Chapter 9

QosCosGrid e-Science Infrastructure for Large-Scale Complex System Simulations

Krzysztof Kurowski, Bartosz Bosak, Piotr Grabowski, Mariusz Mamonski, and Tomasz Piontek
Poznan Supercomputing and Networking Center, Poznan, Poland

George Kampis
Collegium Budapest (Institute for Advanced Study), Budapest, Hungary

László Gulyás
Aitia International Inc. and Collegium Budapest (Institute for Advanced Study), Budapest, Hungary

Camille Coti
LIPN, CNRS-UMR7030, Université Paris 13, Villetaneuse, France

Thomas Herault and Franck Cappello
National Institute for Research in Computer Science and Control (INRIA), Rennes, France

9.1 INTRODUCTION

Grids and clouds could be viewed as large-scale computing systems with considerable levels of hardware resources but lacking many of the the features that make supercomputers so powerful. In particular, grids and clouds usually

Large-Scale Computing, First Edition. Edited by Werner Dubitzky, Krzysztof Kurowski, Bernhard Schott.

do not provide sophisticated support for parallel and multiphysics applications with significant interprocess communication requirements. Connected via local and wide area networks, such big computing infrastructures typically rely on an opportunistic marshalling of resources into coordinated action to meet the needs of large-scale computing applications. Both grids and clouds are often presented as a panacea for all kinds of computing applications, including those that require supercomputing-like environments. However, this vision of grids or clouds as virtual supercomputers is unattainable without overcoming their performance, coallocation, and reliability issues. The demanding nature of scientific simulations requires a new e-infrastructure that is able to simultaneously manage heterogeneous resources, such as computing resources, storage, and networks to guarantee the level of quality of service demanded by end users and their applications, especially a large number of legacy applications designed to run in parallel. To meet the requirements of large-scale complex simulations, we have built a system capable of bringing supercomputer-like performance to advanced applications, including sophisticated parameter sweep experiments, workflow-intensive applications, and cross-cluster parallel computations. Our system (called QosCosGrid [QCG]) consists of a collection of middleware services and external application tools and was first introduced by Kurowski et al. (2009). QCG is designed as a multilayered architecture that is capable of dealing with computationally intensive large-scale, complex, and parallel simulations that are usually too complex to run within a single computer cluster or machine. The QCG middleware enables computing resources (at the level of processor cores) from different administrative domains (ADs) to be combined into a single powerful computing resource via the Internet. Clearly, bandwidth and latency characteristics of the Internet may have an effect on overall application performance of QCG-enabled applications. However, the ability to connect and efficiently control advanced applications executed in parallel over the Internet is a feature that is highly appreciated by QCG users.

QCG could be viewed as a *quasi-opportunistic supercomputer* whose computational performance exceeds the power offered by a single supercomputing or data center (Kurowski et al., 2010). Nowadays, it is more common for complex system simulations to rely on supercomputers because of the high data volume and computing requirements of the individual computations, but also because of the high communication overhead between the computation tasks on individual elements. However, dedicated supercomputers for such calculations are expensive to acquire and maintain. As a consequence, many organizations do not have access to supercomputing facilities and rely local computing resources. Recently, local computing clusters and other multicore and multimachine systems have become the technology of choice for many complex system modelers. However, with the advent of flexible modeling tools, complex system simulations have become even more difficult to manage. As a result, local clusters are increasingly inadequate to satisfy the required computing and communication needs. QCG aims to address this gap by facilitating supercomputer-like performance and structure through efficient cross-cluster

computations. Thus, new middleware services and application tools for end users were developed and integrated to narrow the gap and to realize large-scale parallel and distributed computing e-infrastructures. The QCG implementation comprises a comprehensive framework for metascheduling and managing topology-aware complex system applications. This framework includes pluggable components that carry out the usual scheduling operations, including the assignment of parallel processes on a time axis, clustering of resources, and matching application requests[1] and available resources across geographic dispersed locations. The QCG framework is highly flexible as it is composed of pluggable components that can be easily modified to support different scheduling and access policies to better maximize a diversity of utility functions. Furthermore, the framework exploits novel algorithms for topology-aware coallocations that are required by parallel programming and execution setups in production-level high-performance computing (HPC) environments, such as the Message Passing Interface (MPIs), ProActive, or their hybrid extensions linking programming models like OpenMP or CUDA.

This chapter focuses on two enhanced and widely used parallel computing environments: QosCosGrid-ProActive (QCG-ProActive) and QosCosGrid-OpenMPI (QCG-OMPI). QCG-OMPI[2] is designed to enable parallel applications to run across geographically widely distributed computational resources owned and secured by different administrative organizations (Agullo et al., 2011). QCG-ProActive was successfully integrated with the Repast Suite, a very popular Java-based, agent-based modeling and simulation platform (Gulyás et al., 2008). A new version of the C++ implementation of Repast for supercomputing environments (called Repast HPC[3]) will be supported by QCG.

The remainder of this chapter is organized as follows. Section 9.2 presents a short overview of validation scenarios; it also discusses the main requirements of complex systems and other demanding parallel applications and classifies these into "templates." Section 9.2 gives a technical overview of QCG-ProActive and QCG-OMPI. In Section 9.4, we introduce the QCG middleware services and their key capabilities relevant to end users. A number of useful libraries and frameworks for application developers and resource providers using QCG are described in Section 9.5. Section 9.6 presents additional Web-based monitoring and troubleshooting tools for QCG, which are already available on some production-level computational clusters and supercomputers. Finally, Section 9.7 concludes this chapter and discusses future development plans.

9.2 DISTRIBUTED AND PARALLEL SIMULATIONS

In this section, we briefly introduce the main requirements of a variety of complex system applications. To classify frequently occurring application

[1] We have categorized these into six generic templates; see Section 9.2.
[2] More details on the QCG-OMPI middleware are presented in Chapter 8 of this volume.
[3] Chapter 5 presents an overview of the Repast HPC framework and its implementation.

patterns into communication templates, Gulyás et al. (2008) studied the interaction topologies of a wide range of complex system simulations. The proposed communication templates were used to inform the design and implementation of the QCG middleware. The QCG middleware controls a hierarchical structure of scheduler and map application requests to distributed computational resources and networks to enable efficient processing. Each template may be accompanied by one or more template implementations and parameters that determine how the information in the templates is to be used by the QCG middleware. This feature is specifically designed to meet the needs of complex system modelers. The modelers take advantage of this by identifying the communication class their models belong to and then populate the templates with the details of their specific simulation models and tasks. While this approach may not achieve the most efficient distributed implementation, it is likely to reduce the implementation effort that would otherwise be required to realize many simulations.

Two special application patterns may be distinguished. In the first, the dependencies among components do not follow a predetermined structure (i.e., they form a uniform random distribution) and change regularly over time (e.g., dependencies are resampled prior to each update). The second special case is when no communication occurs among complex system components.

One might argue that these are overly simplified examples and that a collection of components (as in the second example) may not qualify as a system at all. However, we believe this is merely a question of the level of abstraction adopted. In the first scenario, which is not as uncommon as it may seem, a distributed parameter space search is arguably the most adequate implementation strategy, which is just an example of our second scenario: the individual simulation runs can be viewed as noninteracting components. On the other hand, more sophisticated parameter space search methods introduce dependencies among individual runs by determining which parameter combination to explore next on the basis of the results computed previously (i.e., sampling in more "turbulent" parameter regions). In this case, a parameter space search becomes a nontrivial complex system again, worthy of dependency analysis in its own right. Dealing with more complex cases, our first observation is that static communication patterns allow for the direct application of distribution algorithms. Therefore, we handle these cases separately from the dynamic topologies. Next, we point out that the dependencies of the update functions may be dependent on the components' states. In many cases, it is possible to project the components' state information to a metric space and update dependencies based on distance in this space. For example, if components are agents moving in a space (in computational models often realized as a two-dimensional lattice), then each agent state will (among other things) include the coordinates of the agent. If in the model the agents interact only with the agents in their vicinity, then the update dependencies of the components are distance dependent. This spatial property of a complex system simulation, if present, may be successfully exploited in determining the partitions of a distributed

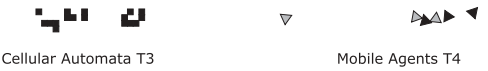

Figure 9.1 *QosCosGrid communication templates that help users to understand and classify main complex system requirements for large-scale simulations.*

implementation. It is worth noting, however, that complex systems may have such a spatial property implicitly. For example, a social system where people are likely to interact with like-minded partners may have this property where the natural metric space is an abstract similarity space. Based on the observations above, we propose five plus one *communication templates* that, as we believe, represent commonly occurring classes of complex system simulations (Fig. 9.1).

Clearly, it is possible to identify many more templates or to refine the presented classification scheme by introducing subclasses. However, we believe the scheme of six main template categories (Fig. 9.1) is sufficient to realize a wide range of complex system simulations and to implement a flexible approach to support efficient distributed computing strategies.

Template 0 (T0) of our classification is the case where no interaction occurs among components. Here, the component partitioning is only constrained by load balancing considerations.

Template 5 (T5), the other extreme, is the case with random or unpredictable interaction among components. In general, we are not able to predict any communication topology in advance so we recommend to run such components independently. The four remaining cases are created at the intersection of the spatial/nonspatial and static/dynamic properties.

Template 1 (T1) (static networks) describes a nonspatial system with a static communication pattern. It is assumed that the exact communication pattern can be extracted from the system, or that it is defined explicitly. To the thus defined communication graph, a variety of graph partitioning algorithms can be applied, in particular, approaches proposed by Fjallstrom (1998).

Template 2 (T2) (dynamic networks) introduces dynamism into Template 1. The assumption about the existence of a communication graph is maintained, but, in contrast to Template 5, it is assumed that the changes and their frequency are defined by a graph transition function that provides enough information to predict in advance communication requirements among distributed components. (Sometimes, it may be sufficient to know that the level of change in the communication graph is low, such that it is sufficient to repartition nodes at regular intervals, at every 10,000 time steps, for instance.) The implementation approach we propose for such systems is the regular application of classic graph repartitioning algorithms, that is, graph partitioning algorithms that attempt to improve on an existing partition (Barnard, 1995).

Template 3 (T3) (static spatial systems) moves away from Template 1 along the other axis. It maintains the assumption about a static communication pattern but requires the spatial property. Prime examples of such systems are cellular automata. Template 3 is the pattern that is geared toward a distributed implementation; such implementations may be based on methods and algorithms developed for distributed cellular automata (Mazzariol et al., 2000).

Finally, *Template 4* (T4) (dynamic spatial systems) assumes a spatial system in which the communication pattern evolves over time. One example of such a system is the case discussed earlier in which agents move in space and communicate with other agents in their immediate vicinity. To realize distribution computing strategies for complex systems belonging to this template, we point to algorithms specially developed for such systems using buffering and messaging solutions and ways of predicting the speed of spatial movement (Scheutz and Schermerhorn, 2006). It is worth pointing out that the fundamental assumption of this template and thus a key to the successful implementation of these solutions is that changes in spatial positions are slow relative to the frequency of state updates.

9.3 PROGRAMMING AND EXECUTION ENVIRONMENTS

The main goal in the development of the QCG middleware was to provide a flexible, efficient, and secure distributed computing system that is able to deal with large-scale simulations over distributed computing resources connected via local and wide area networks (in particular via Internet connections). To inform the development of the QCG middleware and to facilitate its testing and validation, a number of concrete resource-demanding complex system simulations have been identified and classified into communication templates (Section 9.2). The initial set of use cases included the living simulation, evolu-

tionary computation, and agent-based modeling. Recently, we added new requirements arising from various multiscale and multiphysics use cases to be support by QCG, for example, the multiscale modeling in computational biomedicine (Sloot and Hoekstra, 2010).

From a development perspective, the applications were grouped into two classes: (1) Java applications taking advantage of the ProActive library as the parallelization technology and (2) applications based on ANSI C or similar codes, which rely on the message passing paradigm. Based on these groups, QCG was designed to support two parallel programming and execution environments, namely, QCG-OpenMPI (aiming at C/C++ and Fortran parallel applications developers) and QCG-ProActive (aiming at Java parallel application developers).

In this section, we briefly describe those two programming and execution environments together with a number of useful features that make them easy to use and powerful for end users concerned with large-scale parallel simulations of complex systems. All the presented technologies and integration efforts described in the next two subsections were needed to faciliate cross-cluster execution of advanced applications in firewall-protected and Network Address Translation (NAT) environments. Next, we also present our new approach to a common problem in many grids and clouds regarding the simultaneous access and synchronization of a large number of computational resources. We also provide an overview of the innovative cross-cluster deployment protocol developed within QCG, which is designed to simplify the management of complex parallel processes that are organized in groups and hierarchies. Even though the QCG-managed cross-cluster parallel executions are limited to the two enhanced parallel environments, other advanced management capabilities for applications are supported as well, including support for workflows or parameter sweep jobs.

9.3.1 QCG-OMPI

MPI is a de facto a standard in the domain of parallel applications. MPI provides end users with both the programming interface consisting of simple communication primitives and the environment for spawning and monitoring MPI processes. A variety of implementations of the MPI standard are available (both as commercial and open source). QCG uses the OpenMPI implementation of the MPI 2.0 standard. Of key importance are the intercluster communication techniques that deal with firewalls and NAT. In addition, the mechanism for spawning new processes in OpenMPI is integrated in the QCG middleware. The extended version of the OpenMPI framework is referred to as QCG-OMPI (Coti et al., 2008). The QCG-OMPI is as follows:

1. Internally, QCG-OMPI improves the MPI library through a variety of connectivity techniques to enable direct connections between MPI ranks that are located in remote clusters, potentially separated by firewalls.

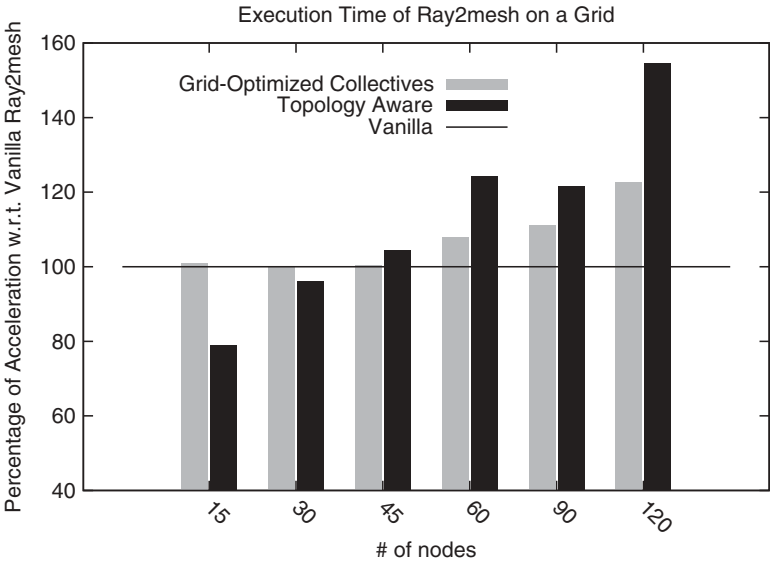

Figure 9.2 *QosCosGrid-OpenMPI (QCG-OMPI) performance on test-bed resources.*

2. Mechanisms going beyond the MPI standard were realized to accommodate QCG's semiopportunistic approach; this was achieved by a new interface to describe the actual topology provided by the Metascheduler.
3. MPI collective operations were upgraded to be hierarchy aware and were optimized for the grid.

We briefly present in this section each of these extensions and some results that demonstrate the performance achievable with QCG-OMPI.

Agullo et al. (2010) present two MPI applications that have been ported to the QCG approach using QCG-OMPI. The first is a simple ray tracing application (Ray2mesh) based on a hierarchical master–worker scheme. Results from performance tests are depicted in Figure 9.2 and are contrasted with the performance of non-QCG implementations of this application. Using grid-aware collective operations is a simple approach for porting an application to the grid. Ray2mesh uses collective operations to communicate parameters and collect partial results from the computing nodes. Figure 9.2 illustrates that the performance obtained through Ray2mesh using grid-optimized collectives (gray bars) is consistently outperforming the Vanilla implementation; the performance gain increases as the number of processes increase (20% with 120 processes).

Another key element of porting an application to the grid requires the adaptation of communication and computation patterns that fit the underlying topology (black bars). The performance improvement at a larger number of nodes or processors is outperforming topology-aware collective communica-

tions (55% with 120 processes). We can see that small-scale executions show lower performance than the Vanilla implementation. The reason for this is that for a given number of processes, the grid-enabled implementation dedicates more processes to control and scheduling (i.e., master processes) than the Vanilla implementation. As a consequence, fewer processes are available for computation (i.e., worker processes). However, it would be possible to run worker processes on the same nodes as master processes since the latter are carrying out input/output (I/O) operations, whereas the former is running CPU-intensive operations.

The *port range* technique establishes a direct connection between processes that communicate with each other (Agullo et al., 2010). As a consequence, this approach has no overhead on the performance of the communication library: The bandwidth and the latency of the communications are the same as the those obtained by the Vanilla communication libraries.

Interconnecting processes through a proxy does not enable a direct connection between them. It introduces an extra "hop" between the two processes; hence, the process–process latency is the result of adding the two process–proxy latencies (Agullo et al., 2010). As a consequence, the physical location of the proxy with respect to the processes is of major importance in order to minimize this additional overhead. Besides, the proxy's bandwidth is shared between all the processes that are using it. If several processes are communicating at the same time through a given proxy, the available bandwidth for each process will be divided between them. Finally, performance evaluations have been conducted on "raw communications," and the impact of the performance obtained by these techniques on benchmark applications has been compared by Coti et al. (2008).

More comprehensive studies of QCG-OMPI and another grid-enabled application are presented by Agullo et al. (2010). A linear algebra factorization has been adapted to the grid using a communication-avoiding algorithm with grid-aware domain decomposition and a reduction algorithm that confines the communications within sets of processes that match the underlying physical topology. The performance of this application shows good scalability on the grid. Chapter 8 of this volume presents a topology-aware evolutionary algorithm and its application to gene regulatory network modeling and simulation.

9.3.2 QCG-ProActive

The vast number of Java-based legacy applications in use prompted developments attempting to provide a similar functionality for parallel Java applications as MPI offers to C/C++ or Fortran parallel code. Instead of exploiting existing Java bridges to MPI implementations, we decided to use the ProActive Parallel suite (Baduel et al., 2006). The library uses the standard Java RMI framework as a portable communication layer. With a reduced set of simple primitives, ProActive (version 3.9 as used in QCG) provides a comprehensive

toolkit that simplifies the programming of applications distributed on local area networks, clusters, Internet grids, and peer-to-peer intranets for Java-based applications. However, when we designed QCG, the standard ProActive framework did not provide any support for multiuser environments, advance reservation, and cross-cluster coallocation. To meet the requirements of complex system simulation applications and users, we developed extensions to the ProActive library (called QCG-ProActive) with the following goals:

- To preserve standard ProActive library properties (i.e., allow legacy ProActive applications to be seamlessly ported to QCG)
- To provide end users with a consistent QCG-Broker (see further) Job Profile schema as a single document for describing application parameters required for execution as well as resource requirements (in particular, network topology and estimated execution time)
- To prevent end users from the necessity to have direct (i.e., over Secure Shell [SSH]) access to remote clusters and machines

9.3.2.1 Cross-Cluster Deployment and Communication In the QCG environment, additional services were required in order to support the spawning of parallel application processes on coallocated computational resources. The main reason for this was that standard deployment methodologies used in OpenMPI and ProActive relied on either RSH/SSH or specific local queuing functionalities. Both are limited to single-cluster runs (e.g., the SSH-based deployment methods are problematic if at least one cluster has worker nodes that have private IP addresses). Those services are called the coordinators and are implemented as Web services. Taking into account different existing cluster configurations, we may distinguish the following general situations:

1. **A Computing Cluster with Public IP Addresses.** Both the front end and the worker nodes have public IP addresses. Typically, a firewall is used to restrict access to internal nodes.
2. **A Computing Cluster with Private IP Addresses.** Only the front-end machine is accessible from the Internet; all the worker nodes have private IP addresses. Typically, NAT is used to provide outbound connectivity.

These cluster configuration types influence intercluster communication techniques supported in QCG, called port range and *proxy* respectively.

9.3.2.2 Port Range Technique The port range technique is a simple approach that makes the deployment of parallel environments firewall friendly. Most of the existing parallel environments use random ports by default to listen for incoming TCP/IP traffic. This makes cross-domain application execution almost impossible as most system administrators typically forbid to open all inbound ports to the Internet due to security reasons. By forcing the parallel

Figure 9.3 *A typical port range technique used by administrators of computational resources for a cross-cluster communication.*

environments to use only a predefined, unprivileged range of ports, it is much easier for administrators to configure the firewall in a way to allow incoming MPI and ProActive traffic without exposing critical system services to the Internet.

The port range technique is illustrated in Figure 9.3. Each of the site administrators has to choose a range of ports to be used (e.g., [5000–5100] for the parallel communication) and configure the firewall appropriately. One should note that the port range technique solves the problem of the cross-cluster connectivity for computing clusters where all worker nodes have public IP addresses.

Figure 9.4 *A new proxy technique for QCG-OMPI and QCG-ProActive based on SOCKS servers in QosCosGrid.*

9.3.2.3 Proxy Technique In the second category of clusters where worker nodes have private IP addresses, the port range technique is not sufficient as all the worker nodes are not addressable from the outside networks. Therefore, in addition to the port range technique, which only shapes incoming traffic, we adopted a new proxy technique. In our approach, additional SOCKS proxy services have to be deployed on front-end machines to route incoming traffic to the MPI and ProActive processes running inside clusters on local worker nodes. The new proxy-based technique is depicted in Figure 9.4.

9.4 QCG MIDDLEWARE

In a nutshell, the QCG middleware consists of two logical levels: grid level and AD level. Grid-level services control, schedule, and generally supervise the execution of end-user applications, which are spread over independent

ADs. The AD represents a single resource provider (e.g., HPC or data center) participating in a certain grid or cloud environment by sharing its computational resources with both local and external end users. The logical separation of ADs corresponds to the fact that they are owned by different institutions or resource providers. Each institution contributes its resources for the benefit of the entire grid or cloud while controlling its own AD and own resource allocation/sharing policies. All involved organizations agree to connect their resource pools exposed by AD-level services to a trusted upper-level middleware, in this case, QCG middleware. Based on these assumptions, QCG middleware tries to achieve optimal resource utilization and to ensure the requested level of quality of service for all the end users. The key component of every AD in QCG is the QCG-Computing service, which provides remote access to queuing system resources. The QCG-Computing service supports advance reservation, parallel execution environments—OpenMPI and ProActive, with coordinators being responsible for the synchronization of cross-cluster executions—and data transfer services for managing input and output data. Another relevant service at the AD level is in charge of notification mechanisms: It is called QCG-Notification. All AD-level services are tightly integrated and connected to the grid-level services in QCG. There are two critical services at the grid-level: the QCG-Broker, which is a metascheduling framework controlling executions of applications on the top of queuing systems via QCG-Computing services, and the Grid Authorization Service (GAS), which offers dynamic, fine-grained access control and enforcement for shared computing services and resources. From an architectural perspective, GAS can also be treated as a trusted single logical point for defining security policies. The overall QCG architecture is depicted in Figure 9.5.

9.4.1 QCG-Computing Service

QCG-Computing (the successor of the SMOA Computing and OpenDSP projects) is an open architecture implementation of the SOAP Web service for multiuser access and policy-based job control routines by various queuing and batch systems managing local computational resources. This key service is using Distributed Resource Management Application API (DRMAA) to communicate with the underlying queuing systems (Troger et al., 2007). QCG-Computing has been designed to support a variety of plug-ins and modules for external communication as well as to handle a large number of concurrent requests from external clients and services, in particular, QCG-Broker and GAS. Consequently, it can be used and integrated with various authentication, authorization, and accounting services or to extend capabilities of existing e-infrastructures based on UNICORE, gLite, Globus Toolkit, and others. QCG-Computing service is compliant with the Open Grid Forum (OGF) HPC Basic Profile specification, which serves as a profile over OGF standards like Job Submission Description Language (JSDL) and Open Grid Services Architecture (OGSA) Basic Execution Service (OGF, 2007). In addition, it

Figure 9.5 *The overall QosCosGrid architecture and its main services supporting QCG-OMPI and QCG-ProActive.*

offers remote interfaces for advance reservation management and supports basic file transfer mechanisms. QCG-Computing was successfully tested with the following queuing systems: Sun Grid Engine (SGE), Platform LSF, Torque/Maui, PBS Pro, Condor, and Apple XGrid. Therefore, as a crucial component in QCG, it can be easily set up on the majority of computing clusters and supercomputers running the aforementioned queuing systems. Currently, advance reservation capabilities in QCG-Computing are exposed for SGE, Platform LSF, and Maui (a scheduler that is typically used in conjunction with Torque). Moreover, generic extensions for advance reservation have been proposed for the next DRMAA standard release.

9.4.2 QCG-Notification and Data Movement Services

QCG-Notification (the successor of SMOA Notification) is an open source implementation of the family of WS-Notification standards (version 1.3) (OASIS Standards, 2008). In the context of QCG, it is used to extend features provided by QCG-Computing by adding standards-based synchronous and asynchronous notification features. QCG-Notification supports the topic-based publish/subscribe pattern for asynchronous message exchange among Web services and other entities, in particular, services or clients that want to be integrated with QCG. The main part of QCG-Notification is based on a highly efficient, extended version of the Notification Broker, managing all items participating in notification events. Today, QCG-Notification offers sophisticated notification capabilities, for example, topic and message content notification

TABLE 9.1 Functional Comparison of Leading Notification Frameworks

	QCG-Notification	ServiceMix	WebSphere	GT 4.x
Language	Ansi C	Java	Java	C/Java
Type	Brokered	Brokered	Brokered	Base
Topic namespaces	Yes	No	Yes	Yes (flat only)
Dialect	Full	Simple	Full	Simple
Dynamic topics	Yes*)	Yes	Yes	Yes
Message filters	Yes	Yes	Yes	No
Pull points	Yes	Yes	Yes	No
Core functions	QCG-Core, XMPP	JBI services Bus JMS JMS	Enterprise Service bus	Java WS-Core C WS-Core

filtering and pull-and-push styles of transporting messages. QCG-Notification has been integrated with a number of communication protocols as well as various Web service security mechanisms. The modular architecture of QCG-Notification makes it relatively straightforward to develop new extensions and plug-ins to meet new requirements. In QCG, for example, it was used for brokering notification messages about the job state changes linking QCG-Broker and QCG-Computing. All instances of the QCG-Computing service act as information *producers*, while the QCG-Broker service is the *consumer* of job notifications in QCG. More sophisticated configurations of QCG-Notification with both QCG middleware services and external entities based on service-oriented architecture patterns are also possible. Table 9.1 presents the result of a functional comparison between QCG-Notification and other popular notification frameworks, namely, Apache ServiceMix (version 3.3.1), IBM WebSphere (version 7.0) and Globus Toolkit (version 4.2). A set of analyzed functional features covers most of the WS-Notification concepts, and it allows us to highlight fundamental differences among existing standards-compliant notification systems.

As many other e-infrastructures controlled by middleware services, QCG takes advantage of the GridFTP protocol for large data transfer operations, in particular, to stage in and stage out files for advanced simulations. GridFTP is a high-performance, secure, reliable data transfer protocol optimized for high-bandwidth wide area networks. It is a de facto standard for all data transfers in grid and cloud environments and extends the standard FTP protocol with functions such as third-party transfer, parallel and striped data transfer, self-tuning capabilities, X509 proxy certificate-based security, and support for reliable and restartable data transfers. The development of GridFTP is coordinated by the GridFTP Working Group under the hood of the OGF community.

9.4.3 QCG-Broker Service

QCG-Broker (formerly named GRMS) was designed to be an open source metascheduling framework that allows developers to build and easily deploy

resource management systems to control large-scale distributed computing infrastructures running queuing or batch systems locally. Based on dynamic resource selection, advance reservation and various scheduling methodologies, combined with feedback control architecture, QCG-Broker deals efficiently with various metascheduling challenges, for example, coallocation, load balancing among clusters, remote job control, file staging support, or job migration (Kurowski et al., 2004). The main goal of QCG-Broker was to manage the whole process of remote job submission and advance reservation to various batch queuing systems and subsequently to underlying clusters and computational resources. It has been designed as an independent core component for resource management processes that can take advantage of various low-level core and grid services and existing technologies, such as QCG-Computing, QCG-Notification, or GAS, as well as various grid middleware services such as gLite, Globus, or UNICORE. Addressing various demanding computational needs of large-scale complex simulations, which in many cases can exceed capabilities of a single cluster, the QCG-Broker can flexibly distribute and control applications onto many computing clusters or supercomputers on behalf of end users. Moreover, owing to some built-in metascheduling procedures, it can optimize and run efficiently a wide range of applications while at the same time increasing the overall throughput of computing e-infrastructures. Advance reservation mechanisms are used to create, synchronize, and simultaneously manage the coallocation of computing resources located at different ADs. The XML-based job definition language Job Profile makes it relatively easy to specify the requirements of large-scale parallel applications together with the complex parallel communication topologies. Consequently, application developers and end users are able to run their experiments in parallel over multiple clusters.

Defined communication topologies may contain definitions of groups of MPI or ProActive processes with resource requirements, using resource and network attributes for internal and external group-to-group communication. Therefore, various application-specific topologies such as master–slave, all-to-all, or ring are supported in the Job Profile language. The Job Profile language has been adopted for complex system modeling and simulation purposes elsewhere (see, e.g., Chapter 8 of this volume or the work reported by Agullo et al., 2010).

To meet the requirements of complex scenarios consisting of many cooperating and possibly concurrent applications, for example, exchanging steering parameters in multiscale simulation studies, QCG-Broker is able to deal with complex applications defined as a set of tasks with precedence relationships (workflows). The workflow model built into QCG-Broker is based on direct acyclic graphs. In this approach, an end user specifies in advance precedence constraints of a task in the form of *task–state relationships*. What differentiates QCG-Broker from other middleware services supporting workflows is that every single task can be connected not only with input

TABLE 9.2 Comparison of QCG-Broker with Other Leading Brokering and Scheduling Services

Feature/System	QCG-Broker	Moab Grid Suites	CSF4	GUR	HARC
Negotiation protocol	Enhancement 1-phase commit	—	None	Reserve Cancel	Paxos
Economic support	Yes	No	No	No	No
Coallocation	Yes	Yes	No	Yes	Yes
Topology-aware coallocation	Yes	No	No	No	No
Scheduling on time axis	Yes	Yes	No	No	No
Local schedulers support	LSF Torque PBSPro SGE SLURM OpenPBS	Torque PBSPro LSF SGE SLURM OpenPBS LoadLeveler	LSF PBS Condor SGE	Catalina	LoadLeveler PBSPro LSF Torque
DRMAA support	Yes	Yes	No	No	No
Open source code	Yes	No	Apache	GPL	OpenSource
OpenDSP support	Yes	No	No	No	No

or output files but may also be triggered by predefined conditional rules or by the status of one or more jobs or tasks. Additionally, QCG-Broker supports parameter sweeps and allows to start in a single call multiple instances of the same application with different sets of arguments. For each task in the collection, the value of one or more of the task parameters may be changed in some predefined fashion, thus creating a parameter space. This is a very useful feature and gives the end user an easy way to search the parameter space for the concrete set of parameters that meet the defined criteria. QCG-Broker is a unique feature to allow end users to define multidimensional parameter spaces. Moreover, QCG-Broker has been successfully integrated with the Grid Scheduling SIMulator (GSSIM) to perform advanced metascheduling optimization, tuning, or reconfiguration experiments (Kurowski et al., 2007). Table 9.2 depicts the main features of QCG-Broker and compares these to other metascheduling frameworks deployed in e-infrastructures.

9.5 ADDITIONAL QCG TOOLS

9.5.1 Eclipse Parallel Tools Platform (PTP) for QCG

The PTP is intended to address a major deficiency in the development of parallel programs, namely, the lack of a robust open source targeted

Figure 9.6 *Eclipse PTP plug-in for parallel application development and debugging integrated with QosCosGrid.*

development environment with tools that assist in software development for parallel applications. The PTP offers a variety of useful features for parallel application developers, in particular, a parallel integrated development environment, a scalable debugger, integration with parallel tools, and interaction with parallel systems. Currently, the main supported languages are C/C++ and Fortran. We used the Eclipse PTP framework to support QCG-OMPI and remote job submission, debugging, and monitoring using the QCG middleware, in particular, QCG-Computing services.

In order to connect to QCG-Computing services, it is necessary to provide connection data. These are inserted by the user via a graphical user interface (GUI) wizard page from the QCG RM plug-in. As the wizard is accepted, the data reach an object from class `QCGServiceProvider` responsible for keeping the data persistent. When the Eclipse PTP requests a connection name for the first time (which it is going to use for the connection), a new instance from the `QCGConnection` class is created and registered at `QCGConnectionManager`. This object contains the Web service ports for `rsync`, staging and activity management. Once the user requests to start the Resource Manager, classes `QCGResourceManager` and `QCGRuntimeSystem` are created and certain methods are called. These classes obtain basic data about the system (by calling `getFactoryAttributes()`) and display them in Eclipse's *parallel run-time* perspective. A generic architecture of a new Eclipse PTP plug-in available for QCG users and developers is depicted in Figure 9.6.

9.6 QOSCOSGRID SCIENCE GATEWAYS

The advanced Web-based graph- and multimedia-oriented user interfaces designed for scientists and engineers could change the way end users collaborate, deal with advanced simulations, share results, and work together to solve challenging problems. Moreover, future science and engineering gateways will influence the way end users will access not only their data but also control and monitor demanding computing simulations over the Internet. To allow end users to interact remotely with future supercomputers and large-scale comput-

ing environments in a more visual manner, we developed a Web tool called Vine Toolkit. Russell et al. (2008) demonstrated that this tool can be used as a core Web platform for various science gateways integrated with various e-infrastructures based on UNICORE, gLite, or Globus Toolkit middleware services. The Vine Toolkit is a modular, extensible, and easy-to-use tool as well as a high-level application program interface for various applications, visualization components, and building blocks to allow interoperability between a wide range of grid and supercomputing technologies. Similar to stand-alone GUIs, it supports Adobe Flex and BlazeDS technologies, allowing the creation of advanced Web applications. Additionally, the Vine Toolkit has been integrated with well-known open source Web frameworks, such as Liferay and GridSphere. Using the enhanced version of Vine Toolkit, we created a new Science Gateway called QCG Gateway. The QosCosGrid Science Gateway consists of a general part showing and monitoring computational resource characteristics as well as a set of domain-specific Web applications developed for certain complex system use cases. With these tools, end users are able to use only Web browsers to create and submit their complex simulations, monitor their progress, and access and analyze the results generated (Fig. 9.7).

The QCG e-infrastructure has been deployed on a large number of computational resources provided by various research centers, such as the Poznan Supercomputing and Networking Center (Poland), National Institute for Research in Computer Science and Control (INRIA) (France), and Dortmund University of Technology (Germany). A number of complex systems scientists

Figure 9.7 *Example complex system simulations executed and controlled via the Web-based QosCosGrid Science Gateway.*

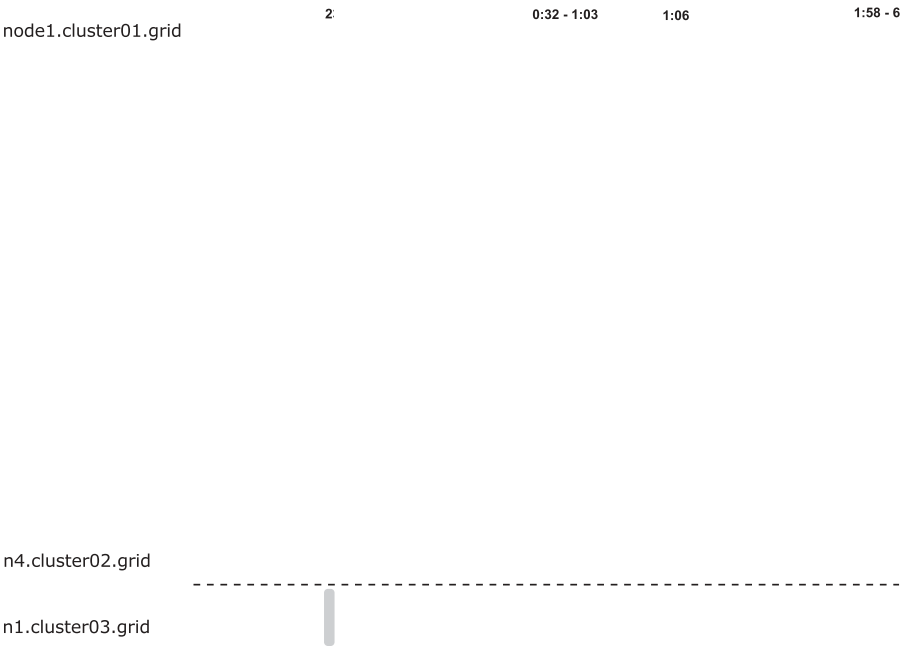

Figure 9.8 *Example coallocations using advance reservation QosCosGrid capabilities for cross-cluster parallel simulations.*

and modelers have performed various tests and benchmarks using QCG solutions. The QCG Gateway displays monitoring and resource information about the underlying infrastructure and provides a feedback not only for end users but also for resource owners and administrators (Fig. 9.8). Various monitoring tests are invoked periodically and their results are presented as graphical maps and diagrams in a Web browser. The first two maps show results of bidirectional tests of cross-domain QCG-OMPI and QCG-ProActive applications, measuring bandwidth and latency among front-end machines of computing clusters in different ADs connected via the Internet. We have also added useful Gantt charts displaying local and cross-domain job execution information as well as information on advance reservation and coallocation of computing resources. Both the test-bed monitoring and GUI layers were developed using Adobe Flex-based components.

9.7 DISCUSSION AND RELATED WORK

Existing grid e-infrastructures, such as TeraGrid in the United States and CERN's Enabling Grids for E-sciencE (EGEE)/LCG in Switzerland, Grid'5000 in France, D-Grid in Germany, or PL-GRID in Poland, provide thousands of

computational resources, offering facilities similar to large-scale parallel computing production environments. One of the critical components for the execution of advanced applications in such environments is the *metascheduler*. Its main role is to schedule complex applications that end users wish to run on available computational resources and to guarantee "topology awareness." Topology awareness implies that matching of the resource offers and requests must take into account not only the computational characteristics of the resources but also their interconnections. To the best of our knowledge, these kinds of requirements have not been addressed by any of the existing metaschedulers. Thus, the majority of the up-to-date computing e-infrastructures support relatively easy metascheduling strategies, in particular, easy matching-based techniques, without advance reservation and coallocation of computational resources. Typical systems include the EGEE Workload Management System, eNANOS, and GridWay. The relatively simple metascheduling approach adopted by these systems is sufficient to execute "embarrassingly parallel" applications, such as parameter sweeps or MPI applications, within a *single* computing cluster. This approach limits end users to the computing power of one computing cluster, which may not be sufficient to perform more sophisticated complex system simulations. For cross-cluster MPI and ProActive applications, the requested and offered network topology cannot be ignored, meaning that topology-aware scheduling, together with coallocation capabilities, must be employed. These features have been successfully implemented and integrated to provide a new QCG e-infrastructure for complex system modelers, scientists, and end users.

Emerging e-infrastructures offer new quality-of-service capabilities as well as new hybrid programming and execution environments such as MPI/OpenMP and MPI/CUDA parallel hybrids. Taking advantage of the results presented in this chapter, the adaptation of QCG solutions to many-core systems would be an interesting area of our future research. Another natural direction of future research and development would be to explore new optimization criteria, for example, energy consumption or heating, as these have become critical in large distributed computing environments. As virtualization techniques of operating systems and embedded applications are maturing, another fruitful avenue of future research revolves around efficient resource management techniques using the virtual machine concept. Moreover, the recent technological shift from the area of multicore to many-core systems and systems-on-the-chip introduced another scheduling layer at the operating system level. Future large-scale computing systems will have to deal with a hierarchy of complex static and reconfigurable computing structures, and new resource management techniques need to be invented to control such systems on a very low level of granularity.

Ethernet technology has been the dominant data link protocol in local area networks for many years, and it is widely offered as a customer service, but it was not designed for large carrier-scale transport networks. However, because of its suitability for data and multimedia applications, its flexibility, and its

widespread use, many carriers now consider Ethernet as a potential convergence solution for next-generation networks. Research grid networks link distributed computing resources to perform highly demanding computations needing vast processing capability. To make optimum use of the computing resources, high-capacity, low-latency connections are needed, and complex optimization algorithms need to calculate the optimal exploitation of the computing resources, taking into account factors such as the data rate offered by the connecting links and their latency.

Currently, QCG is extensively tested by its developers and by external research communities interested in new computing e-infrastructures for distributed multiscale simulations across disciplines. Driven by seven challenging applications from five representative scientific domains (fusion, clinical decision making, systems biology, nanoscience, and engineering), they will deploy a computational science environment for distributed multiscale computing under the MAPPER[4] project based on QCG technologies. Various extensions proposed to QCG will result in high-quality components for today's e-infrastructures in Europe by enabling distributed execution of two modes (loosely and tightly coupled) of multiscale computing in a user-friendly and transparent way.

REFERENCES

E. Agullo, C. Coti, J. Dongarra, et al. QR factorization of tall and skinny matrices in a grid computing environment. In *Proc. of the 24th IEEE Int'l Parallel and Distributed Processing Symposium*, 2010.

E. Agullo, C. Coti, T. Herault, et al. QCG-OMPI: MPI applications on grids. *Future Generation Computer Systems*, 27(4):357–369, 2011.

L. Baduel, G. Baude, D. Caromel, et al. Programming, deploying, composing, for the grid. In J. C. Cunha and O. F. Rana, editors, *Grid Computing: Software Environments and Tools*, pp. 205–229, London: Springer-Verlag, 2006.

S. T. Barnard. PMRSB: Parallel multilevel recursive spectral bisection. In *Proc. of the 1995 ACM/IEEE Conference on Supercomputing, Supercomputing '95*, New York: ACM, 1995.

C. Coti, T. Herault, S. Peyronnet, et al. Grid services for MPI. In ACM/IEEE, editor, *Proc. of the 8th IEEE Int'l Symposium on Cluster Computing and the Grid (CCGrid'08)*, pp. 417–424, Lyon, France: IEEE Computer Society, 2008.

P.-O. Fjallstrom. Algorithms for graph partitioning: A survey. *Linkoping Electronic Articles in Computer and Information Science*, 3(10):1–34, 1998.

L. Gulyás, G. Szemes, G. Kampis, et al. A modeler-friendly API for ABM partitioning. In *Proc. of the ASME 2009 Conference*, San Diego, California, USA, 2008.

K. Kurowski, W. de Back, W. Dubitzky, et al. *Complex System Simulations with QosCosGrid*, Vol. 5544, pp. 387–396, Berlin and Heidelberg: Springer, 2009.

[4] http://www.mapper-project.eu.

K. Kurowski, B. Ludwiczak, J. Nabrzyski, et al. Dynamic grid scheduling with job migration and rescheduling in the GridLab resource management system. *Scientific Programming*, 12:263–273, 2004.

K. Kurowski, J. Nabrzyski, A. Oleksiak, et al. Grid scheduling simulations with GSSIM. In *Proc. of the 13th Int'l Conference on Parallel and Distributed Systems—Vol. 2*, pp. 1–8, Washington, DC: IEEE Computer Society, 2007.

K. Kurowski, T. Piontek, P. Kopta, et al. Parallel large-scale simulations in the pl-grid environment. In M. Stroinski et al., editors, *Computational Methods in Science and Technology*, pp. 47–56, Poznan, Poland, 2010.

M. Mazzariol, B. A. Gennart, and R. D. Hersch. Dynamic load balancing of parallel cellular automata, 2000.

OASIS Standards. OASIS web services notification technical committee, 2008. URL http://www.oasis-open.org.

OGF. Open grid forum, 2007. http://ogf.org.

M. Russell, P. Dziubecki, P. Grabowski, et al. The Vine Toolkit: A Java framework for developing grid applications. In R. Wyrzykowski et al., editors, *Parallel Processing and Applied Mathematics, volume 4967 of Lecture Notes in Computer Science*, pp. 331–340, Berlin and Heidelberg: Springer, 2008.

M. Scheutz and P. Schermerhorn. Adaptive algorithms for the dynamic distribution and parallel execution of agent-based models. *Journal of Parallel Distributed Computing*, 66:1037–1051, 2006.

P. M. A. Sloot and A. G. Hoekstra. Multi-scale modelling in computational biomedicine. *Briefings in Bioinformatics*, 11(1):142–152, 2010.

P. Troger, H. Rajic, A. Haas, et al. Standardization of an API for distributed resource management systems. In *Proc. of the 7th IEEE Int'l Symposium on Cluster Computing and the Grid, CCGRID'07*, pp. 619–626, Washington, DC: IEEE Computer Society, 2007.