UNIVERSITY OF AMSTERDAM

UvA-DARE (Digital Academic Repository)

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

Mościcki, J.T.

**Publication date**
2011

Link to publication

**Citation for published version (APA):**
Mościcki, J. T. (2011). *Understanding and mastering dynamics in computing grids: processing moldable tasks with user-level overlay*.

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

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

*UvA-DARE is a service provided by the library of the University of Amsterdam (https://dare.uva.nl)*

Download date:27 Sep 2021

# 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 grid[1] 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

[1] We 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 begining 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 then 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 stategies 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 normalized[2] 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

[2] Reliability 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 scheduling[3] and prioritization[4] 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 (Amdhal'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

[3] Scheduling is a problem of mapping of a set of activities onto a set of limited resources over time.
[4] Prioritization 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 operation[5]. 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, EGEE[6] or projects such as NorduGrid/NDGF[7] 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) levels[8]. 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 OSG[9],

[5] A 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

[8] Virtual 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 TeraGrid[10], 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. Adavanced scheduling policies are not readily available for the EGEE Grid users, despite the fact that internally gLite WMS uses Condor-based components. One unanswered issues 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 metadata 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-continous 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 sucessfully 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 fulfulled 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-QoSm [10] system aimed at achieving similar goals using Open Grid Services Architecture (OGSA). Despite being a promising solution, neither GARA nor G-QoSm have not been widely adopted for practical use (in the latter case the OGSA standard became obsolete before the G-QoSm was able to make impact). G-RSVPM [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 expertize. 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.