different distributed environments is another challenge.

Obtaining more reliable application execution times and boosting parallel speedup are important to increase the research capabilities of scientific communities. Late binding is one of techniques to achieve these goals because the majority of jobs which are in production in grids and supercomputers are moldable. Moldable jobs may use variable number of resources and be more flexibly partitioned than classical, rigid parallel jobs. Moldable job application examples include Monte Carlo simulations, parameter sweeps, directed acyclic graphs and workflows, data-parallel analysis algorithms and many more.

We analyze spatial and temporal dynamics and study the performance variations in large, loosely coupled distributed systems such as the EGEE Grid – the largest Grid infrastructure to date. We develop a mathematical description of task processing in the Grid, where system parameters are taken as random variables with empirical distributions. We analyze the Quality of Service indicators such as variance of makespan to qualitatively compare late and early-binding task processing models. Using a continuous approximation we analytically demonstrate that properties of the late-binding model allow to reduce the makespan distribution according to fundamental laws of statistics. To analyze the discrete cases and more complex parameters, including the communication overheads, we use Monte Carlo simulation. We identify that under certain conditions late binding allows to achieve speedups which are often greater than an

order of magnitude compared to early binding.

We describe the principles guiding the development of a lightweight, User-level Overlay which exploits late binding to achieve an improved Quality of Service in unreliable and unpredictable distributed environments. Our strategy is based on loosely coupled, user-space tools, where the DIANE scheduler manages task allocation in a pool of worker nodes which is asynchronously created and managed by the GANGA interface. This approach makes it easy (1) to create resource selection mechanisms such as the heuristic-based worker agent factory, and (2) to plug-in adaptive workload-balancing algorithms for task scheduling. Other key features include an ability to interface to a wide range of distributed systems; an ability to extend and customize the system with application-specific scheduling and processing methods; ease of use and uniform interface to heterogeneous job management systems.

Using real-life applications in the EGEE Grid, local batch systems and dedicated clusters we demonstrate new and improved capabilities which are provided by the GANGA/DIANE User-level Overlay above generic middleware stack. These capabilities include efficient short-deadline computing, increased dependability, autonomous large-scale operations, efficient parameter sweeps, man-in-the loop scenarios, automated DAGs/workflows and semi-interactivity.

We present two case-studies of capacity and capability computing with the User-level Overlay. We show how a large number of tasks with a short deadline was coordinated on the Grid to improve dependability of locally available resources for the International Telecommunication Union Regional Radio Conference 2006. Then we describe how task prioritization and resource selection was implemented for the Lattice QCD simulations for the QCD thermodynamics studies in the context of heavy-ion collisions experiments (LHC, RHIC).

This work is a contribution to the debate if Quality of Service in grids may be efficiently implemented at the application level. We demonstrated that it is indeed possible (1) by giving a theoretical explanation of the effects of late binding on key task processing metrics, and (2) by showing examples of applications which successfully applied the User-level Overlay.

Nederlandse samenvatting

Wetenschappelijke gemeenschappen maken in toenemende mate gebruik van gedistribueerde systemen, lokale batch systemen, gemeenschap-specifieke systemen, supercomputers en wereldwijde grid-infrastructuren. De toename van onderzoeksmogelijkheden voor de wetenschap rechtvaardigt dergelijke infrastructuren, die toegang verschaffen tot grootschalige en diverse rekenkracht, opslag en dataverwerking. Grids zijn tamelijk chaotische en zeer heterogene, gedecentraliseerde systemen waar in de regel sprake is van onvoorspelbare belasting, falen van componenten en variatie in het executieplatform. Het begrijpen en beheersen van de heterogeniteit en dynamiek van dergelijke gedistribueerde systemen is voor de eindgebruikers alleen mogelijk indien zij de benodigde methodieken en werktuigen weten te gebruiken. Een andere uitdaging is de noodzakelijke tijdsinvestering voor het begrijpen en gebruiken van de interfaces van de verschillende gedistribueerde omgevingen.

Meer voorspelbare applicatie-executietijden en vergroting van bandbreedte door middel van parallellisatie zijn belangrijke factoren voor een toename van onderzoeksmogelijkheden voor wetenschappelijke gemeenschappen. ‘Late binding’ is een van de technieken om deze doelen te verwezelijken, omdat het grootste deel van de taken binnen grids en supercomputers ‘moldable’ (vervormbaar) zijn. ‘Moldable’ taken kunnen gebruik maken van een variabel aantal resources en kunnen flexibeler gepartitioneerd worden dan klassieke, rigide, parallelle taken. Voorbeelden van ‘moldable’ taken zijn Monte Carlo simulaties, parameter-sweep applicaties, acyclische grafen en workflows, parallelle data analyse algoritmes etc.

We analyseren ruimtelijke en temporele dynamieken en bestuderen de variaties in prestatie in grote, ‘loosely coupled’ gedistribueerde systemen zoals het EGEE Grid, tot op heden de grootste grid infrastructuur. We ontwikkelen een mathematische beschrijving voor taakverwerking binnen het grid, waar we systeemparemeters als stochastische variabelen beschouwen met empirische distributies. We analyseren ‘Quality of Service’ indicatoren zoals variatie in ‘makespan’ om tot een vergelijking van ‘late’ en ‘early-binding’ taakverwerkingsmodellen te komen. Gebruik makend van een continue

benadering tonen we analytisch aan dat eigenschappen van het ‘late-binding’ model het mogelijk maken om de ‘makespan’ distributie volgens fundamentele wetten van de statistiek te laten afnemen. Voor een analyse van discrete modellen en meer complexe parameters, inclusief communicatie overhead, zijn Monte Carlo simulaties gebruikt. ‘Late binding’ leidt onder bepaalde omstandigheden tot een snelheidsverbeteringen van een grootte-orde of meer ten opzichte van ‘early binding’.

We beschrijven begeleidende grondslagen bij de ontwikkeling van lichtgewicht, ‘user-level overlay’ die ‘late binding’ toepassen om een verbeterde quality of service te bereiken in onbetrouwbare en onvoorspelbare gedistribueerde omgevingen. Onze strategie is gebaseerd op ‘loosely-coupled user-space’ gereedschappen, waar de Diane ‘scheduler’ de taak allocatie verzorgt binnen een verzameling van ‘worker nodes’ die asynchroon gecreëerd en onderhouden wordt door de Ganga interface. Deze benadering maakt het eenvoudig (1) om resource-selectie mechanismen te creëren zoals de op heuristisch gebaseerde ‘worker agent factory’ en (2) om op plugin-basis gebruik te maken van adaptieve ‘workload balancing’ algoritmen voor het plannen van taken. Andere belangrijke functies zijn de mogelijkheid om naar vele verschillende gedistribueerde systemen te interfacen; de mogelijkheid om het systeem uit te breiden met applicatie-specifieke scheduling en processing methodieken; eenvoud van gebruik en een uniform interface naar heterogene job management systemen.

We laten nieuwe en verbeterde mogelijkheden zien van de op een generieke middleware stack gepositioneerde Ganga Diane User-level Overlay door gebruik te maken van realistische applicaties in het EGEE Grid, lokale batch-systemen en dedicated clusters. Deze mogelijkheden zijn onder andere efficiënte ‘short-deadline’ berekeningen, verbeterde betrouwbaarheid, autonome grootschalige operaties, efficiënte parameter-sweeps, mens-in-de-loop scenario’s, geautomatiseerde DAG’s/workflows en semi-interactiviteit.

We rapporteren twee -studies aangaande ‘capacity’ en ‘capability’ computing met de ‘user-level overlay’. We demonstreren hoe, voor het verbeteren van de betrouwbaarheid van lokale resources voor de International Telecommunication Union Regional Radio Conference 2006, een groot aantal taken met een korte deadline gecoördineerd werd op het Grid. Vervolgens beschrijven we hoe ‘task prioritization’ en resource selectie geïmplementeerd werd voor de Lattice QCD simulaties die voortkwamen uit QCD thermodynamica onderzoek in het kader van heavy-ion collisions experimenten (LHC, RHIC).

Dit werk levert een bijdrage aan het debat over de mogelijkheid van het al dan niet efficiënt toepassen van Quality of Service binnen grids op applicatie niveau.

We laten zien dat dit wel degelijk mogelijk is door een theoretische verklaring van de effecten van ‘late binding’ op belangrijke taak processing metriecken te leveren, en door voorbeelden van applicaties te laten zien die op succesvolle wijze gebruik maakten van ‘user-level overlay’.

Streszczenie po polsku

Społeczności naukowe używają coraz większej liczby systemów obliczeniowych: od lokalnych systemów typu batch, specjalizowanych serwisów obliczeniowych i superkomputerów aż po globalnie rozproszone infrastruktury typu gridowego. Głównym zadaniem globalnych infrastruktur obliczeniowych jest zwiększanie potencjału badawczego w nauce poprzez udostępnianie na wielką skalę zasobów obliczeniowych oraz przechowywania danych. Cechą charakterystyczną zdecentralizowanych systemów gridowych jest chaotyczność i niejednorodność; jest to środowisko, w którym występują trudne do przewidzenia obciążenia zasobów i częste awarie podsystemów. Zrozumienie i opanowanie niejednorodności oraz dynamiki tego środowiska jest praktycznie niewykonalne przez pojedynczego użytkownika, jeżeli nie jest on wyposażony w odpowiednie metody i narzędzia. Czas niezbędny na naukę różnorodnych interfejsów użytkownika wielu środowisk obliczeniowych stanowi kolejne wyzwanie.

Skrócenie czasu wykonania zadań obliczeniowych oraz łatwiejsze jego przewidywanie jest jednym z najważniejszych czynników zwiększających efektywność pracy użytkowników. Technika późnego wiązania zasobów (late binding) pozwala osiągnąć te cele i może zostać wykorzystana, ponieważ większość wykonywanych zadań obliczeniowych w systemach gridowych jest "giętka" (moldable). Giętkość oznacza, że zadanie może zostać efektywnie wykonane na zmiennej liczbie zasobów obliczeniowych (procesorów) gdyż może ono zostać elastycznie podzielone na dowolną liczbę podzadań (w przeciwieństwie do typowych zadań równoległych). Giętkość charakteryzuje zadania z wielu dziedzin zastosowań takich jak symulacje Monte Carlo, dobór optymalny parametrów (parameter sweep), zadania typu grafowego (DAG), analiza danych itp.

W tej pracy jest analizowana przestrzenna i czasowa dynamika globalnej infrastruktury gridowej – EGEE Grid. W celu analizy zmienności wydajności systemu gridowego został opracowany model przetwarzania zadań gridowych, w którym parametry systemu gridowego reprezentowane są jako zmienne losowe o rozkładach empirycznych. W oparciu o ten model została przeprowadzona analiza metryk jakości usług gridowych związanych z przetwarzaniem zadań takich jak wariancja rozpiętości czasu wykonania

zadań (variance of makespan). Analiza ta umożliwia porównanie metody późnego i wczesnego wiązania zasobów. Stosując przybliżenie ciągle modelu dyskretnego i fundamentalne prawa statystyki, zostało wykazane, że właściwości metody późnego wiązania zasobów pozwalają na poprawę metryk jakości usług systemu gridowego. Do analizy dyskretnego modelu przetwarzania zadań została zastosowana symulacja metodą Monte Carlo, co pozwala na uwzględnienie dodatkowych parametrów takich jak narzuty komunikacyjne. Przeprowadzona analiza pozwala stwierdzić, że w pewnych przypadkach metoda późnego wiązania zasobów pozwala na skrócenie czasu wykonania zadań o rząd wielkości w porównaniu do metody wczesnego wiązania.

W tej pracy zostały opisane też zasady tworzenia lekkiej nakładki użytkownika (User-level Overlay), która wykorzystuje późne wiązanie zasobów w celu poprawy jakości przetwarzania zadań w systemach obciążonych dużą niepewnością i nieprzewidywalnością. Opracowana w tej pracy strategia, bazuje na luźno zintegrowanych narzędziach dostępnych bezpośrednio dla użytkownika: DIANE jest silnikiem zarządzania zadaniami przetwarzanymi przez zbiór węzłów obliczeniowych zarządzanych za pomocą interfejsu Ganga. Takie podejście umożliwia łatwe tworzenie algorytmów wyboru zasobów (na przykład za pomocą mechanizmów heurystycznych) oraz łatwe tworzenie algorytmów zarządzania zadaniami (na przykład za pomocą metod adaptacyjnych i równoważenia obciążenia zasobów). Pozostałe kluczowe cechy tego systemu to dostęp do szerokiej gamy rozproszonych systemów obliczeniowych, możliwość rozszerzenia i adaptacji systemu do specyficznych wymagań konkretnych aplikacji, łatwość użycia i jednorodny interfejs ułatwiający korzystanie z niejednorodnych środowisk obliczeniowych.

Nakładka User-level Overlay umożliwia osiągnięcie nowych zdolności zarządzania zadaniami w stosunku do klasycznych systemów obliczeniowych. Zdolności te zostały zaprezentowane na przykładzie istniejących aplikacji w środowisku gridowym i obejmują: zwiększenie wydajności obliczeń o krótkim terminie wykonania (short-deadline), zwiększoną niezawodność i automatyzację wykonywania zadań, wsparcie dla algorytmów przeszukiwania przestrzeni parametrów (parameter sweep), automatyzacja zadań typu grafowego i workflow oraz zdolność pracy pół-interaktywnej.

Zaprezentowane zostały dwa przypadki realizacji aplikacji typu Capability Computing i Capacity Computing z wykorzystaniem nakładki User-level Overlay. Pierwszy przypadek obejmuje koordynację dużej liczby zadań o bardzo krótkim czasie wykonania dla obliczeń związanych z planowaniem nowych standardów nadawania cyfrowego w ramach konferencji RRC06 Międzynarodowej Unii Telekomunikacyjnej (ITU). Drugi przypadek obejmuje priorytezację zadań oraz selekcję zasobów obliczeniowych dla symulacji Lattice QCD w fizyce teoretycznej dla potrzeb eksperymentów na ciężkich jonach (LHC, RHIC).

Niniejsza praca stanowi wkład do badań nad sposobami zwiększenia jakości usług związanych z przetwarzaniem zadań w systemach gridowych na poziomie użytkownika i aplikacji. Zostało zademonstrowane, iż istotnie jest to możliwe poprzez wyjaśnienie podstaw teoretycznych wpływu metody późnego wiązania zasobów oraz wdrożenie z sukcesem nakładki User-level Overlay w wielu praktycznych zastosowaniach naukowych.

Publications in peer-reviewed journals

1. J. Mościcki, M. Lamanna, M. Bubak, and P. Sloot. Processing moldable tasks on the Grid: late job binding with lightweight User-level Overlay. (*accepted for publication*) in *Future Generation Computer Systems*, 2011
2. J. T. Mościcki, M. Wos, M. Lamanna, P. de Forcrand, and O. Philipsen. Lattice QCD thermodynamics on the Grid. *Computer Physics Communications*, 181(10):1715 – 1726, 2010
3. S. C. Pop, T. Glatard, J. Mościcki, H. Benoit-Cattin, and D. Sarrut. Dynamic partitioning of GATE Monte-Carlo simulations on EGEE. *J. Grid Computing*, 8(2):241–259, 2010
4. J. Mościcki, F. Brochu, J. Ebke, U. Egede, J. Elmsheuser, K. Harrison, R. Jones, H. Lee, D. Liko, A. Maier, A. Muraru, G. Patrick, K. Pajchel, W. Reece, B. Samset, M. Slater, A. Soroko, C. Tan, D. van der Ster, and M. Williams. Ganga: A tool for computational-task management and easy access to Grid resources. *Computer Physics Communications*, 180(11):2303 – 2316, 2009,
5. V. Korkhov, J. Mościcki, and V. Krzhizhanovskaya. The user-level scheduling of divisible load parallel applications with resource selection and adaptive workload balancing on the Grid. *IEEE Systems Journal*, 3:121–130, March 2009
6. V. Korkhov, J. T. Mościcki, and V. Krzhizhanovskaya. Dynamic workload balancing of parallel applications with user-level scheduling on the Grid. *Future Generation Computer Systems*, 25(1):28 – 34, 2009,
7. C. Germain-Renaud, C. Loomis, J. Mościcki, and R. Texier. Scheduling for responsive Grids. *J. Grid Computing*, 6:15–27, 2008

Conference papers published in journals

1. A. Maier, F. Brochu, G. Cowan, U. Egede, J. Elmsheuser, B. Gaidioz, K. Harrison, H.-C. Lee, D. Liko, J. Mościcki, A. Muraru, K. Pajchel, W. Reece, B. Samset, M. Slater, A. Soroko, D. van der Ster, M. Williams, and C. L. Tan. User analysis of LHCb data with Ganga. *Journal of Physics: Conference Series*, 219(7):072008, 2010
2. J. Elmsheuser, F. Brochu, U. Egede, B. Gaidioz, K. Harrison, H. Lee, D. Liko, A. Maier, J. Mościcki, A. Muraru, V. Romanovsky, A. Soroko, and C. Tan. Distributed analysis using Ganga on the EGEE/LCG infrastructure. *Journal of Physics: Conference Series*, 119(7):072014 (8pp), 2008
3. J. Mościcki. Distributed analysis environment for HEP and interdisciplinary applications. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 502(2-3):426 – 429, 2003. Proceedings of the VIII International Workshop on Advanced Computing and Analysis Techniques in Physics Research
4. J. Mościcki, S. Guatelli, A. Mantero, and M. Pia. Distributed Geant4 simulation in medical and space science applications using DIANE framework and the Grid. *Nuclear Physics B - Proceedings Supplements*, 125:327 – 331, 2003

Conference proceedings

1. J. Mościcki, M. Bubak, H. Lee, A. Muraru, and P. Sloot. Quality of service on the grid with user level scheduling. In M. Bubak, M. Turala, and K. Wiatr, editors, *CGW'06 Proceedings*, pages 119–129. 2007 (best poster award)
2. S. Chauvie, P. Lorenzo, A. Lechner, J. Mościcki, and M. Pia. Benchmark of medical dosimetry simulation using the Grid. In *IEEE Nuclear Science Symposium Conference Record NSS '07*, volume 2, pages 1100–1106, 2007
3. S. Guatelli, A. Mantero, P. Mendez-Lorenzo, J. Mościcki, and M. Pia. Geant4 simulation in a distributed computing environment. In *IEEE Nuclear Science Symposium Conference Record, 2006*, volume 1, pages 110–113, 2006
4. R. Mendez-Lorenzo, J. Mościcki, and A. Ribon. Experiences in the gridification of the Geant4 toolkit in the WLCG/EGEE environment. In *IEEE Nuclear Science Symposium Conference Record*, volume 2, pages 879–884, 2006
5. J. Mościcki, H. Lee, S. Guatelli, S. Lin, and M. Pia. Biomedical applications on the Grid: efficient management of parallel jobs. In *IEEE Nuclear Science Symposium Conference Record, 2004*, volume 4, pages 2143 – 2147, 2004
6. M. Gallas, J. Mościcki, M. Lamanna, and L. Mancera. Quality assurance and testing in LCG. In CERN, editor, *Computing for High Energy Physics, 2004*. Interlaken (Switzerland), September 2004

7. R. Chytracek, D. Dullmann, M. Frank, M. Girone, G. Govi, J. Mościcki, I. Papadopoulos, H. Schmuecker, K. Karr, D. Malon, A. Vaniachine, W. Tanenbaum, Z. Xie, T. Barrass, and C. Cioffi. LCG POOL development status and production experience. In *IEEE Nuclear Science Symposium Conference Record*, volume 4, pages 2077–2081 Vol. 4, 2004
8. F. Foppiano, S. Guatelli, J. Mościcki, and M. Pia. From DICOM to Grid: a dosimetric system for brachytherapy born from HEP. In *IEEE Nuclear Science Symposium Conference Record*, volume 3, pages 1746–1750 Vol.3, 2003
9. J. Mościcki. DIANE - distributed analysis environment for grid-enabled simulation and analysis of physics data. In *IEEE Nuclear Science Symposium Conference Record*, volume 3, pages 1617–1620 Vol.3, 2003
10. M. Bubak, J. Mościcki, and J. Shiers. Design of high-performance C++ package for handling of multidimensional histograms. In P. Sloot, M. Bubak, A. Hoekstra, and B. Hertzberger, editors, *High-Performance Computing and Networking*, volume 1593 of *Lecture Notes in Computer Science*, pages 543–552. Springer Berlin / Heidelberg, 1999. 10.1007/BFb0100615

Newsletters and reports

1. J. Mościcki. The DIANE user-scheduler provides quality of service. *CERN Computer Newsletter*, 9 2006
2. O. Couet, D. Ferrero-Merlino, Z. Molnar, J. Mościcki, A. Pfeiffer, and M. Sang. Anaphe - OO libraries and tools for data analysis. Technical Report CERN-IT-2001-012, CERN, Geneva, Sep 2001

Acknowledgments

First and foremost, I would like to thank my promoter Marian Bubak for all his generous help, advice and persistent confidence he has given to me, and without whom this work would never be completed. I would like to thank my co-promoter Peter Sloom for inspiring discussions and invaluable support. Very special thanks go to Massimo Lamanna (CERN) for his firm support and involvement in many software and application projects described in this work, and for his enthusiastic interest in my scientific developments.

I am very grateful to the committee members for their role and involvement in the final stages of this work: Hamideh Afsarmanesh, Alfons G. Hoekstra, Juergen Knobloch, Cees Th.A.M. de Laat and Krzysztof Zieliński.

This work would not be possible without support of my line management at CERN. I would like to thank Frederic Hemmer, CERN-IT Department Head for his encouragement to complete this work, Juergen Knobloch, CERN-IT-PSS Group Leader who accepted the idea and gave it a green light in his group and Massimo Lamanna – the distributed data analysis section leader.

This work builds upon collaboration with many great scientists and engineers. I would like to thank the entire GANGA team for great collaboration we have had together over the years and their contributions to the GANGA paper. Special thanks go to Ulrik Egede (Imperial College, London) for his great attitude as a project leader. Pere Mato (CERN) deserves a special mention here as a spiritual father of GANGA. Likewise, many people were involved in the DIANE project, which started as an R&D activity for distributed ntuple analysis in the team managed by Andreas Pfeiffer and Dino Ferrero Merlino (CERN). Maria Grazia Pia (INFN Genova) deserves special thanks for her support in the early stages of the project. She also provided a great link to the Geant 4 world which enabled successful collaboration for low energy applications with Susanna Guatelli (ANSTO, Australia), Anton Lechner (CERN) and others.

I have also shared much fun collaborating with Patricia Mendez Lorenzo (CERN), Hurng-Chun Lee (NIKHEF, Amsterdam), Andrea Manara (ITU Geneva), Philippe de

Forcrand (CERN and ETH Zurich), Maciej Woś (University of Cambridge), Vladimir Korkhov (University of Amsterdam), Valeria Krzhizhanovskaya (University of Amsterdam), Cecile Germain-Renaud (Université Paris-Sud), Tristan Glatard and Sorina Camarasu (Creatis, Lyon), Shantenu Jha (Louisiana State University) and Ole Weidner (Louisiana State University) on software and application projects and research papers. Many thanks to Bernd Panzer-Steindel (CERN) for his support in accessing computing resources in the CERN Computing Center. Thanks to Max Sang for inventing the name of the DIANE project and much fun we had in the early days.

My frequent visits to Amsterdam owe a lot to the hospitality and friendship of Harmen, Kim, Otwin and Gabrielle with whom I shared so many good moments and whose houses were always open for me.

None of that would ever been possible without my Parents, Halina Juda-Mościcka and Włodzimierz Jerzy Mościcki, whom I owe the general education and passion for science. They have been always supportive and encouraging. Thank you for everything.

My little family has been very understanding and patient while I was working long weekends and evenings on this thesis. Without gentle support of my wife, Ania Kondziela, and an energy kick and inspiration from my little daughter Marysia and recently born son Adam Ignacy, it would have been close to impossible to complete this work. And there was often Rydzyk joining me in this solitaire effort and finding a perfect application for the heat generated by computer hardware: a cat's perfect nap.

Index

- agent factory, [64](#), [81](#), [87](#), [126](#), [130](#), [137](#),
[138](#), [140](#)
 - heuristic, [82](#), [84](#)
 - hybrid, [85](#), [87](#)
 - simple, [81](#), [82](#)
- Amdhal's Law, [7](#)
- ARC , [64](#)
- Athena analysis framework, [102](#)
- ATLAS, [27](#), [102](#), [103](#), [146](#)
 - VO, [44](#), [46](#), [48](#)
- BLAST (Basic Local Alignment Tool), [10](#),
[15](#), [106](#)
- Capability Computing, *see* HPC
- Capacity Computing, *see* HTC
- Central Limit Theorem, [50](#)
- CERN, [4](#), [8](#), [20](#), [23](#), [46](#), [55](#), [102](#), [114](#), [122](#)
- CMS
 - VO, [46](#)
- Condor, [6](#), [9–11](#), [34](#), [64](#)
 - M/W, [10](#), [11](#)
- connection management, [71](#)
- CORBA, [69](#), [80](#)
- CREAM, [8](#), [34](#)
- degree of parallelism, [3](#)
- DEISA, [9](#)
- DIANE framework, [48](#), [55](#), [59](#), [63–68](#), [71–](#)
[73](#), [76](#), [77](#), [80](#), [82](#), [84](#), [93–95](#), [97](#),
[99–106](#), [113](#), [115](#), [121](#), [125](#), [126](#),
[128](#), [129](#), [140](#), [145](#), [146](#), [150](#), [157](#),
[158](#)
- DIRAC workload management system, [45](#),
[102](#)
- divisible load, [42](#), [50](#), [86](#)
- early binding, [10](#), [11](#), [16](#), [28](#), [43](#), [56](#), [92](#), [93](#),
[101](#), [104](#)
- EGEE Grid, [80](#)
- evolving jobs, [2](#)
- FORTTRAN, [7](#), [130](#)
- Ganga interface, [59](#), [63–66](#), [73–82](#), [84](#), [93](#),
[95](#), [99–107](#), [113](#), [115](#), [121](#), [126](#),
[140](#), [145](#), [146](#), [150](#), [157](#)
- Gaudi analysis framework, [102](#)
- Geant4, [91](#), [97](#)
 - VO, [31](#), [44](#), [46](#), [48](#), [93](#)
- glxexec, [35](#)
- gLite workload management system, [67](#), [76](#)
- Globus, [64](#)
- Google
 - Summer of code, [106](#)
- GSI, [72](#), [76](#)
- Gustafson's Law, [7](#)
- High Performance Computing, [5](#), [9](#), [10](#), [125](#),
[141](#)
- High Throughput Computing, [5](#), [8](#), [43](#)
- HPC, *see* High Performance Computing

- HTC, *see* High Throughput Computing
- in silico experiments, 104
- IPython interface, 73
- job queuing time, 37, 43–46, 48, 53, 144
 dispersion coefficient, 52, 54
- job submission, 33, 41, 56, 64, 76, 83, 97,
 102, 104, 117
- Kerberos, 76, 80
- late binding, 10, 11, 16, 41, 43, 48, 53, 57,
 61, 62, 89, 101, 105, 121, 144
- layering, 67
- LHC project, 102
- LHCb, 102, 103, 146
 VO, 44, 46, 48
- LSF, 9, 64, 81
- makespan, 14, 43, 50, 54, 98, 102, 116, 144
 bound of, 49
 distribution, 42, 48
- malleable jobs, 2
- Many Task Computing (MTC), 5
- Massively Parallel Processors (MPP), 9
- moldable jobs, 2
- Moldable Task Applications, 2, 7
- moldable workload, 42
- Monte Carlo simulation, 3, 4, 6, 14, 42, 55,
 91, 95, 121, 123, 125, 127, 132
- monte Carlo simulation, 48
- MPI, 7, 10, 13, 15, 66, 76, 92, 139, 140, 146
- MTA
 see Moldable Task Applications, 2
- MTC (Many Task Computing), 5
- MyProxy, 35
- non-deterministic selection, 82
- NWS, 11
- omniORB, 66, 71
- OpenMP, 7, 66, 139, 140, 146
- PANDA workload management system, 11,
 27, 102, 103
- PBS, 9, 64
 pilot jobs, 10
 placeholder scheduling, 10
 prioritization, 60, 130
 Python programming language, 61, 62, 66–
 69, 73, 75, 83, 96, 107, 125
- Quality of Service, 9, 12, 13, 16, 41, 50, 53,
 57, 62, 144
 metrics, 48, 53
- resource discovery, 83
- resource heterogeneity metric, 89
- resource selection, 62–64, 80, 82, 85
- resubmission, 6, 14, 37
 deep, 28
 shallow, 28
- rigid jobs, 2
- SAGA, 12, 64, 140
- scheduling, 10, 14, 53, 60, 68, 72, 89, 93,
 117, 130
- security, 5, 11, 12, 72, 76
- SGE, 64
- software bus, 67
- speedup, 7, 53, 57, 81, 85, 88, 89, 97, 104,
 110, 140
- SSL, 72
- Symmetric Multi-Processors (SMP), 9
- task paging, 54
- taxonomy of parallel jobs, 2
- Tera Grid, 9, 139, 140
- VOMS, 35
- WISDOM, 104, 105
- WLCG, 8, 82
- WMS, *see* workload management system
- workload balancing, 85–89, 117, 120, 144
 effects, 54
- workload management system, 12, 21, 27,
 35, 44, 81–83